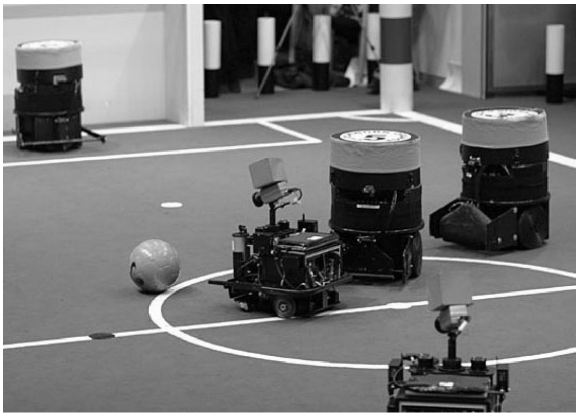
A Concise Introduction to

Multiagent Systems and Distributed AI

Nikos Vlassis

Intelligent Autonomous Systems
Informatics Institute
University of Amsterdam



Universiteit van Amsterdam

Contents

Preface **iii**

1 Introduction **1**
1.1 Multiagent systems and distributed AI 1
1.2 Characteristics of multiagent systems 1
1.3 Applications 4
1.4 Challenging issues 5
1.5 Notes and further reading 6

2 Rational agents **7**
2.1 What is an agent? 7
2.2 Agents as rational decision makers 8
2.3 Observable worlds and the Markov property 8
2.4 Stochastic transitions and utilities 10
2.5 Notes and further reading 14

3 Strategic games **15**
3.1 Game theory 15
3.2 Strategic games 16
3.3 Iterated elimination of strictly dominated actions 18
3.4 Nash equilibrium 19
3.5 Notes and further reading 22

4 Coordination **23**
4.1 Distributed decision making 23
4.2 Coordination games 24
4.3 Social conventions 25
4.4 Roles 26
4.5 Coordination graphs 28
4.6 Notes and further reading 31

5 Common knowledge **33**
5.1 Thinking interactively 33
5.2 The puzzle of the hats 34

5.3 Partial observability and information 34
5.4 A model of knowledge 37
5.5 Knowledge and actions 38
5.6 Notes and further reading 40

6 Communication **41**
6.1 Communicating agents 41
6.2 Communicative acts 42
6.3 The value of communication 43
6.4 Coordination via communication 45
6.5 Notes and further reading 48

7 Mechanism design **49**
7.1 Self-interested agents 49
7.2 The mechanism design problem 50
7.3 The revelation principle 53
7.4 The Groves-Clarke mechanism 54
7.5 Notes and further reading 55

8 Learning **57**
8.1 Reinforcement learning 57
8.2 Markov decision processes 58
8.3 Multiagent reinforcement learning 60
8.4 Exploration policies 62
8.5 Notes and further reading 63

Preface

This text contains introductory material on the subject of Multiagent Systems and Distributed AI. It has been used as lecture notes for 3rd and 4th year courses at the Informatics Institute of the University of Amsterdam, The Netherlands. An on-line version can be found at

`http://www.science.uva.nl/~vlassis/cimasdai/`

with a link to accompanying software. I would be most grateful to receive any kind of feedback on this text, concerning errors, ideas for improvement, etc. You may contact me at

`vlassis@science.uva.nl`

I would like to thank the following people for providing useful comments on the text: Taylan Cemgil, Jelle Kok, Jan Nunnink, Matthijs Spaan.

Nikos Vlassis
Amsterdam, September 2003

Version: 14 October 2003

Chapter 1

Introduction

In this chapter we give a brief introduction to multiagent systems, discuss their differences with single-agent systems, and outline possible applications and challenging issues for research.

1.1 Multiagent systems and distributed AI

The modern approach to **artificial intelligence** (AI) is centered around the concept of a **rational agent**. An agent is anything that can perceive its environment through sensors and act upon that environment through actuators (Russell and Norvig, 2003). An agent that always tries to optimize an appropriate performance measure is called a rational agent. Such a definition of a rational agent is fairly general and can include human agents (having eyes as sensors, hands as actuators), robotic agents (having cameras as sensors, wheels as actuators), or software agents (having a graphical user interface as sensor and as actuator). From this perspective, AI can be regarded as the study of the principles and design of artificial rational agents.

However, agents are seldom stand-alone systems. In many situations they coexist and interact with other agents in several different ways. Examples include software agents on the Internet, soccer playing robots (see Fig. 1.1), and many more. Such a system that consists of a group of agents that can potentially interact with each other is called a **multiagent system** (MAS), and the corresponding subfield of AI that deals with principles and design of multiagent systems is called **distributed AI**.

1.2 Characteristics of multiagent systems

What are the fundamental aspects that characterize a MAS and distinguish it from a single-agent system? One can think along the following dimensions:

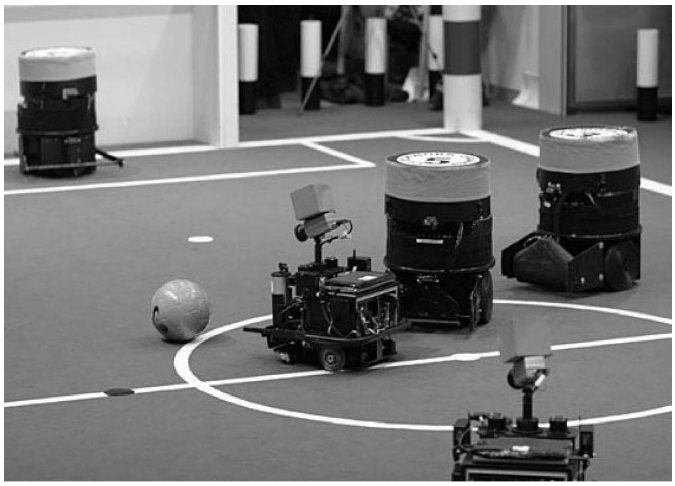


Figure 1.1: A robot soccer team is an example of a multiagent system.

Agent design

It is often the case that the various agents that comprise a MAS are designed in different ways. A typical example is software agents, also called **softbots**, that have been implemented by different people. In general, the design differences may involve the hardware (for example soccer robots based on different mechanical platforms), or the software (for example software agents running different operating systems). We often say that such agents are **heterogeneous** in contrast to **homogeneous** agents that are designed in an identical way and have a priori the same capabilities. However, this distinction is not clear-cut; agents that are based on the same hardware/software but implement different behaviors can also be called heterogeneous. Agent heterogeneity can affect all functional aspects of an agent from perception to decision making, while in single-agent systems the issue is simply nonexistent.

Environment

Agents have to deal with environments that can be either **static** (time-invariant) or **dynamic** (nonstationary). Most existing AI techniques for single agents have been developed for static environments because these are easier to handle and allow for a more rigorous mathematical treatment. In a MAS, the mere presence of multiple agents makes the environment appear dynamic from the point of view of each agent. This can often be problematic, for instance in the case of concurrently **learning** agents where non-stable behavior can be observed. There is also the issue which parts of a dynamic environment an agent should treat as other agents and which not. We will discuss some of these issues in chapter 8.

Perception

The collective information that reaches the sensors of the agents in a MAS is typically **distributed**: the agents may observe data that differ spatially (appear at different locations), temporally (arrive at different times), or even semantically (require different interpretations). This automatically makes the world state **partially observable** to each agent, which has various consequences in the decision making of the agents. An additional issue is **sensor fusion**, that is, how the agents can optimally combine their perceptions in order to increase their collective knowledge about the current state. We will discuss distributed perception and its consequences in chapter 5.

Control

Contrary to single-agent systems, the **control** in a MAS is typically distributed (decentralized). This means that there is no central process that collects information from each agent and then decides what action each agent should take. The decision making of each agent lies to a large extent within the agent itself. The general problem of multiagent decision making is the subject of **game theory** which we will cover in chapter 3. In a **cooperative** or **team** MAS[1], distributed decision making results in asynchronous computation and certain speedups, but it also has the downside that appropriate **coordination** mechanisms need to be additionally developed. Coordination ensures that the individual decisions of the agents result in good joint decisions for the group. Chapter 4 is devoted to the topic of coordination.

Knowledge

In single-agent systems we typically assume that the agent knows its own actions but not necessarily how the world is affected by its actions. In a MAS, the levels of **knowledge** of each agent about the current world state can differ substantially. For example, in a team MAS involving two homogeneous agents, each agent may know the available action set of the other agent, both agents may know (by communication) their current perceptions, or they can infer the intentions of each other based on some shared prior knowledge. On the other hand, an agent that observes an adversarial team of agents will typically be unaware of their action sets and their current perceptions, and might also be unable to infer their plans. In general, in a MAS each agent must also consider the knowledge of each other agent in its decision making. A crucial concept here is that of **common knowledge**, according to which every agent knows a fact, every agent knows that every

[1] We will interchangeably use the terms 'cooperative' or 'team' MAS to refer to agents that share the same interests. We note, however, that the game-theoretic use of the term 'cooperative' is different, referring to agents that are freely allowed to communicate and enforce agreements prior to taking decisions (Harsanyi and Selten, 1988).

other agent knows this fact, and so on. We will discuss common knowledge in detail in chapter 5.

Communication

Interaction is often associated with some form of **communication**. Typically we view communication in a MAS as a two-way process, where all agents can potentially be senders and receivers of messages. Communication can be used in several cases, for instance, for coordination among cooperative agents or for **negotiation** among **self-interested** agents (we will discuss the latter case in some detail in chapter 7). Moreover, communication additionally raises the issues of what network protocols to use in order for the exchanged information to arrive safely and timely, and what **language** the agents must speak in order to understand each other (especially if they are heterogeneous). We will address communication in chapter 6.

1.3 Applications

Just as with single-agent systems in traditional AI, it is difficult to anticipate the full range of applications where MASs can be used. Some applications have already appeared, especially in software engineering where MAS technology is viewed as a novel and promising software building paradigm. A complex software system can be treated as a collection of many small-size autonomous agents, each with its own local functionality and properties, and where interaction among agents enforces total system integrity. Some of the benefits of using MAS technology in large software systems are (Sycara, 1998):

- Speedup and efficiency, due to the asynchronous and parallel computation.
- Robustness and reliability, in the sense that the whole system can undergo a 'graceful degradation' when one or more agents fail.
- Scalability and flexibility, since it is easy to add new agents to the system.
- Cost, assuming that an agent is a low-cost unit compared to the whole system.
- Development and reusability, since it is easier to develop and maintain a modular software than a monolithic one.

A very challenging application domain for MAS technology is the Internet. Today the Internet has developed into a highly distributed **open** system

where heterogeneous software agents come and go, there are no well established protocols or languages on the 'agent level' (higher than TCP/IP), and the structure of the network itself keeps on changing. In such an environment, MAS technology can be used to develop agents that act on behalf of a user and are able to negotiate with other agents in order to achieve their goals. Auctions on the Internet and electronic commerce are such examples (Noriega and Sierra, 1999; Sandholm, 1999). One can also think of applications where agents can be used for distributed data mining and information retrieval.

MASs can also be used for traffic control where agents (software or robotic) are located in different locations, receive sensor data that are geographically distributed, and must coordinate their actions in order to ensure global system optimality (Lesser and Erman, 1980). Other applications are in social sciences where MAS technology can be used for simulating interactivity and other social phenomena (Gilbert and Doran, 1994), in robotics where a frequently encountered problem is how a group of robots can localize themselves within their environment (Roumeliotis and Bekey, 2002), and in virtual reality and computer games where the challenge is to build agents that exhibit intelligent behavior (Terzopoulos, 1999).

Finally, an application of MASs that has recently gained popularity is robot soccer. There, teams of real or simulated autonomous robots play soccer against each other (Kitano et al., 1997). Robot soccer provides a testbed where MAS algorithms can be tested, and where many real-world characteristics are present: the domain is continuous and dynamic, the behavior of the opponents may be difficult to predict, there is uncertainty in the sensor signals, etc.

1.4 Challenging issues

The transition from single-agent systems to MASs offers many potential advantages but also raises challenging issues. Some of these are:

- How to decompose a problem, allocate subtasks to agents, and synthesize partial results.
- How to handle the distributed perceptual information. How to enable agents to maintain consistent shared models of the world.
- How to implement decentralized control and build efficient coordination mechanisms among agents.
- How to design efficient multiagent planning and learning algorithms.
- How to represent knowledge. How to enable agents to reason about the actions, plans, and knowledge of other agents.

- How to enable agents to communicate. What communication languages and protocols to use. What, when, and with whom should an agent communicate.
- How to enable agents to negotiate and resolve conflicts.
- How to enable agents to form organizational structures like teams or coalitions. How to assign roles to agents.
- How to ensure coherent and stable system behavior.

Clearly the above problems are interdependent and their solutions may affect each other. For example, a distributed planning algorithm may require a particular coordination mechanism, learning can be guided by the organizational structure of the agents, and so on. In the following chapters we will try to provide answers to some of the above questions.

1.5 Notes and further reading

The review articles of Sycara (1998) and Stone and Veloso (2000) provide concise and readable introductions to the field. The textbooks of Huhns (1987), Singh (1994), O'Hare and Jennings (1996), Ferber (1999), Weiss (1999), and Wooldridge (2002) offer more extensive treatments, emphasizing different AI and software engineering aspects. A website on multiagent systems is: `www.multiagent.com`

Chapter 2

Rational agents

In this chapter we describe what a rational agent is, we investigate some characteristics of an agent's environment like observability and the Markov property, and we examine what is needed for an agent to behave optimally in an uncertain world where actions do not always have the desired effects.

2.1 What is an agent?

Following Russell and Norvig (2003), an **agent** is anything that can be viewed as perceiving its **environment** through sensors and acting upon that environment through actuators.[1] Examples include humans, robots, or software agents. We often use the term **autonomous** to refer to an agent whose decision making relies to a larger extent on its own perception than to prior knowledge given to it at design time.

In this chapter we will study the problem of **optimal decision making** of an agent. That is, how an agent can choose the best possible action at each time step, given what it knows about the world around it. We will say that an agent is **rational** if it always selects an action that optimizes an appropriate **performance measure**, given what the agent knows so far. The performance measure is typically defined by the user (the designer of the agent) and reflects what the user expects from the agent in the task at hand. For example, a soccer robot must act so as to maximize the chance of scoring for its team, a software agent in an electronic auction must try to minimize expenses for its designer, and so on. A rational agent is also called an **intelligent** agent.

In the following we will mainly focus on **computational** agents, that is, agents that are explicitly designed for solving a particular task and are implemented on some computing device.

[1] In this chapter we will use 'it' to refer to an agent, to emphasize that we are talking about computational entities.

2.2 Agents as rational decision makers

The problem of optimal decision making of an agent was first studied in **optimal control** (Bellman, 1961). For the purpose of our discussion, we will assume a discrete set of time steps $t = 1, 2, \ldots$, in each of which the agent must choose an action a_t from a finite set of actions A that it has available. Intuitively, in order to act rationally, an agent should take both the *past* and the *future* into account when choosing an action. The past refers to what the agent has perceived and what actions it has taken until time t, and the future refers to what the agent expects to perceive and do after time t.

If we denote by o_τ the perception of an agent at time τ, then the above implies that in order for an agent to optimally choose an action at time t, it must in general use its complete **history** of perceptions o_τ and actions a_τ for $\tau \leq t$. The function

$$\pi(o_1, a_1, o_2, a_2, \ldots, o_t) = a_t \tag{2.1}$$

that in principle would require mapping the complete history of perception-action pairs up to time t to an optimal action a_t is called the **policy** of the agent.

As long as we can find a function π that implements the above mapping, the part of optimal decision making that refers to the past is solved. However, defining and implementing such a function is problematic; the complete history can consist of a very large (even infinite) number of perception-action pairs, which can vary from one task to another. Merely storing all perceptions would require very large memory, aside from the computational complexity for actually computing π.

This fact calls for simpler policies. One possibility is for the agent to ignore all its percept history except for the last perception o_t. In this case its policy takes the form

$$\pi(o_t) = a_t \tag{2.2}$$

which is a mapping from the current perception of the agent to an action. An agent that simply maps its current perception o_t to a new action a_t, thus effectively ignoring the past, is called a **reflex** agent, and its policy (2.2) is called **reactive** or **memoryless**. A natural question to ask now is: how successful such a reflex agent can be? As we will see next, for a particular class of environments a reflex agent can do pretty well.

2.3 Observable worlds and the Markov property

From the discussion above it is clear that the terms 'agent' and 'environment' are coupled, so that one cannot be defined without the other. In fact, the distinction between an agent and its environment is not always clear, and it

is sometimes difficult to draw a line between these two (Sutton and Barto, 1998, ch. 3).

To simplify things we will assume hereafter the existence of a **world** in which one or more agents are embedded, and in which they perceive, think, and act. The collective information that is contained in the world at any time step t, and that is relevant for the task at hand, will be called a **state** of the world and denoted by s_t. The set of all states of the world will be denoted by S. As an example, in a robot soccer game a world state can be characterized by the the soccer field layout, the positions and velocities of all players and the ball, what each agent knows about each other, and other parameters that are relevant to the decision making of the agents like the elapsed time since the game started, etc.

Depending on the nature of problem, a world can be either **discrete** or **continuous**. A discrete world can be characterized by a finite number of states. Examples are the possible board configurations in a chess game. On the other hand, a continuous world can have infinitely many states. For example, for a mobile robot that translates and rotates freely in a static environment and has coordinates (x, y, θ) with respect to a fixed Cartesian frame holds $S = \mathbb{R}^3$. Most of the existing AI techniques have been developed for discrete worlds, and this will be our main focus as well.

Observability

A fundamental property that characterizes a world from the point of view of an agent is related to the perception of the agent. We will say that the world is **(fully) observable** to an agent if the current perception o_t of the agent completely reveals the current state of the world, that is, $s_t = o_t$. On the other hand, in a **partially observable** world the current perception o_t of the agent provides only partial information about the current world state in the form of a conditional probability distribution $P(s_t|o_t)$ over states. This means that the current perception o_t does not fully reveal the true world state, but to each state s_t the agent assigns probability $P(s_t|o_t)$ that s_t is the true state (with $0 \leq P(s_t|o_t) \leq 1$ and $\sum_{s_t \in S} P(s_t|o_t) = 1$). Here we treat s_t as a random variable that can take all possible values in S.

Partial observability can in principle be attributed to two factors. First, it can be the result of **noise** in the agent's sensors. For example, due to sensor malfunction, the same state may 'generate' different perceptions to the agent at different points in time. That is, every time the agent visits a particular state it may perceive something different. Second, partial observability can be related to an inherent property of the environment referred to as **perceptual aliasing**: different states may produce identical perceptions to the agent at different time steps. In other words, two states may 'look' the same to an agent, although the states are different from each other. For example, two identical doors along a corridor will look exactly the same to

the eyes of a human or the camera of a mobile robot, no matter how accurate each sensor system is.

Partial observability is much harder to handle than full observability, and algorithms for optimal sequential decision making in a partially observable world can easily become intractable (Russell and Norvig, 2003, sec. 17.4). Partial observability is of major concern especially in multiagent systems where, as it will also be clearer in chapter 5, it may affect not only what each agent knows about the world state, but also what each agent knows about each other's knowledge. We will defer partial observability until chapter 5.

The Markov property

Let us consider again the case of a reflex agent with a reactive policy $\pi(o_t) = a_t$ in a fully observable world. The assumption of observability implies $s_t = o_t$, and therefore the policy of the agent reads

$$\pi(s_t) = a_t. \tag{2.3}$$

In other words, in an observable world the policy of a reflex agent is a mapping from world states to actions. The gain comes from the fact that in many problems the state of the world at time t provides a *complete description* of the history before time t. Such a world state that summarizes all the relevant information about the past in a particular task is said to be **Markov** or to have the **Markov property**. As we conclude from the above, in a Markov world an agent can safely use the memoryless policy (2.3) for its decision making, in place of the theoretically optimal policy function (2.1) that in principle would require very large memory.

So far we have discussed how the policy of an agent may depend on its past experience and the particular characteristics of the environment. However, as we argued at the beginning, optimal decision making should also take the future into account. This is what we are going to examine next.

2.4 Stochastic transitions and utilities

As mentioned above, at each time step t the agent chooses an action a_t from a finite set of actions A. When the agent takes an action, the world changes as a result of this action. A **transition model** (sometimes also called **world model**) specifies how the world changes when an action is executed. If the current world state is s_t and the agent takes action a_t, we can distinguish the following two cases:

- In a **deterministic** world, the transition model maps a state-action pair (s_t, a_t) to a single new state s_{t+1}. In chess for example, every move changes the configuration on the board in a deterministic manner.

- In a **stochastic** world, the transition model maps a state-action pair (s_t, a_t) to a probability distribution $P(s_{t+1}|s_t, a_t)$ over states. As in the partial observability case above, s_{t+1} is a random variable that can take all possible values in S, each with corresponding probability $P(s_{t+1}|s_t, a_t)$. Most real-world applications involve stochastic transition models, for example, robot motion is inaccurate because of wheel slip and other effects.

We saw in the previous section that sometimes partial observability can be attributed to uncertainty in the perception of the agent. Here we see another example where uncertainty plays a role; namely, in the way the world changes when the agent executes an action. In a stochastic world, the effects of the actions of the agent are not known a priori. Instead, there is a random element that decides how the world changes as a result of an action. Clearly, stochasticity in the state transitions introduces an additional difficulty in the optimal decision making task of the agent.

From goals to utilities

In classical AI, a **goal** for a particular task is a desired state of the world. Accordingly, **planning** is defined as a search through the state space for an optimal path to the goal. When the world is deterministic, planning comes down to a graph search problem for which a variety of methods exist, see for example (Russell and Norvig, 2003, ch. 3).

In a stochastic world, however, planning can not be done by simple graph search because transitions between states are nondeterministic. The agent must now take the uncertainty of the transitions into account when planning. To see how this can be realized, note that in a deterministic world an agent prefers by default a goal state to a non-goal state. More generally, an agent may hold **preferences** between any world states. For example, a soccer agent will mostly prefer to score, will prefer less (but still a lot) to stand with the ball in front of an empty goal, and so on.

A way to formalize the notion of state preferences is by assigning to each state s a real number $U(s)$ that is called the **utility** of state s for that particular agent. Formally, for two states s and s' holds $U(s) > U(s')$ if and only if the agent prefers state s to state s', and $U(s) = U(s')$ if and only if the agent is indifferent between s and s'. Intuitively, the utility of a state expresses the 'desirability' of that state for the particular agent; the larger the utility of the state, the better the state is for that agent. In the world of Fig. 2.1, for instance, an agent would prefer state d3 than state b2 or d2. Note that in a multiagent system, a state may be desirable to a particular agent and at the same time be undesirable to an other agent; in soccer, for example, scoring is typically unpleasant to the opponent agents.

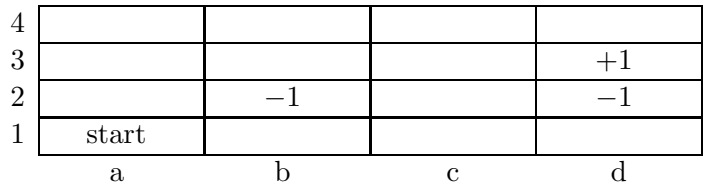


Figure 2.1: A world with one desired (+1) and two undesired (−1) states.

Decision making in a stochastic world

Equipped with utilities, the question now is how an agent can efficiently use them for its decision making. Let us assume that the world is stochastic with transition model $P(s_{t+1}|s_t, a_t)$, and is currently in state s_t, while an agent is pondering how to choose its action a_t. Let $U(s)$ be the utility of state s for the particular agent (we assume there is only one agent in the world). Utility-based decision making is based on the premise that the optimal action a_t^* of the agent should maximize **expected utility**, that is,

$$a_t^* = \arg\max_{a_t \in A} \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) U(s_{t+1}) \tag{2.4}$$

where we sum over all possible states $s_{t+1} \in S$ the world may transition to, given that the current state is s_t and the agent takes action a_t. In words, to see how good an action is, the agent has to multiply the utility of each possible resulting state with the probability of actually reaching this state, and sum up the resulting terms. Then the agent must choose the action a_t^* that gives the highest sum.

If each world state has a utility value, then the agent can do the above calculations and compute an optimal action for each possible state. This provides the agent with a policy that maps states to actions in an optimal sense. In particular, given a set of optimal (i.e., highest attainable) utilities $U^*(s)$ in a given task, the **greedy** policy

$$\pi^*(s) = \arg\max_{a} \sum_{s'} P(s'|s, a) U^*(s') \tag{2.5}$$

is an **optimal policy** for the agent.

There is an alternative and often useful way to characterize an optimal policy. For each state s and each possible action a we can define an optimal **action value** (or **Q-value**) $Q^*(s, a)$ that measures the 'goodness' of action a in state s for that agent. For this value holds $U^*(s) = \max_a Q^*(s, a)$, while an optimal policy can be computed as

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

4	0.818 (→)	0.865 (→)	0.911 (→)	0.953 (↓)
3	0.782 (↑)	0.827 (↑)	0.907 (→)	+1
2	0.547 (↑)	−1	0.492 (↑)	−1
1	0.480 (↑)	0.279 (←)	0.410 (↑)	0.216 (←)
	a	b	c	d

Figure 2.2: Optimal utilities and an optimal policy of the agent.

which is a simpler formula than (2.5) and moreover it does not require a transition model. In chapter 8 we will see how we can compute optimal utilities $U^*(s)$ and action values $Q^*(s, a)$ in a stochastic observable world.

Example: a toy world

Let us close the chapter with an example, similar to the one used in (Russell and Norvig, 2003, chap. 21). Consider the world of Fig. 2.1 where in any state the agent can choose one of the actions {*Up, Down, Left, Right*}. We assume that the world is fully observable (the agent always knows where it is), and stochastic in the following sense: every action of the agent to an intended direction succeeds with probability 0.8, but with probability 0.2 the agent ends up perpendicularly to the intended direction. Bumping on the border leaves the position of the agent unchanged. There are three terminal states, a desired one (the 'goal' state) with utility +1, and two undesired ones with utility −1. The initial position of the agent is a1.

We stress again that although the agent can perceive its own position and thus the state of the world, it cannot predict the effects of its actions on the world. For example, if the agent is in state c2, it knows that it is in state c2. However, if it tries to move *Up* to state c3, it may reach the intended state c3 (this will happen in 80% of the cases) but it may also reach state b2 (in 10% of the cases) or state d2 (in the rest 10% of the cases).

Assume now that optimal utilities have been computed for all states, as shown in Fig. 2.2. Applying the principle of maximum expected utility, the agent computes that, for instance, in state b3 the optimal action is *Up*. Note that this is the only action that avoids an accidental transition to state b2. Similarly, by using (2.5) the agent can now compute an optimal action for every state, which gives the optimal policy shown in parentheses.

Note that, unlike path planning in a deterministic world that can be described as graph search, decision making in stochastic domains requires computing a complete policy that maps states to actions. Again, this is a consequence of the fact that the results of the actions of an agent are unpredictable. Only after the agent has executed its action can it observe the new state of the world, from which it can select another action based on its precomputed policy.

2.5 Notes and further reading

We have mainly followed chapters 2, 16, and 17 of the book of Russell and Norvig (2003) which we strongly recommend for further reading. An illuminating discussion on the agent-environment interface and the Markov property can be found in chapter 3 of the book of Sutton and Barto (1998) which is another excellent text on agents and decision making, and is electronically available: `www-anw.cs.umass.edu/~rich/book/the-book.html`

Chapter 3

Strategic games

In this chapter we study the problem of **multiagent decision making**, where a group of agents coexist in an environment and take simultaneous decisions. We use game theory to analyze the problem, in particular, strategic games, where we examine two main solution concepts, iterated elimination of strictly dominated actions and Nash equilibrium.

3.1 Game theory

As we saw in chapter 2, an agent will typically be uncertain about the effects of its actions to the environment, and it has to take this uncertainty into account in its decision making. In a multiagent system, where many agents take decisions at the same time, an agent will also be uncertain about the decisions of the other participating agents. Clearly, what an agent should do depends on what the other agents will do.

Multiagent decision making is the subject of **game theory** (Osborne and Rubinstein, 1994). Although originally designed for modeling economical interactions, game theory has developed into an independent field with solid mathematical foundations and many applications. The theory tries to understand the behavior of interacting agents under conditions of uncertainty, and is based on two premises. First, that the participating agents are **rational**. Second, that they reason **strategically**, that is, they take into account the other agents' decisions in their decision making.

Depending on the way the agents choose their actions, we can distinguish two types of games. In a **strategic game**, each agent chooses his strategy only once at the beginning of the game, and then all agents take their actions simultaneously. In an **extensive game**, the agents are allowed to reconsider their plans during the game. Another distinction is whether the agents have **perfect** or **imperfect** information about aspects that involve the other agents. In this chapter we will only consider strategic games of perfect information.

3.2 Strategic games

A strategic game (also called game in **normal form**) is the simplest game-theoretic model of agent interactions. It can be viewed as a multiagent extension of the decision-theoretic model of chapter 2, and is characterized by the following elements:

- There are $n > 1$ agents in the world.[1]
- Each agent i can choose an action a_i (also called a **strategy**) from his own action set A_i. The vector $(a_1, \ldots, a_n)$ of individual actions is called a **joint action** or an **action profile**, and is denoted by a or (a_i). We will use the notation a_{-i} to refer to the actions of all agents except i, and (a_{-i}, a_i) to refer to a joint action where agent i takes a particular action a_i.
- The game is 'played' on a *fixed* world state s (thus we will not deal with dubious state transitions). The state can be defined as consisting of the n agents, their action sets A_i, and their payoffs (see next).
- Each agent i has his own action value function $Q_i^*(s, a)$ that measures the goodness of the *joint* action a *for the agent* i. Note that each agent may give different preferences to different joint actions. Since s is fixed, we drop the symbol s and instead use $u_i(a) \equiv Q_i^*(s, a)$ which is called the **payoff function** of agent i. We assume that the payoff functions are predefined and fixed. (We will deal with the case of learning the payoff functions in chapter 8.)
- The state is fully observable to all agents. That is, all agents know (i) each other, (ii) the action sets of each other, and (iii) the payoffs of each other. More strictly, the primitives (i)-(iii) of the game are **common knowledge** among agents. That is, all agents know (i)-(iii), they all know that they all know (i)-(iii), and so on to any depth. (We will discuss common knowledge in detail in chapter 5).
- Each agent chooses a single action; it is a single-shot game. Moreover, all agents choose their actions simultaneously and independently; no agent is informed of the decision of any other agent prior to making his own decision.

In summary, in a strategic game, each agent chooses a single action and then he receives a payoff that depends on the selected joint action. This joint action is called the **outcome** of the game. The important point to note is that, although the payoff functions of the agents are common knowledge,

[1] In this chapter we will use 'he' or 'she' to refer to an agent, following the convention in the literature (Osborne and Rubinstein, 1994, p. xiii).

	Not confess	*Confess*
Not confess	$3, 3$	$0, 4$
Confess	$4, 0$	$1, 1$

Figure 3.1: The prisoner's dilemma.

an agent does not know in advance the action choices of the other agents. The best he can do is to try to *predict* the actions of the other agents. A **solution** to a game is a prediction of the outcome of the game using the assumption that all agents are rational and strategic.

In the special case of two agents, a strategic game can be graphically represented by a **payoff matrix**, where the rows correspond to the actions of agent 1, the columns to the actions of agent 2, and each entry of the matrix contains the payoffs of the two agents for the corresponding joint action. In Fig. 3.1 we show the payoff matrix of a classical game, the **prisoner's dilemma**, whose story goes as follows:

> Two suspects in a crime are independently interrogated. If they both confess, each will spend three years in prison. If only one confesses, he will run free while the other will spend four years in prison. If neither confesses, each will spend one year in prison.

In this example each agent has two available actions, *Not confess* or *Confess*. Translating the above story into appropriate payoffs for the agents, we get in each entry of the matrix the pairs of numbers that are shown in Fig. 3.1 (note that a payoff is by definition a 'reward', whereas spending three years in prison is a 'penalty'). For example, the entry $4, 0$ indicates that if the first agent confesses and the second agent does not, then the first agent will get payoff 4 and the second agent will get payoff 0.

In Fig. 3.2 we see two more examples of strategic games. The game in Fig. 3.2(a) is known as 'matching pennies'; each of two agents chooses either *Head* or *Tail*. If the choices differ, agent 1 pays agent 2 a cent; if they are the same, agent 2 pays agent 1 a cent. Such a game is called **strictly competitive** or **zero-sum** because $u_1(a) + u_2(a) = 0$ for all a. The game in Fig. 3.2(b) is played between two car drivers at a crossroad; each agent wants to cross first (and he will get payoff 1), but if they both cross they will crash (and get payoff -1). Such a game is called a **coordination game** (we will extensively study coordination games in chapter 4).

What does game theory predict that a rational agent will do in the above examples? In the next sections we will describe two fundamental solution concepts for strategic games.

	Head	*Tail*
Head	$1, -1$	$-1, \ 1$
Tail	$-1, \ 1$	$1, -1$

(a)

	Cross	*Stop*
Cross	$-1, -1$	$1, \ 0$
Stop	$0, \ 1$	$0, \ 0$

(b)

Figure 3.2: A strictly competitive game (a), and a coordination game (b).

3.3 Iterated elimination of strictly dominated actions

The first solution concept is based on the assumption that a rational agent will never choose a suboptimal action. With suboptimal we mean an action that, no matter what the other agents do, will always result in lower payoff for the agent than some other action. We formalize this as follows:

Definition 3.3.1. We will say that an action a_i of agent i is **strictly dominated** by another action a_i' of agent i if

$$u_i(a_{-i}, a_i') > u_i(a_{-i}, a_i) \tag{3.1}$$

for all actions a_{-i} of the other agents.

In the above definition, $u_i(a_{-i}, a_i)$ is the payoff the agent i receives if he takes action a_i while the other agents take a_{-i}. In the prisoner's dilemma, for example, *Not confess* is a strictly dominated action for agent 1; no matter what agent 2 does, the action *Confess* always gives agent 1 higher payoff than the action *Not confess* (4 compared to 3 if agent 2 does not confess, and 1 compared to 0 if agent 2 confesses). Similarly, *Not confess* is a strictly dominated action for agent 2.

Iterated elimination of strictly dominated actions (IESDA) is a solution technique that iteratively eliminates strictly dominated actions from all agents, until no more actions are strictly dominated. It is solely based on the following two assumptions:

- A rational agent would never take a strictly dominated action.
- It is common knowledge that all agents are rational.

As an example, we will apply IESDA to the prisoner's dilemma. As we explained above, the action *Not confess* is strictly dominated by the action *Confess* for both agents. Let us start from agent 1 by eliminating the action *Not confess* from his action set. Then the game reduces to a single-row payoff matrix where the action of agent 1 is fixed (*Confess*) and agent 2 can choose between *Not confess* and *Confess*. Since the latter gives higher

	L	M	R
U	$1, 0$	$1, 2$	$0, 1$
D	$0, 3$	$0, 1$	$2, 0$

(a)

	L	M	R
U	$1, 0$	$1, 2$	$0, 1$
D	$0, 3$	$0, 1$	$2, 2$

(b)

Figure 3.3: Examples where IESDA predicts a single outcome (a), or predicts that any outcome is possible (b).

payoff to agent 2 (4 as opposed to 3 if she does not confess), agent 2 will prefer *Confess* to *Not confess*. Thus IESDA predicts that the outcome of the prisoner's dilemma will be (*Confess*, *Confess*).

As another example consider the game of Fig. 3.3(a) where agent 1 has two actions U and D and agent 2 has three actions L, M, and R. It is easy to verify that in this game IESDA will predict the outcome (U, M) by first eliminating R, then D, and finally L. However, IESDA may sometimes produce very inaccurate predictions for a game, as in the two games of Fig. 3.2 and also in the game of Fig. 3.3(b) where no actions can be eliminated. In these games IESDA essentially predicts that *any* outcome is possible.

A characteristic of IESDA is that the agents do not need to maintain *beliefs* about the other agents' strategies in order to compute their optimal actions. The only thing that is required is the common knowledge assumption that each agent is rational. Moreover, it can be shown that the algorithm is insensitive to the speed and the elimination order; it will always give the same results no matter how many actions are eliminated in each step and in which order. However, as we saw in the examples above, IESDA can sometimes fail to make accurate predictions for the outcome of a game.

3.4 Nash equilibrium

A **Nash equilibrium** (NE) is a stronger solution concept than IESDA, in the sense that it produces more accurate predictions in a wider class of games. It can be formally defined as follows:

Definition 3.4.1. A Nash equilibrium is a joint action a^* with the property that for every agent i holds

$$u_i(a^*_{-i}, a^*_i) \geq u_i(a^*_{-i}, a_i) \tag{3.2}$$

for all actions $a_i \in A_i$.

In other words, a NE is a joint action from where no agent can unilaterally improve his payoff, and therefore no agent has any incentive to deviate. Note that, contrary to IESDA that describes a solution of a game by means

of an algorithm, a NE describes a solution in terms of the *conditions* that hold at that solution.

There is an alternative definition of a NE that makes use of the so-called **best-response function**. This is defined as

$$B_i(a_{-i}) = \{a_i \in A_i : u_i(a_{-i}, a_i) \geq u_i(a_{-i}, a_i') \text{ for all } a_i' \in A_i\} \tag{3.3}$$

where $B_i(a_{-i})$ can be a set containing many actions. In the prisoner's dilemma, for example, when agent 2 takes the action *Not confess*, the best-response of agent 1 is the action *Confess* (because $4 > 3$). Similarly, we can compute the best-response function of each agent:

$$\begin{aligned} B_1(\textit{Not confess}) &= \textit{Confess}, \\ B_1(\textit{Confess}) &= \textit{Confess}, \\ B_2(\textit{Not confess}) &= \textit{Confess}, \\ B_2(\textit{Confess}) &= \textit{Confess}. \end{aligned}$$

In this case, the best-response functions are singleton-valued. Using the definition of a best-response function, we can now formulate the following:

Definition 3.4.2. A Nash equilibrium is a joint action a^* with the property that for every agent i holds

$$a_i^* \in B_i(a_{-i}^*). \tag{3.4}$$

That is, at a NE, each agent's action is an optimal response to the other agents' actions. In the prisoner's dilemma, for instance, given that $B_1(\textit{Confess}) = \textit{Confess}$, and $B_2(\textit{Confess}) = \textit{Confess}$, we conclude that (*Confess*, *Confess*) is a NE. Moreover, we can easily show the following:

Proposition 3.4.1. *The two definitions 3.4.1 and 3.4.2 of a NE are equivalent.*

Proof. Suppose that (3.4) holds. Then, using (3.3) we see that for each agent i, the action a_i^* must satisfy $u_i(a_{-i}^*, a_i^*) \geq u_i(a_{-i}^*, a_i')$ for all $a_i' \in A_i$. But this is precisely the definition of a NE according to (3.2). Similarly for the converse. □

The definitions 3.4.1 and 3.4.2 suggest a brute-force method for finding the Nash equilibria of a game: enumerate all possible joint actions and then verify which ones satisfy (3.2) or (3.4). Note that the cost of such an algorithm is exponential in the number of agents.

It turns out that a strategic game can have zero, one, or more than one Nash equilibria. For example, (*Confess*, *Confess*) is the only NE in the prisoner's dilemma. We also find that the zero-sum game in Fig. 3.2(a) does not have a NE, while the coordination game in Fig. 3.2(b) has two Nash

equilibria (*Cross*, *Stop*) and (*Stop*, *Cross*). Similarly, (U, M) is the only NE in both games of Fig. 3.3.

We argued above that a NE is a stronger solution concept than IESDA in the sense that it produces more accurate predictions of a game. For instance, the game of Fig. 3.3(b) has only one NE, but IESDA predicts that any outcome is possible. In general, we can show the following two propositions (the proof of the second is left as an exercise):

Proposition 3.4.2. *A NE always survives IESDA.*

Proof. Let a^* be a NE, and let us assume that a^* does not survive IESDA. This means that for some agent i the component a_i^* of the action profile a^* is strictly dominated by another action a_i of agent i. But then (3.1) implies that $u_i(a_{-i}^*, a_i) > u_i(a_{-i}^*, a_i^*)$ which contradicts the definition 3.4.1 of a NE. □

Proposition 3.4.3. *If IESDA eliminates all but a single joint action a, then a is the unique NE of the game.*

Note also that in the prisoner's dilemma, the joint action (*Not confess*, *Not confess*) gives both agents payoff 3, and thus it should have been the preferable choice. However, from this joint action each agent has an incentive to deviate, to be a 'free rider'. Only if the agents had made an agreement in advance, and only if trust between them was common knowledge, would they have opted for this non-equilibrium joint action which is optimal in the following sense:

Definition 3.4.3. A joint action a is **Pareto optimal** if there is no other joint action a' for which $u_i(a') > u_i(a)$ for all agents i.

In the above discussion we implicitly assumed that when the game is actually played, each agent i will choose his action deterministically from his action set A_i. This is however not always true. In many cases there are good reasons for the agent to introduce randomness in his behavior. In general, we can assume that an agent i chooses actions a_i with some probability distribution $p_i(a_i)$ which can be different for each agent. This gives us the following definition:

Definition 3.4.4. A **mixed strategy** for an agent i is a probability distribution $p_i(a_i)$ over the actions $a_i \in A_i$.

In his famous theorem, Nash (1950) showed that a strategic game with a finite number of agents and a finite number of actions always has an equilibrium in mixed strategies. Osborne and Rubinstein (1994, sec. 3.2) give several interpretations of such a mixed strategy Nash equilibrium.

3.5 Notes and further reading

The book of von Neumann and Morgenstern (1944) and the half-page long article of Nash (1950) are classics in game theory. Our discussion was mainly based on the book of Osborne and Rubinstein (1994) which is the standard textbook on game theory (and is rather technical). The book of Gibbons (1992) and the (forthcoming) book of Osborne (2003) offer a readable introduction to the field, with several applications. Russell and Norvig (2003, ch. 17) also include an introductory section on game theory.

Chapter 4

Coordination

In this chapter we take a closer look at the problem of **coordination**, that is, how the individual decisions of the agents can result in good joint decisions for the group. We analyze the problem using the framework of strategic games that we studied in chapter 3, and we describe several practical techniques like social conventions, roles, and coordination graphs.

4.1 Distributed decision making

As we already discussed in chapter 1, a distinguishing feature of a multiagent system is the fact that the decision making of the agents can be distributed. This means that there is no central controlling agent that decides what each agent must do at each time step, but each agent is to a certain extent 'responsible' for its own decisions. The main advantages of such a decentralized approach over a centralized one are *efficiency*, due to the asynchronous computation, and *robustness*, in the sense that the functionality of the whole system does not rely on a single agent.

In order for the agents to be able to take their actions in a distributed fashion, appropriate coordination mechanisms must be additionally developed. Coordination can be regarded as the process by which the individual decisions of the agents result in good joint decisions for the group. A typical situation where coordination is needed is among cooperative agents that form a team, and through this team they make joint plans and pursue common goals. In this case, coordination ensures that the agents do not obstruct each other when taking actions, and moreover that these actions serve the common goal. Robot soccer is such an example, where a team of robots attempt to score goals against an opponent team. Here, coordination ensures that, for instance, two teammate robots will never try to kick the ball at the same time. In robot soccer there is no centralized controlling agent that can instruct the robots in real-time whether or not to kick the ball; the robots must decide for themselves.

	Thriller	*Comedy*
Thriller	$1, 1$	$0, 0$
Comedy	$0, 0$	$1, 1$

Figure 4.1: A coordination game.

4.2 Coordination games

A way to describe a coordination problem is to model it as a strategic game and solve it according to some solution concept, for example, Nash equilibrium. We have already seen an example in Fig. 3.2(b) of chapter 3 where two cars meet at a crossroad and the drivers have to decide what action to take. If they both cross they will crash, and it is not of either's interest to stop. Only one driver is allowed to cross and the other driver must stop. Who is going to cross then? This is an example of a **coordination game** that involves two agents and two possible solutions: (*Cross*, *Stop*) and (*Stop*, *Cross*). As we saw in chapter 3, these two joint actions are Nash equilibria of the game. Moreover, they are both Pareto optimal.

In the case of fully cooperative agents, all n agents in the team share the same payoff function $u_1(a) = \ldots = u_n(a) \equiv u(a)$ in the corresponding coordination game. Figure 4.1 shows an example of a coordination game between two cooperative agents. The agents want to go to the movies together. Each agent has a choice between two types of movies, either a thriller or a comedy. Each agent has no prior information what movie the other agent will choose, and the agents choose independently and simultaneously. Choosing the same movie gives them payoff 1, otherwise they get payoff 0. In this game the agents have to coordinate their actions in order to maximize their payoff. As in the previous example, the two joint actions where the agents choose the same movie are two Pareto optimal Nash equilibria of the coordination game. Generalizing from the above examples we can formulate the following:

Definition 4.2.1. Coordination is the process in which a group of agents choose a single Pareto optimal Nash equilibrium in a strategic game.

An equilibrium that is Pareto optimal is also said to **payoff-dominate** all other equilibria (Harsanyi and Selten, 1988). Note that in chapter 3 we described a Nash equilibrium in terms of the conditions that hold at the equilibrium point, and disregarded the issue of how the agents can actually reach this point. The above definition shows that coordination is a more earthy concept: it asks how the agents can actually agree on a single equilibrium in a strategic game that involves more than one such equilibria. For simplicity, in the rest of the chapter, by 'equilibrium' we will always mean 'Pareto optimal Nash equilibrium', unless otherwise stated.

Reducing coordination to the problem of equilibrium selection in a strategic game allows for the application of existing techniques from game theory (Harsanyi and Selten, 1988). In the rest of this chapter we will focus on some simple coordination techniques that can be readily implemented in practical systems. We will throughout assume that the agents are cooperative (they share the same payoff function), and they have perfect information about the game primitives (see section 3.2).

4.3 Social conventions

As we saw above, in order to solve a coordination problem, a group of agents are faced with the problem how to choose their actions in order to select the same equilibrium in a game. Clearly, there can be no recipe to tell the agents *which* equilibrium to choose in every possible game they may play in the future. Nevertheless, we can devise recipes that will instruct the agents *how* to choose a single equilibrium in any game. Such a recipe will be able to guide the agents in their action selection procedure.

A **social convention** (or **social law**) is such a recipe that places constraints on the possible action choices of the agents. It can be regarded as a rule that dictates how the agents should choose their actions in a coordination game in order to reach an equilibrium. Moreover, given that the convention has been established and is common knowledge among agents, no agent can benefit from not abiding by it.

Boutilier (1996) has proposed a general convention that achieves coordination in a large class of systems and is very easy to implement. The convention assumes a unique *ordering scheme* of joint actions that is common knowledge among agents. In a particular game, each agent first computes all equilibria of the game, and then selects the first equilibrium according to this ordering scheme. For instance, a lexicographic ordering scheme can be used in which the agents are ordered first, and then the actions of each agent are ordered. In the coordination game of Fig. 4.1, for example, we can order the agents lexicographically by $1 \succ 2$ (meaning that agent 1 has 'priority' over agent 2), and the actions by *Thriller* $\succ$ *Comedy*. The first equilibrium in the resulting ordering of joint actions is (*Thriller*, *Thriller*) and this will be the unanimous choice of the agents.

Given that a single equilibrium has been selected, each agent can easily choose his individual action as the corresponding component of the selected equilibrium. The complete algorithm, which we will refer to as **coordination by social conventions**, is shown in Fig. 4.2. Note that the algorithm is executed identically by each agent in parallel. Clearly, since the ordering scheme is common knowledge among agents, all agents must necessarily agree on the same equilibrium. The time and space requirements of the algorithm are dominated by the computation and storage of all equilibria of

For each agent i in parallel
    Compute all equilibria of the game.
    Order these equilibria based on a unique ordering scheme.
    Select the first equilibrium $a^* = (a^*_{-i}, a^*_i)$ in the ordered list.
    Choose action a^*_i.
End

Figure 4.2: Coordination by social conventions.

the game.

When the agents can perceive more aspects of the world state than just the primitives of the game (actions and payoffs), one can think of more elaborate ordering schemes for coordination. Consider the traffic game of Fig. 3.2(b), for example, as it is 'played' in the real world. Besides the game primitives, the state now also contains the relative orientation of the cars in the physical environment. If we assume that the perception of the agents fully reveals the state (full observability), then a simple convention is that the driver coming from the right will always have priority over the other driver in the lexicographic ordering. If we also order the actions by $\mathit{Cross} \succ \mathit{Stop}$, then coordination by social conventions implies that the driver from the right will cross the road first. Similarly, if traffic lights are available, then the state also includes the color of the light, in which case the established convention is that the driver who sees the red light must stop.

4.4 Roles

Coordination by social conventions relies on the assumption that an agent can compute all equilibria in a game before choosing a single one. However, computing equilibria can be expensive when the action sets of the agents are large, therefore one would like to reduce the size of the action sets first. Such a reduction can have computational advantages in terms of speed, but more importantly, it can simplify the equilibrium selection problem. In some cases, in the resulting subgame there is only one equilibrium left which is trivial to find.

A natural way to reduce the action sets of the agents is by assigning **roles** to the agents. Formally, a role can be regarded as a masking operator on the action set of an agent, given a particular state. In practical terms, if an agent is assigned a role at a particular state, then some of the agent's actions are deactivated at this state. In soccer for example, an agent that is currently in the role of defender cannot attempt to *Score*.

A role can facilitate the solution of a coordination game by reducing

For each agent in parallel
    $I = \{\}$.
    For each role $j = 1, \ldots, n$
        For each agent $i = 1, \ldots, n$ with $i \notin I$
            Compute the potential r_{ij} of agent i for role j.
        End
        Assign role j to agent $i^* = \arg\max_i \{r_{ij}\}$.
        Add i^* to I.
    End
End

Figure 4.3: Role assignment.

it to a subgame where the equilibria are easier to find. For example, in Figure 4.1, if agent 2 is assigned a role that forbids him to select the action *Thriller* (e.g., he is under 16), then agent 1, assuming he knows the role of agent 2, can safely choose *Comedy* resulting in coordination. Note that there is only one equilibrium left in the subgame formed after removing the action *Thriller* from the action set of agent 2.

In general, suppose that there are n available roles (not necessarily distinct), that the state is fully observable to the agents, and that the following facts are common knowledge among agents:

- There is a fixed ordering $\{1, 2, \ldots, n\}$ of the roles. Role 1 must be assigned first, followed by role 2, etc.
- For each role j there is a function that assigns to each agent i a **potential** r_{ij} that reflects how appropriate agent i is for the role j given the current state.
- Each agent can be assigned only one role.

Then, **role assignment** can be carried out by the algorithm shown in Fig. 4.3. Each role is assigned to the agent that has the highest potential for that role. This agent is eliminated from the role assignment process, a new role is assigned to another agent, and so on, until all agents have been assigned a role.

The algorithm runs in time polynomial in the number of agents and roles, and is executed identically by each agent in parallel. Note that we have assumed full observability of the state, and that each agent can compute the potential of each other agent. After all roles have been assigned, the original coordination game is reduced to a subgame that can be further solved using coordination by social conventions. Recall from Fig. 4.2 that

the latter additionally requires an ordering scheme of the joint actions that is common knowledge.

Like in the traffic example of the previous section, roles typically apply on states that contain more aspects than only the primitives of the strategic game; for instance, they may contain the physical location of the agents in the world. Moreover, the role assignment algorithm applies even if the state is continuous; the algorithm only requires a function that computes potentials, and such a function can have a continuous state space as domain. To give an example, suppose that in robot soccer we want to assign a particular role j (e.g., attacker) to the robot that is closer to the ball than any other teammate. In this case, the potential r_{ij} of a robot i for role j can be given by a function of the form

$$r_{ij} = -||\mathbf{x}_i - \mathbf{x}_b||, \tag{4.1}$$

where $\mathbf{x}_i$ and $\mathbf{x}_b$ are the locations in the field of the robot i and the ball, respectively.

4.5 Coordination graphs

As mentioned above, roles can facilitate the solution of a coordination game by reducing the action sets of the agents prior to computing the equilibria. However, computing equilibria in a subgame can still be a difficult task when the number of involved agents is large; recall that the joint action space is exponentially large in the number of agents. As roles reduce the size of the action sets, we also need a method that reduces the number of agents involved in a coordination game.

Guestrin et al. (2002a) introduced the **coordination graph** as a framework for solving large-scale coordination problems. A coordination graph allows for the decomposition of a coordination game into several smaller subgames that are easier to solve. Unlike roles where a single subgame is formed by the reduced action sets of the agents, in this framework various subgames are formed, each typically involving a small number of agents.

In order for such a decomposition to apply, the main assumption is that the global payoff function $u(a)$ can be written as a linear combination of k local payoff functions f_j, each involving only few agents. For example, suppose that there are $n = 4$ agents, and $k = 3$ local payoff functions, each involving two agents:

$$u(a) = f_1(a_1, a_2) + f_2(a_1, a_3) + f_3(a_3, a_4). \tag{4.2}$$

Here, $f_2(a_1, a_3)$ for instance involves only agents 1 and 3, with their actions a_1 and a_3. Such a decomposition can be graphically represented by a graph (hence the name), where each node represents an agent and each edge corresponds to a local payoff function. For example, the decomposition (4.2) can be represented by the graph of Fig. 4.4.

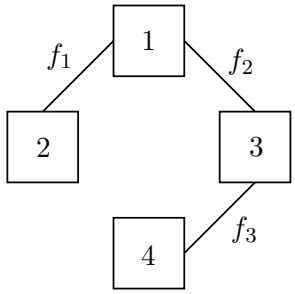


Figure 4.4: A coordination graph for a 4-agent problem.

Let us now see how this framework can be used for coordination. Recall that a solution to a coordination problem is a Pareto optimal Nash equilibrium in the corresponding strategic game. By definition, such an equilibrium is a joint action a^* that maximizes $u(a)$. The key idea in coordination graphs is that the linear decomposition of $u(a)$ allows for an *iterative* maximization procedure in which agents are eliminated one after the other.

We will illustrate this on the above example. We start by eliminating agent 1 in (4.2). We collect all local payoff functions that involve agent 1, these are f_1 and f_2. The maximum of $u(a)$ can then be written

$$\max_a u(a) = \max_{a_2,a_3,a_4} \Big\{ f_3(a_3, a_4) + \max_{a_1} \big[f_1(a_1, a_2) + f_2(a_1, a_3) \big] \Big\}. \tag{4.3}$$

Next we perform the inner maximization over the actions of agent 1. For each combination of actions of agents 2 and 3, agent 1 must choose an action that maximizes $f_1 + f_2$. This essentially involves computing the best-response function $B_1(a_2, a_3)$ of agent 1 (see section 3.4) in the subgame formed by agents 1, 2, and 3, and the sum of payoffs $f_1 + f_2$. The function $B_1(a_2, a_3)$ can be thought of as a *conditional* strategy for agent 1, given the actions of agents 2 and 3.

The above maximization and the computation of the best-response function of agent 1 define a new payoff function $f_4(a_2, a_3) = \max_{a_1}[f_1(a_1, a_2) + f_2(a_1, a_3)]$ that is independent of a_1. Agent 1 has been eliminated. The maximum (4.3) becomes

$$\max_a u(a) = \max_{a_2,a_3,a_4} \big[f_3(a_3, a_4) + f_4(a_2, a_3) \big]. \tag{4.4}$$

We can now eliminate agent 2 as we did with agent 1. In (4.4), only f_4 involves a_2, and maximization of f_4 over a_2 gives the best-response function $B_2(a_3)$ of agent 2 which is a function of a_3 only. This in turn defines a new payoff function $f_5(a_3)$, and agent 2 is eliminated. Now we can write

$$\max_a u(a) = \max_{a_3,a_4} \big[f_3(a_3, a_4) + f_5(a_3) \big]. \tag{4.5}$$

For each agent in parallel
  $F = \{f_1, \ldots, f_k\}$.
  For each agent $i = 1, 2, \ldots, n$
    Find all $f_j(a_{-i}, a_i) \in F$ that involve a_i.
    Compute $B_i(a_{-i}) = \arg\max_{a_i} \sum_j f_j(a_{-i}, a_i)$.
    Compute $f_{k+i}(a_{-i}) = \max_{a_i} \sum_j f_j(a_{-i}, a_i)$.
    Remove all $f_j(a_{-i}, a_i)$ from F and add $f_{k+i}(a_{-i})$ in F.
  End
  For each agent $i = n, n-1, \ldots, 1$
    Choose $a_i^* \in B_i(a_{-i}^*)$ based on a fixed ordering of actions.
  End
End

Figure 4.5: Coordination by variable elimination.

Agent 3 is eliminated next, resulting in $B_3(a_4)$ and a new payoff function $f_6(a_4)$. Finally, $\max_a u(a) = \max_{a_4} f_6(a_4)$, and since all other agents have been eliminated, agent 4 can simply choose an action a_4^* that maximizes f_6.

The above procedure computes an optimal action only for the last eliminated agent (assuming that the graph is connected). For the other agents it computes only conditional strategies. A second pass in the reverse elimination order is needed so that all agents compute their optimal (unconditional) actions from their best-response functions. Thus, in the above example, plugging a_4^* into $B_3(a_4)$ gives the optimal action a_3^* of agent 3. Similarly, we get a_2^* from $B_2(a_3^*)$ and a_1^* from $B_1(a_2^*, a_3^*)$, and thus we have computed the joint optimal action $a^* = (a_1^*, a_2^*, a_3^*, a_4^*)$. Note that one agent may have more than one best-response actions, in which case the first action can be chosen according to an a priori ordering of the actions of each agent that must be common knowledge.

The complete algorithm, which we will refer to as **coordination by variable elimination**, is shown in Fig. 4.5. Note that the notation $-i$ that appears in $f_j(a_{-i}, a_i)$ refers to all agents other than agent i that are involved in f_j, and it does not necessarily include all $n-1$ agents. Similarly, in the best-response functions $B_i(a_{-i})$ the action set a_{-i} may involve less than $n-1$ agents. The algorithm runs identically for each agent in parallel. For that we require that all local payoff functions are common knowledge among agents, and that there is an a priori ordering of the action sets of the agents that is also common knowledge. The latter assumption is needed so that each agent will finally compute the same joint action.

The main advantage of this algorithm compared to coordination by social conventions in Fig. 4.2 is that here we need to compute best-response func-

tions in subgames involving only few agents, while computing all equilibria in Fig. 4.2 requires computing best-response functions in the complete game involving all n agents. When n is large, the computational gains of variable elimination over coordination by social conventions can be significant.

For simplicity, in the above algorithm we have fixed the elimination order of the agents as $1, 2, \ldots, n$. However, this is not necessary. Each agent running the algorithm can choose a different elimination order, and the resulting joint action a^* will always be the same. The total runtime of the algorithm, however, will not be the same. Different elimination orders produce different intermediate payoff functions, and thus subgames of different size. It turns out that computing the optimal elimination order (that minimizes the execution cost of the algorithm) is NP-complete.

In all algorithms in this chapter the agents can coordinate their actions without the need to communicate with each other. As we saw, in all cases each agent runs the same algorithm identically and in parallel, and coordination is guaranteed as long as certain facts are common knowledge among agents. In chapter 6 we will relax some of the assumptions of common knowledge, and show how the above algorithms can be modified when the agents can explicitly communicate with each other.

4.6 Notes and further reading

The problem of multiagent coordination has been traditionally studied in distributed AI, where typically some form of communication is allowed among agents (Jennings, 1996). (See also chapter 6.) The framework of 'joint intentions' of Cohen and Levesque (1991) provides a formal characterization of multiagent coordination through a model of joint beliefs and intentions of the agents. Social conventions were introduced by Shoham and Tennenholtz (1992), as constraints on the set of allowed actions of a single agent at a given state (similar to roles as in section 4.4). Boutilier (1996) extended the definition to include also constraints on the joint action choices of a group of agents, and proposed the idea of coordination by lexicographic ordering. A version of role assignment (that relies on communication) similar to Fig. 4.3 has been used by Castelpietra et al. (2000). Coordination graphs are due to Guestrin et al. (2002a). An application of roles and coordination graphs in robot soccer can be found in (Kok et al., 2003).

Chapter 5

Common knowledge

In the previous chapters we generally assumed that the world state is fully observable to the agents. Here we relax this assumption and examine the case where parts of the state are hidden to the agents. In such a partially observable world an agent must always reason about his knowledge, and the knowledge of the others, prior to making decisions. We formalize the notions of knowledge and common knowledge in such domains, and demonstrate their implications by means of examples.

5.1 Thinking interactively

In order to act rationally, an agent must always reflect on what he knows about the current world state. As we saw in chapter 2, if the state is fully observable, an agent can perform pretty well without extensive deliberation. If the state is partially observable, however, the agent must first consider carefully what he knows and what he does not know before choosing an action. Intuitively, the more an agent knows about the true state, the better the decisions he can make.

In a multiagent system, a rational agent must also be able to think *interactively*, that is, to take into account the knowledge of the other agents in his decision making. In addition, he needs to consider what the other agents know about him, and also what they know about his knowledge. In the previous chapters we have often used the term **common knowledge** to refer to something that every agent knows, that every agent knows that every other agent knows, and so on. For example, the social convention that a driver must stop in front of a red traffic light is supposed to be common knowledge among all drivers.

In this chapter we will define common knowledge more formally, and illustrate some of its strengths and implications through examples. One of its surprising consequences, for instance, is that it cannot be common knowledge that two rational agents would ever want to bet with each other!

5.2 The puzzle of the hats

We start with a classical puzzle that illustrates some of the implications of common knowledge. The story goes as follows:

> Three agents (say, girls) are sitting around a table, each wearing a hat. A hat can be either red or white, but suppose that all agents are wearing red hats. Each agent can see the hat of the other two agents, but she does not know the color of her own hat. A person who observes all three agents asks them in turn whether they know the color of their hats. Each agent replies negatively. Then the person announces "At least one of you is wearing a red hat", and then asks them again in turn. Agent 1 says *No*. Agent 2 also says *No*. But when he asks agent 3, she says *Yes*.

How is it possible that agent 3 can finally figure out the color of her hat? Before the announcement that at least one of them is wearing a red hat, no agent is able to tell her hat color. What changes then after the announcement? Seemingly the announcement does not reveal anything new; each agent already knows that there is at least one red hat because she can see the red hats of the other two agents.

Given that everyone has heard that there is at least one red hat, agent 3 can tell her hat color by reasoning as follows: "Agent's 1 *No* implies that either me or agent 2 is wearing a red hat. Agent 2 knows this, so if my hat had been white, agent 2 would have said *Yes*. But agent 2 said *No*, so my hat must be red."

Although each agent already knows (by perception) the fact that at least one agent is wearing a red hat, the key point is that the public announcement of the person *makes this fact common knowledge* among the agents. Implicitly we have also assumed that it is common knowledge that each agent can see and hear well, and that she can reason rationally. The puzzle is instructive because it shows the implications of interactive reasoning and the strength of the common knowledge assumption.

5.3 Partial observability and information

Now we will try to formalize some of the concepts that appear in the puzzle of the hats. The starting point is that world state is partially observable to the agents. Recall that in a partially observable world the perception of an agent provides only partial information about the true state, in the form of a conditional probability distribution over states (see section 2.3). In the puzzle of the hats this distribution is uniform over subsets of the state space, as we show next.

		World states							
		a	b	c	d	e	f	g	h
Agents	1	R	R	R	R	W	W	W	W
	2	R	R	W	W	R	R	W	W
	3	R	W	R	W	R	W	R	W

Figure 5.1: The eight world states in the puzzle of the hats.

In general, let S be the set of all states and $s \in S$ be the current (true) state of the world. We assume that the perception of an agent i provides information about the state s through an **information function**

$$P_i : s \mapsto P_i(s), \tag{5.1}$$

where $P_i(s)$ is a nonempty subset of S called the **information set** of agent i in state s. The interpretation of the information set is that when the true state is s, agent i thinks that *any* state in $P_i(s)$ could be the true state. The set $P_i(s)$ will always contain s, but essentially this is the only thing that agent i knows about the true state. In the case of multiple agents, each agent can have a different information function.

In the puzzle of the hats, a state is a three-component vector containing the colors of the hats. Let R and W denote red and white. There are in total eight states $S = \{a, b, c, d, e, f, g, h\}$, as shown in Fig. 5.1. By assumption, the true state is $s = a$. From the setup of the puzzle we know that the state is partially observable to each agent; only two of the three hat colors are directly perceivable by each agent. In other words, in any state s the information set of each agent contains two equiprobable states, those in which the only difference is in her own hat color. For instance, in state $s = a$ the information set of agent 2 is $P_2(s) = \{a, c\}$, a two-state subset of S.

As we mentioned above, the information set $P_i(s)$ of an agent i contains those states in S that agent i considers possible, if the true state is s. In general, we assume that the information function of an agent divides the state space into a collection of mutually disjoint subsets, called **cells**, that together form a **partition** $\mathcal{P}_i$ of S. The information set $P_i(s)$ for agent i in true state s is exactly that cell of $\mathcal{P}_i$ that contains s, while the union of all cells in $\mathcal{P}_i$ is S.

Based on the information functions, we can compute the partitions of the agents in the puzzle of the hats:

$$\mathcal{P}_1^t = \{\{a, e\}, \{b, f\}, \{c, g\}, \{d, h\}\}, \tag{5.2}$$

$$\mathcal{P}_2^t = \{\{a, c\}, \{b, d\}, \{e, g\}, \{f, h\}\}, \tag{5.3}$$

$$\mathcal{P}_3^t = \{\{a, b\}, \{c, d\}, \{e, f\}, \{g, h\}\}, \tag{5.4}$$

where t refers to the time step before any announcement took place. Clearly, in the true state $s = a = RRR$ no agent knows her hat color, since the corresponding cell of each partition contains two equiprobable states. Thus, agent 1 considers a and e possible, agent 2 considers a and c possible, and agent 3 considers a and b possible. (Note again that we know that the true state is a but the agents in our puzzle do not.)

Now we make the additional assumption that *all partitions are common knowledge* among the agents. In the case of homogeneous agents, for instance, this is not an unrealistic assumption; typically each agent will be aware of the perception capabilities of each other. In the puzzle of the hats, for example, it is reasonable to assume that all agents can see and hear well and that they are all rational. Then, simply the positioning of the agents around the table makes the above partitions common knowledge.

If the partitions are common knowledge, then in state a agent 1 thinks that agent 2 may think that agent 3 might think that $h = WWW$ is possible! Why is that? Note from (5.2) that in state a agent 1 thinks that either a or e could be the true state. But if e is the true state, then from (5.3) we see that agent 2 may consider g to be the true state. But then we see from (5.4) that agent 3 may consider h to be the true state. Note how the above analytical framework allows for a fairly straightforward formulation of otherwise complicated statements.

Now the announcement of the person reveals that the true state is *not* h. This automatically changes the partitions of the agents:

$$\begin{aligned} \mathcal{P}_1^{t+1} &= \{\{a,e\},\{b,f\},\{c,g\},\{d\},\{h\}\}, \\ \mathcal{P}_2^{t+1} &= \{\{a,c\},\{b,d\},\{e,g\},\{f\},\{h\}\}, \\ \mathcal{P}_3^{t+1} &= \{\{a,b\},\{c,d\},\{e,f\},\{g\},\{h\}\}. \end{aligned} \tag{5.5}$$

Note that h has been disambiguated from d, f, and g, in the three partitions. The person then asks each agent in turn whether she knows the color of her hat. Agent 1 says *No*. In which case would agent 1 have said *Yes*? As we see from the above partitions, only in state d would agent 1 have known her hat color. But the true state is a, and in this state agent 1 still considers e possible. (Compare this analysis with the logical reasoning of section 5.2.)

The reply of agent 1 eliminates state d from the set of candidate states. This results in a refinement of the partitions of agents 2 and 3:

$$\begin{aligned} \mathcal{P}_1^{t+2} &= \{\{a,e\},\{b,f\},\{c,g\},\{d\},\{h\}\}, \\ \mathcal{P}_2^{t+2} &= \{\{a,c\},\{b\},\{d\},\{e,g\},\{f\},\{h\}\}, \\ \mathcal{P}_3^{t+2} &= \{\{a,b\},\{c\},\{d\},\{e,f\},\{g\},\{h\}\}. \end{aligned} \tag{5.6}$$

Next agent 2 is asked. From her partition P_2^{t+2} we see that she would have known her hat color only in state b or f (d and h are already ruled out by the previous announcements). However, in the true state a agent 2 still

considers c possible, therefore she replies negatively. Her reply excludes b and f from the set of candidate states, resulting in a further refinement of the partitions of agent 1 and 3:

$$\begin{aligned}\mathcal{P}_1^{t+3} &= \{\{a,e\},\{b\},\{f\},\{c,g\},\{d\},\{h\}\},\\ \mathcal{P}_2^{t+3} &= \{\{a,c\},\{b\},\{d\},\{e,g\},\{f\},\{h\}\},\\ \mathcal{P}_3^{t+3} &= \{\{a\},\{b\},\{c\},\{d\},\{e\},\{f\},\{g\},\{h\}\}.\end{aligned} \tag{5.7}$$

The partition of agent 3 now contains only singleton cells, thus agent 3 can now tell her hat color. Note that agents 1 and 2 still cannot tell their hat colors. In fact, they will be unable to tell their har colors no matter how many more announcements will take place; the partitions (5.7) cannot be further refined. Interestingly, the above analysis would have been exactly the same if the true state had been any one in the set $\{a,c,e,g\}$. (Try to verify this with logical reasoning.)

5.4 A model of knowledge

Any subset E of S is called an **event**. If for an agent i holds $P_i(s) \subseteq E$ in true state s, then we say that agent i **knows**[1] E. Generalizing, the **knowledge function** of an agent i is defined as

$$K_i(E) = \{s \in S : P_i(s) \subseteq E\}. \tag{5.8}$$

That is, for any event E, the set $K_i(E)$ contains all states in which agent i knows E. It is not difficult to see that $K_i(E)$ can be written as the union of all cells of $\mathcal{P}_i$ that are fully contained in E. In the puzzle of the hats, for example, in the final partitions (5.7) holds $K_1(\{a,e,c\}) = \{a,e\}$, while for the event $E = \{a,c,e,g\}$ holds $K_i(E) = E$ for all $i = 1,2,3$.

An event $E \subseteq S$ is called **self-evident** to agent i if E can be written as a union of cells of $\mathcal{P}_i$. For example, in (5.7) the event $E = \{a,c,e,g\}$ is self-evident to all three agents. As another example, suppose that the state space consists of the integer numbers from 1 to 8, the true state is $s = 1$, and two agents have the following partitions:

$$\begin{aligned}\mathcal{P}_1 &= \{\{1,2\},\{3,4,5\},\{6\},\{7,8\}\},\\ \mathcal{P}_2 &= \{\{1,2,3\},\{4\},\{5\},\{6,7,8\}\}.\end{aligned} \tag{5.9}$$

[1] This definition of knowledge is slightly different from the one used in **epistemic logic**. There an agent is said to know a fact ϕ if ϕ is true in all states the agent considers possible. In the event-based framework, an agent knows an event E if all the states the agent considers possible are contained in E. Intuitively, if a fact ϕ is true in all $s \in E$, and the agent knows E, then the agent also knows ϕ (since ϕ is true in all states the agent considers possible). Fagin et al. (1995, sec. 2.5) show that the two approaches, logic-based and event-based, are in fact equivalent.

In $s = 1$ agent 1 thinks that $\{1, 2\}$ are possible. Agent 1 also thinks that agent 2 may think that $\{1, 2, 3\}$ are possible. Furthermore, agent 1 thinks that agent 2 may think that agent 1 might think that $\{1, 2\}$ or $\{3, 4, 5\}$ are possible. But nobody needs to think beyond 5. In this example, the event $\{1, 2, 3, 4\}$ is self-evident to agent 2, while the event $\{1, 2, 3, 4, 5\}$ is self-evident to both agents.

We can now formalize the notion of common knowledge. For simplicity, the first definition is formulated for only two agents.

Definition 5.4.1. An event $E \subseteq S$ is common knowledge between agents 1 and 2 in true state $s \in S$, if s is a member of *every* set in the infinite sequence $K_1(E)$, $K_2(E)$, $K_1(K_2(E))$, $K_2(K_1(E))$, ...

Definition 5.4.2. An event $E \subseteq S$ is common knowledge among a group of agents in true state $s \in S$, if s is a member of *any* set $F \subseteq E$ that is self-evident to all agents.

It turns out that the two definitions are equivalent (Osborne and Rubinstein, 1994, prop. 74.2). However, the second definition is much easier to apply; it only requires computing self-evident sets which are unions of partition cells and thus easy to find. For instance, in the above example the event $E = \{1, 2, 3, 4, 5\}$ is common knowledge between the two agents because E is self-evident to both of them and the true state $s = 1$ belongs to E. Similarly, in the puzzle of the hats, in the final partitions (5.7), and with true state $s = a$, the event $E = \{a, c, e, g\}$ is common knowledge among all three agents.

5.5 Knowledge and actions

So far we have discussed how the perceptions of the agents are related to the world states through the information functions, and what it means to say that an event is common knowledge among a group of agents. We now extend the framework to incorporate also the actions of the agents.

In section 2.2 we defined a memoryless policy π of an agent as a mapping from perceptions to actions (2.2). Equivalently, in the current framework we can assume that the policy of an agent i with information function P_i is a mapping from information sets to actions:

$$\pi_i : P_i(s) \mapsto a_i \tag{5.10}$$

where s is the current state. For example, in the puzzle of the hats, a reply *Yes* or *No* of an agent to the question "Do you know your hat color?" can be regarded as an action taken by the agent given her current information. In the final partitions (5.7), agent 1 will reply *No* given her information set $\{a, e\}$ and agent 3 will reply *Yes* given her information set $\{a\}$.

Let E be the set of states in which agent i takes a particular action a_i, that is, $E = \{s \in S : \pi_i(P_i(s)) = a_i\}$. If another agent j knows the policy and the information function of agent i, then agent j will know that agent i takes action a_i in state s if and only if agent j knows E. Similarly, assuming that the policy and the information function of agent i is common knowledge, then it will be common knowledge in s that agent i takes action a_i if and only if E is common knowledge.

Let us also assume that for two information sets E and F of an agent i the following 'union-consistency' property holds:

$$\pi_i(E) = \pi_i(F) \implies \pi_i(E) = \pi_i(E \cup F). \tag{5.11}$$

In other words, if an agent is taking the same action in two different information sets, then he must be taking the same action in their union as well. In the puzzle of the hats, for instance, if an agent answers *No* if she knows that the true state s is in E, and she also answers *No* if she knows that $s \in F$, then she will also answer *No* if she only knows that $s \in E \cup F$.

Given the above, it turns out that for a group of agents with potentially different information functions, the following result holds:

Proposition 5.5.1. *If the agents run the same policy π, and it is common knowledge in the current state what actions the agents are taking, then the agents must be taking the same actions.*

Proof. The fact that in the current state the action a_i of an agent i is common knowledge among all agents implies that there must be a set $E \subseteq S$ that is common knowledge among all agents, and in all states of which agent i would take action a_i, that is, $\pi(P_i(s)) = a_i$ for all $s \in E$. Since E is common knowledge among the agents, it must be self-evident to all agents, thus we can write $E = \bigcup_{s \in E} P_i(s)$ for all i. But then (5.11) implies that $\pi(E) = a_i$ for all i, therefore all a_i must be equal. □

As noted by Geanakoplos (1992), a surprising consequence of this proposition is that, for instance, rational agents would never want to bet with each other if their bets are common knowledge. Moreover, this is true even if the agents have different information about the current state. Betting is equivalent to deciding whether a random variable has positive or negative expectation. For expectations of random variables the union-consistency property (5.11) holds, therefore, assuming the agents are of the same 'mind', proposition 5.5.1 applies.

Finally, there is the question of how the agents can actually *reach* common knowledge of actions. As we saw in the puzzle of the hats, the public announcement of the action of each agent allows for a stepwise refinement of the partitions of the agents. At some point in time this refinement stops because there are only a finite number of states. At this point it will be

common knowledge in every state of the world what each agent is going to do in the future. Geanakoplos (1992) cites the following amusing story:

> Two detectives are trained in the same police academy, and share a well-defined policy that specifies whom to arrest in a particular case given the clues that have been discovered. A murder occurs, the detectives collect (different) clues for some time, and finally make up their own (different) opinions about the suspect. Then they meet and exchange their opinions, but not the clues that led them to these opinions. Hearing each other's opinion, each detective may change his mind and give another opinion. This may cause a further change in opinion. If they talk long enough, however, then they will come up with the same opinion at the end! Their final decision can be explained by a common set of clues, although their individual clues might be different.

The above story is based on the fact that common knowledge of actions is achieved after a finite number of public announcements of actions, and then proposition 5.5.1 guarantees that, as long as the agents run the same policy, they must necessarily take the same actions. On the other hand, if the actions of the agents are not publicly announced and there are delays or failures in the transmission of messages from one agent to the other, then, as shown by Fagin et al. (1995), common knowledge cannot be achieved.

5.6 Notes and further reading

The concept of common knowledge was introduced by Lewis (1969). We have mainly followed Osborne and Rubinstein (1994, ch. 5) and Geanakoplos (1992). The latter article is a readable overview on the topic, with many examples. The definition 5.4.2 of common knowledge and proposition 5.5.1 are due to Aumann (1976). Fagin et al. (1995) provide an epistemic-logic treatment of knowledge and common knowledge, and give several impossibility results in the case of unreliable communication between agents.

Chapter 6

Communication

In this chapter we address the issue of multiagent communication. We view communication as a means for the agents to revise their knowledge, and we briefly review communicative acts, a formal framework for agent interaction. Then we explain how an agent can compute the value of a communicative act using the framework of Bayesian games. Finally, we show how communication can be effectively used for multiagent coordination, by appropriately modifying the coordination algorithms of chapter 4.

6.1 Communicating agents

Multiagent interaction is often associated with some form of **communication**. We employ communication in most of our daily social activities, for cooperatively solving a problem, for negotiating with others, or simply for exchanging knowledge with others. As we saw in the puzzle of the hats in chapter 5, by communicating their answers to each other, the agents were able to form more accurate statements about the problem, and eventually solve it.

When we talk about computational agents, communication involves several levels of abstraction. On the lower, 'network' level, one would like to make sure that the messages that are communicated among the agents arrive safely and timely at their destination. For that, several well-studied formalisms and protocols exist in the distributed systems literature (Tanenbaum and van Steen, 2001). On an intermediate, 'language' level, one would like to have a basic set of language primitives and a standardized format for exchanging these primitives, so that agents that speak the same language can easily understand each other. Finally, on a high, 'application' level, one would like to effectively use communication for solving standard multiagent problems, like coordination or negotiation.

Next we will briefly address the issues of defining appropriate language primitives, and using communication for multiagent coordination.

6.2 Communicative acts

As we saw in the previous chapters, a world state is characterized by a number of agents, the available actions and the payoffs of the agents if the latter are involved in a strategic game, and other aspects of the external environment (e.g., the color of a traffic light in a traffic coordination problem). In some cases, as we saw in section 4.4, some of the actions of an agent may be deactivated in a particular state because this agent is assigned a particular role. Similarly, if the world state is partially observable to the agents, each agent may possess different levels of knowledge about the true state, as we saw in chapter 5; if the true state is s, an agent i may consider all states in his information set $P_i(s)$ as candidate true states.

A formal way to describe communication is by treating each communication primitive as an action that updates the knowledge of an agent about aspects of the state like those described above. The communication primitives that are exchanged among agents are typically referred to as **communicative acts** or **speech acts**. Some of the most common types of communicative acts are the following (as they could be applied, for example, by an agent in the traffic coordination problem of Fig. 3.2):

Informing about some aspects of the current state, "The traffic light is out of order," or about what an agent knows, "The car from the right has priority."

Querying aspects of the state that are hidden, "Does your traffic light work?" or "Do you consider crossing?" or "What is the convention in this country?"

Committing to a particular action, "I will cross."

Prohibiting an action, "You may not cross."

Directing an agent to do an action, "Cross!"

Each communicative act can affect the knowledge of the agents in a different way. For example, an informing act can reduce the uncertainty of an agent about the current state by eliminating candidate states from his information set. In the puzzle of the hats, for example, if at the beginning agent 1 says "The hat of agent 3 is red", then, given that agent 1 is truthful, the information set of agent 3 is transformed from $\{a, b\}$ to $\{a\}$, and the partitions of all agents change accordingly. Similarly, the answer *No* of an agent i to the query "Do you know your hat color?" is a communicative act that informs the other agents about the current knowledge of agent i regarding the true state, and results in a refinement of their partitions as we saw in section 5.3.

A committing communicative act, on the other hand, can be used for informing about the chosen course of action of an agent. We have already seen an example where an agent commits to choosing a specific course of action: in the backward pass of the variable elimination algorithm of Fig. 4.5, an agent that computes its optimal action according to its best-response function and given the optimal actions of the other agents so far, implicitly commits to taking this action. In section 6.4, this communicative act will be made more explicit by having each agent communicate its chosen action to the other agents.

Similar interpretations can be given to the other communicative acts. For instance, a prohibiting act can have the same effect as a role (see section 4.4), by enforcing the deactivation of some actions of an agent in a particular situation. A directing act may apear in an organizational structure of the agents in which an agent with more authority can give commands to the other agents.

Recently, several **agent communication languages** have been proposed in the agents community, aiming at standardizing the multiagent communication process (Labrou et al., 1999). The two most notable ones are KQML and FIPA ACL, each using a slightly different syntax and set of communicative acts. Unfortunately, many dialects of these languages have already appeared, and the languages seem not to conform with other standards (for example, in the Internet). As it is often the case, we might see in the future new language standards emerging directly from applications.

6.3 The value of communication

As we mentioned above, a communicative act can be viewed as an action that changes the knowledge state of the involved agents. Given a set of available communicative acts, an agent in a particular state is faced with the question *which* act to use and *whom* to communicate it to. Naturally, we would like to assign a value to a communicative act as an indicator of the goodness of this act, just like a Q-value measures the goodness of a regular action of an agent (see section 2.4). This would allow an agent to select the most appropriate communicative act per case. But where should these values come from?

A formal framework that allows us to properly define the value of communication is a **Bayesian game**. This model combines the strategic game of chapter 3 with the partial observability concepts of chapter 5. In particular, we assume a number of agents, a set S of world states, and an information function for each agent i that gives rise to a partition $\mathcal{P}_i$ of S, different for each agent. We also assume that each state $s \in S$ occurs with some (prior) probability that is assumed equal for all agents, and that it defines a strategic game G_s with corresponding payoffs.

	L	R
U	$2, 2$	$1, 0$
D	$0, 1$	$0, 0$

G_a (probability 0.7)

	L	R
U	$0, 0$	$0, 1$
D	$-1, 0$	$2, 2$

G_b (probability 0.3)

Figure 6.1: A Bayesian game defined by two strategic games, G_a and G_b.

We will explain by an example how the model of a Bayesian game can be used for defining the value of a communicative act. Suppose there are two agents, the state space is $S = \{a, b\}$, and the information partitions of the agents are $\mathcal{P}_1 = \{\{a\}, \{b\}\}$ and $\mathcal{P}_2 = \{\{a, b\}\}$ and they are common knowledge among the agents. In any true state s, agent 1 knows s, but agent 2 does not know s, and agent 1 knows that agent 2 does not know s. Suppose also that state a occurs with probability 0.7 and state b with probability 0.3, and let each of them define a strategic game as shown in Fig. 6.1.

In this problem, the actions of the agents should depend on what state is actually realized, in other words, which one of G_a or G_b actually applies. Note that, contrary to the coordination problems of chapter 4 where the agents had to choose among several equilibria, here each of the two games has a single equilibrium which is trivial to find; in $s = a$ the agents should choose (U, L), and in $s = b$ they should choose (D, R). However, agent 2 does not know what the true state is, and thus it is difficult for him to predict the action of agent 1 and find the correct equilibrium.

Given the partial knowledge of the agents about the true state, the rational joint action (Nash equilibrium) for the agents is (U, L) which gives to each of them *expected* payoff $0.7*2+0.3*0 = 1.4$. Now, if in any true state s agent 1 informs agent 2 about s with a communicative act, the partition of agent 2 will read $\mathcal{P}_2 = \{\{a\}, \{b\}\}$ and the agents can now safely choose the correct equilibrium for each value of s. This gives to each of them payoff 2. The difference $2 - 1.4 = 0.6$ of the expected payoffs of each agent before and after communication defines the value of the corresponding communicative act.

In a similar manner an agent can compute the value of all its available communicative acts in some situation, and then choose the one with the highest value. In practice, however, taking an optimal decision about what and with whom to communicate may require computational resources that go beyond the capacity of an agent. For a **bounded rational** agent with memory and time restrictions, there is always a trade-off between choosing a good communicative act (or any other action) and choosing it quickly. We could also think of having each agent broadcast its private information to all other agents in every time step, but this could lead to communication bottlenecks or other effects.

For each agent i in parallel
  If $i \neq 1$
    Wait until all actions $(a_1^*, \ldots, a_{i-1}^*)$ are received.
  End
  Compute an equilibrium that contains $(a_1^*, \ldots, a_{i-1}^*, \ldots)$.
  Choose component a_i^* from this equilibrium.
  Broadcast action a_i^* to all agents $i+1, \ldots, n$.
End

Figure 6.2: Coordination by broadcast.

6.4 Coordination via communication

In chapter 4 we studied several coordination algorithms for cooperative agents. These algorithms relied on several common knowledge assumptions. In this section we relax some of these assumptions, and show how the algorithms can be modified to explicitly take communication into account. An advantage of using communication is that an agent does not need to predict the actions of all other agents at the equilibrium anymore.

Social conventions

When communication is not available, coordination by social conventions is based on an ordering scheme of joint actions that has been a priori defined and is common knowledge among agents. As we explained in section 4.3, such an ordering scheme can be realized in a lexicographic manner, by first ordering the agents, and then the actions of the agents.

When communication is available, we only need to impose an ordering $i = 1, \ldots, n$ of the agents that is common knowledge. Coordination can now be achieved by the algorithm of Fig. 6.2. Each agent i (except agent 1) waits until all previous agents $1, \ldots, i-1$ in the ordering have broadcast their chosen actions. Then, agent i computes its component of the equilibrium that agrees with the choices of the previous agents, that is $(a_1^*, \ldots, a_{i-1}^*, a_i^*, \ldots)$, and broadcasts a_i^* to all agents that have not chosen an action yet. Note that, unlike the communication-free coordination algorithm of Fig. 4.2 where each agent runs the same procedure identically and in parallel, here the fixed ordering of the agents together with the wait/broadcast primitives result in a synchronized sequential execution order.

Note that in the above algorithm an agent needs to compute only its *own* component of *any* equilibrium that is consistent with the previously broadcast actions, while its communication-free counterpart of Fig. 4.2 requires that each agent computes all equilibria in the game which can be costly.

For each agent i in parallel
    $I = \{\}$.
    For each role $j = 1, \ldots, n$
        Compute the potential r_{ij} of agent i for role j.
        Broadcast r_{ij} to all agents.
    End
    Wait until all $r_{i'j}$, for $j = 1, \ldots, n$, are received.
    For each role $j = 1, \ldots, n$
        Assign role j to agent $i^* = \arg\max_{i' \notin I}\{r_{i'j}\}$.
        Add i^* to I.
    End
End

Figure 6.3: Role assignment by broadcast.

Role assignment

The role assignment algorithm of Fig. 4.3 was based on the assumption that each agent is able to compute the potential of each other agent for a particular role. In practice, however, this is not always possible; in a partially observable domain, a potential function may require elements of the state that are only privately perceived by an agent (a soccer robot may know how close it is to the ball, but may not know how close its teammates are to the ball). In some applications it seems more natural to have each agent compute by itself how suitable it is for a particular role.

When communication is available, role assignment requires that an agent computes only its own potentials for the set of roles, and then broadcasts them to the rest of the agents. The agent then waits for all other potentials to arrive, and finally computes the assignment of roles to agents as in the communication-free role assignment algorithm of Fig. 4.3. Here too we assume a fixed ordering $j = 1, \ldots, n$ of the set of roles and a potential function for each role that are common knowledge among the agents. The algorithm is shown in Fig. 6.3. Note that the broadcast of the potentials to all agents allows each agent to compute the full assignment of all roles to all agents. As in Fig. 4.3, agent i will be assigned role j when $i^* = i$, which will occur only once: every agent i^* that is assigned a role is added to I, and is not considered again in the set of candidates i' for a new role.

Concerning the computational requirements, each agent now needs to compute only $O(n)$ (its own) potentials instead of $O(n^2)$ in the algorithm of Fig. 4.3, but this will be compensated by the total number $O(n^2)$ of potentials that need to be broadcast and processed by the agents. Note that the maximization step over $\{r_{i'j}\}$ is the same in both algorithms.

For each agent i in parallel
  If $i \neq 1$
    Wait until agent $i-1$ sends OK.
  End
  Let $f_j(a_{-i}, a_i)$ be all local payoffs (initial and communicated) that involve agent i.
  Compute $B_i(a_{-i}) = \arg\max_{a_i} \sum_j f_j(a_{-i}, a_i)$.
  Compute $f^*(a_{-i}) = \max_{a_i} \sum_j f_j(a_{-i}, a_i)$.
  Send $f^*(a_{-i})$ to agent $j = \min\{i+1, \ldots, n\},\ j \in -i$.
  If $i \neq n$
    Send OK to agent $i+1$.
    Wait until all a^*_{-i} are received.
  End
  Choose any $a^*_i \in B_i(a^*_{-i})$.
  Broadcast a^*_i to all agents j such that $a_i \in \text{domain}(B_j)$.
End

Figure 6.4: Variable elimination via communication.

Variable elimination

The forward pass of the variable elimination algorithm of Fig. 4.5, where each agent i computes its best-response function $B_i(a_{-i})$, relies on the assumption that all local payoff functions f_j are common knowledge among agents. Similarly, in the backward pass, where an agent informs the other agents of its action choice, we have assumed that the actions of the agents are ordered and these orderings are common knowledge. The effect of these two common knowledge assumptions is that each agent can run the algorithm in parallel, choosing an arbitrary elimination order.

When communication is available, these two assumptions are not needed anymore. In the forward pass, each agent can maintain in its local memory the payoff functions that involve only this agent. The initial distribution of payoff functions to the agents can be done as follows: agent 1 in the elimination order takes all payoff functions that involve this agent, agent 2 takes all functions that involve this agent and are not distributed to agent 1, and so on, until no more payoff functions are left. When an agent computes its best-response function and generates a new payoff function, the agent can broadcast this function to the other agents involved in it. In fact, the agent needs to send the payoff function only to the first non-eliminated agent whose action appears in the domain of this function. Similarly, in the backward pass an agent can wait for the optimal actions of the other agents (unless it is the last eliminated agent), then choose *any* action from

its best-response action set, and finally broadcast this action to all agents that need it in their best-response functions.

The complete algorithm is shown in Fig. 6.4. A crucial difference with the communication-free variable elimination algorithm of Fig. 4.5 is that here the elimination order $i = 1, \ldots, n$ of the agents is a priori fixed and is common knowledge among the agents. The OK signal is required for synchronization, ensuring an execution order of the algorithm according to the elimination order of the agents.

In terms of complexity, the forward pass is slightly slower than in the communication-free case, because here the generated payoffs need to be communicated to the other involved agents. On the other hand, when communication is available the backward pass is fully asynchronous. One can also think of asynchronous versions of the forward pass in which many agents are simultaneously eliminated. This would require some additional bookkeeping for storing the pairwise dependencies between agents.

6.5 Notes and further reading

Speech acts have been studied by Searle (1969). Bayesian games are briefly covered in (Osborne and Rubinstein, 1994). The role assignment algorithm of Fig. 6.3 has been proposed by Castelpietra et al. (2000). Gmytrasiewicz and Durfee (2001) analyze communication in a Bayesian game setting where the agents model the knowledge of each other recursively. The latter provide also a framework for multiagent coordination. Among early approaches in distributed AI to communication-based coordination is the 'contract net protocol' of Smith (1980) where tasks are dynamically distributed among agents using a bidding mechanism (see also chapter 7), and the 'partial global planning' algorithm of Durfee and Lesser (1987) and Decker and Lesser (1995) in which agents exchange and refine local plans in order to reach a common goal. The variable elimination algorithm of Fig. 6.4 is due to Guestrin et al. (2002a).

Chapter 7

Mechanism design

In this chapter we study the problem of mechanism design, which is the development of agent interaction protocols that explicitly take into account the fact that the agents may be self-interested. We discuss the revelation principle and the Groves-Clarke family of mechanisms that allow us to build successful protocols in a wide variety of practical problems.

7.1 Self-interested agents

In the previous chapters we mainly studied multiagent systems that consist of cooperative agents. The fact that the agents work cooperatively for a common goal allowed us to develop algorithms, like the coordination algorithms of chapter 4 and 6, in which the agents are assumed to be sincere to each other and behave as instructed. A soccer agent, for instance, would never violate a protocol that assigns roles to teammates (see Fig. 4.3), because this could potentially harm the performance of his team.

In many practical applications, however, we have to deal with **self-interested** agents, for example, agents that act on behalf of some owner that wants to maximize his or her own profit. A typical example is a software agent that participates in an electronic auction on the Internet. Developing an algorithm or protocol for such a system is a much more challenging task than in the cooperative case. First, we have to motivate an agent to participate in the protocol, which is not *a priori* the case. Second, we have to take into account the fact that an agent may try to *manipulate* the protocol for his own interest, leading to suboptimal results. The latter includes the possibility that the agent may lie, if needed.

The development of protocols that are stable (nonmanipulable) and individually rational for the agents (no agent is worse off by participating) is the subject of **mechanism design** or **implementation theory**. As we will see next, a standard way to deal with the above two problems is to provide payments to the agents in exchange for their services.

7.2 The mechanism design problem

In chapter 3 we used the model of a strategic game to describe a situation in which a group of agents interact with each other. The primitives of such a game are the action sets A_i and the payoff functions $u_i(a)$ of the agents, for $i = 1, \ldots, n$, where $u_i(a)$ reflects the preference of agent i for the joint action a. Moreover, for any profile of payoff functions, a solution concept (e.g., Nash equilibrium) allows us to make predictions over the set of outcomes that may result when the game is played. Our standpoint in chapter 3 was that of an external observer who wants to know the outcome of a game, but cannot affect this outcome in any way.

In mechanism design we go one step further. Here we assume a set $\mathcal{O}$ of possible **outcomes** over which a number of agents form preferences. Our task is to *design* a game that, when played by the agents, brings about a desired outcome from $\mathcal{O}$. In this framework we therefore use a game as a tool for achieving our design goals. An outcome can be practically anything, for example, the assignment of a resource to an agent. The main difficulty in mechanism design is that we often do not know the preferences of the agents in advance.

More formally, in each world state we assume that each agent i receives some private information $\theta_i \in \Theta_i$, called the **type** of the agent, which is not revealed to the other agents or to us (the mechanism designer). We can think of a profile $\theta = (\theta_i)$ of agent types as a world state, and in each state θ agent i considers possible all states in which his type is θ_i (see chapter 5). Moreover, we assume that the type of an agent fully specifies the preferences of this agent over the set of outcomes $o \in \mathcal{O}$. In particular, each agent i has a **valuation function** $\nu_i(o, \theta_i)$, that is parametrized on θ_i, such that agent i in type θ_i prefers outcome o to o' if and only if $\nu_i(o, \theta_i) > \nu_i(o', \theta_i)$. We will assume that in each state θ we know the valuation function of each agent i (but not his type), and this fact is common knowledge among all agents.

In a mechanism design problem we hold a **social choice function** $f(\theta)$ that, for each profile $\theta = (\theta_i)$ of agent types, produces a desired outcome $o = f(\theta)$. We can think of f as an algorithm that solves an optimization problem: given n inputs $\theta = (\theta_i)$, the function f computes an outcome o that maximizes a functional over the set of agents valuations. A typical case, which we will examine further in section 7.4, is to select the outcome that maximizes the sum of the agents valuations given their types:

$$f(\theta) = \arg\max_{o \in \mathcal{O}} \sum_{i=1}^{n} \nu_i(o, \theta_i). \tag{7.1}$$

Such a social choice function is called **allocatively-efficient**.

Implementing a social choice function would be easy if we had full observability of the state θ. Then we could just use θ in (7.1) and compute

the desired optimal outcome (assuming, of course, that we have a tractable algorithm for doing this). However, as we saw above, θ_i is revealed only to agent i. One option would be to ask each agent to tell us his type, but there is no guarantee that an agent will report his true type! Recall that each agent i forms his own preferences over outcomes, given by his valuation function $\nu_i(o, \theta_i)$ with θ_i his true type. If by reporting a false type $\tilde{\theta}_i \neq \theta_i$ an agent i expects to receive higher payoff than by reporting θ_i, then this agent may certainly consider lying. As we mentioned in the introduction, in mechanism design our starting point is that the agents are self-interest. Viewed from a computational perspective, we can therefore characterize mechanism design as follows:

Definition 7.2.1. Mechanism design is the development of efficient algorithms for optimization problems in which some of the parameters of the objective function are in the control of agents that have different preferences for different solutions.

The challenge is therefore to design mechanisms that steer the agents toward selecting the desired $o = f(\theta)$ by themselves, for any profile of their types. However, we first need to motivate the agents to participate in such a mechanism—participation is not a priori the case! To accomplish this, we define a **payment function** $p(o)$ that associates with each outcome o a profile $p(o) = (p_i(o))$ of payments, so that agent i receives payment $p_i(o)$ when o occurs. The valuation function together with the payment function define the **payoff function** of an agent (we assume that payments and valuations are expressed in the same units):

$$u_i(o, \theta_i) = \nu_i(o, \theta_i) + p_i(o). \tag{7.2}$$

A mechanism that implements a social choice function $f(\theta)$ where no agent is worse off by participating, that is, $u_i(f(\theta), \theta_i) \geq 0$ for all i and all θ, is called **individually rational**.

We focus here on simple mechanisms in the form of a strategic game $\mathcal{M} = (A_i, g, p)$ where A_i is a set of available actions for agent i, $g(a) = o$ is an **outcome function** that maps a joint action a to an outcome $o \in \mathcal{O}$, and $p = p(o)$ is a payment function that controls the agents preferences through (7.2). When the agents play the game $\mathcal{M}$, we expect them to choose a joint action $a^*(\theta) = (a_i^*(\theta_i))$ according to some solution concept, where the action $a_i^*(\theta_i)$ of an agent i will typically depend on his type θ_i. This joint action is then mapped through g to an outcome $g(a^*(\theta))$ which we want to equal $f(\theta)$. As solution concept we will consider the following:

Definition 7.2.2. A joint action $a^* = (a_i^*)$ is an **equilibrium in dominant actions** if for every agent i holds

$$u_i(a_{-i}, a_i^*) \geq u_i(a_{-i}, a_i) \tag{7.3}$$

for all joint actions (a_{-i}, a_i).

Our choice of such a solution concept is motivated by the fact that we want to design a mechanism in which each agent i can compute his optimal action a_i^* without having to worry about the actions of the other agents. In terms of predictive power for the solutions of a game, an equilibrium $a^* = (a_i^*)$ in dominant actions is weaker than both a Nash equilibrium and an equilibrium computed by iterated elimination of strictly dominated actions (see chapter 3). However, in the context of mechanism design, the existence of such an equilibrium guarantees that every (rational) agent will adhere to it, even if he has no information about the action choice of the other agents. Such an equilibrium solution is also very attractive computationally, because an agent does not need to consider the actions of the other agents anymore. Summarizing, the mechanism design problem can be defined as follows:

Definition 7.2.3 (The mechanism design problem). Given a set of outcomes $o \in \mathcal{O}$, a profile of valuation functions $\nu_i(o, \theta_i)$ parametrized by θ_i, and a social choice function $f(\theta)$, find appropriate action sets A_i, an outcome function $g(a) = o$, and a payment function $p(o)$, such that for any profile $\theta = (\theta_i)$, and for payoff functions $u_i(o, \theta_i)$ defined via (7.2), holds $g(a^*(\theta)) = f(\theta)$ where $a^*(\theta) = (a_i^*(\theta_i))$ is an equilibrium solution in dominant actions of the strategic game $\mathcal{M} = (A_i, g, p)$. In this case we say that the mechanism $\mathcal{M}$ **implements** the social choice function f in dominant actions.

Example: second-price auction

Consider the following mechanism design problem (an auction). We have n agents and an item (e.g., a resource in a computer network). We want to assign the item to the agent that values it most, but we do not know the true valuations of the agents. In this example, the type θ_i of an agent is his valuation, with $\theta_i \in \mathbb{R}^+$, and an outcome $o \in \{1, \ldots, n\}$ is the index of the agent to whom the item is assigned. The valuation function of an agent i with type i is $\nu_i(o, \theta_i) = \theta_i$ if $o = i$ and zero otherwise. The social choice function is $f(\theta_1, \ldots, \theta_n) = \arg\max_i\{\theta_i\}$ (note that this is a simplified version of (7.1)). If we do not include a payment function, that is $p_i(o) = 0$ for all i, then a mechanism $\mathcal{M}_1 = (A_i, g, 0)$ that implements f is always individually rational because for an agent i holds $u_i(o, \theta_i) = \nu_i(o, \theta_i) = \theta_i \geq 0$.

Suppose now that we incorporate a payment function $p(o)$ such that the agent who receives the item must pay *tax* (negative payment) equal to the second highest valuation, whereas the other agents do not have to pay anything. In this case, the payoff function of an agent i with valuation θ_i equals $u_i(o, \theta_i) = \theta_i + p_i$ if $o = i$ and zero otherwise, where $p_i = -\max_{j \neq i}\{\theta_j\}$. A mechanism $\mathcal{M}_2 = (A_i, g, p)$ with such a payment function p that implements f is still individually rational because for the winning agent k holds $u_k(k, \theta_k) = \theta_k - \max_{j \neq k}\{\theta_j\} = \max_j\{\theta_j\} - \max_{j \neq k}\{\theta_j\} \geq 0$, while for the other agents $j \neq k$ holds $u_j(k, \theta_j) = 0$.

7.3 The revelation principle

Looking at definition 7.2.3, mechanism design seems a formidable task. Our design options can in principle involve all possible action sets A_i, all possible outcome functions g, and all kinds of payments p that we could provide to the agents. Searching in the space of all $\mathcal{M} = (A_i, g, p)$ for a mechanism that implements f would be infeasible. Fortunately, there is a theorem that tells us that we do not need to search in the space of all possible mechanisms.

Proposition 7.3.1 (Revelation principle). *If a social choice function f is implementable in dominant actions by a mechanism $\mathcal{M} = (A_i, g, p)$, then f is also implementable in dominant actions by mechanism $\mathcal{M}' = (\Theta_i, f, p)$ where each agent is simply asked to report his type. Moreover, the dominant action of agent i in $\mathcal{M}'$ is to report his true type θ_i.*

Proof. Suppose truth-telling is not a dominant action in $\mathcal{M}'$. Then there must be a profile of reported types $\tilde{\theta} = (\theta_i)$ for which

$$u_i(f(\tilde{\theta}_{-i}, \tilde{\theta}_i), \theta_i) > u_i(f(\tilde{\theta}_{-i}, \theta_i), \theta_i). \tag{7.4}$$

If f is implementable by $\mathcal{M}$ in dominant actions, then $f(\theta) = g(a^*(\theta))$ for all θ, where $a^*(\theta) = (a_i^*(\theta_i))$ is an equilibrium in dominant actions in $\mathcal{M}$. We can therefore rewrite (7.4) as

$$u_i(g(a_{-i}^*(\tilde{\theta}_{-i}), a_i^*(\tilde{\theta}_i)), \theta_i) > u_i(g(a_{-i}^*(\tilde{\theta}_{-i}), a_i^*(\theta_i)), \theta_i) \tag{7.5}$$

which contradicts the fact that $a_i^*(\theta_i)$ is a dominant action in $\mathcal{M}$. Thus the profile of true types $\theta = (\theta_i)$ must be an equilibrium in dominant actions in $\mathcal{M}'$, from which directly follows that $\mathcal{M}'$ implements f. □

A mechanism in the form $\mathcal{M} = (\Theta_i, f, p)$ in which each agent is asked to report his type is called a **direct-revelation** mechanism. A direct-revelation mechanism in which truth-telling is the dominant action for every agent is called **strategy-proof**. The revelation principle is remarkable because it allows us to restrict our attention to strategy-proof mechanisms only. One of its consequences, for example, is that if we cannot implement a social choice function by a strategy-proof mechanism, then there is no way to implement this function in dominant actions by *any* other general mechanism. The revelation principle has been proven a powerful theoretical tool for establishing several possibility and impossibility results in mechanism design (see, e.g., (Parkes, 2001) for details and references).

Example: second-price sealed-bid (Vickrey) auction

Let us return to the auction example, and consider a direct-revelation mechanism $\mathcal{M}_3(\Theta_i, f, p)$ with p as in $\mathcal{M}_2$. In other words, each agent is asked to

bid a price, and the item is allocated to the agent with the highest bid, but he has to pay the second highest bid. We will show that in mechanism $\mathcal{M}_3$ truth-telling is a dominant action for each agent, that is, each agent must bid his true valuation.

Let $b_i(\theta_i)$ be the bid of agent i given that his true valuation is θ_i, and $b' = \max_{j \neq i}\{b_j(\theta_j)\}$ be the highest bid among all other agents. The payoff of agent i is $u_i(o, \theta_i) = \theta_i - b'$ if $b_i(\theta_i) > b'$ and zero otherwise. Ignoring ties, if $b' < \theta_i$ then any bid $b_i(\theta_i) > b'$ is optimal (results in positive payoff $u_i(i, \theta_i) = \theta_i - b' > 0$). If $b' > \theta_i$ then any bid $b_i(\theta_i) < b'$ is optimal (results in nonnegative payoff). Truth-telling bid $b_i(\theta_i) = \theta_i$ is optimal in both cases, and is thus a dominant action in $\mathcal{M}_3$.

7.4 The Groves-Clarke mechanism

The second-price sealed-bid auction that we saw above, implements the social choice function $f(\theta_1, \ldots, \theta_n) = \arg\max_i\{\theta_i\}$ which can be regarded as a special case of the allocatively-efficient social choice function (7.1). We return now to the more general case. We assume a direct mechanim in which the agents are asked to report their types, and based on their reports $\tilde{\theta} = (\tilde{\theta}_i)$ the mechanism computes an optimal outcome $f(\tilde{\theta})$ that solves

$$f(\tilde{\theta}) = \arg\max_{o \in \mathcal{O}} \sum_{i=1}^{n} \nu_i(o, \tilde{\theta}_i). \tag{7.6}$$

In a **Groves** mechanism, the payment function that is associated with outcome $f(\tilde{\theta})$ is defined for each agent as

$$p_i(f(\tilde{\theta})) = \sum_{j \neq i} \nu_j(f(\tilde{\theta}), \tilde{\theta}_j) - h_i(\tilde{\theta}_{-i}), \tag{7.7}$$

for arbitrary function $h_i(\tilde{\theta}_{-i})$ that does not depend on the report of agent i. In this case, and for payoffs given by (7.2), we can show the following (the proof is left as an exercise):

Proposition 7.4.1. *A Groves mechanism is a strategy-proof mechanism.*

Having the freedom to choose any function $h_i(\tilde{\theta}_{-i})$, the **Clarke** mechanism uses

$$h_i(\tilde{\theta}_{-i}) = \sum_{j \neq i} \nu_j(f'(\tilde{\theta}_{-i}), \tilde{\theta}_j) \tag{7.8}$$

where $f'(\tilde{\theta}_{-i})$ is an allocatively-efficient social choice function with agent i excluded:

$$f'(\theta) = \arg\max_{o \in \mathcal{O}} \sum_{j \neq i} \nu_j(o, \theta_j). \tag{7.9}$$

Under quite general conditions, the Clarke mechanism can be shown to be individually rational. Moreover, in some applications the payments p_i to the agents are negative, so the mechanism does not need to be externally subsidized (however, the collected tax must be burnt).

Example: shortest path

This is classical example with many applications, that is based on the Groves-Clarke mechanism. We want to compute the shortest path between two fixed nodes in a graph. Each edge i in the graph has cost (length) $\theta_i \geq 0$, and is operated by an agent (e.g., a transport company) who would preferably stay out of the path. We do not know the cost of each edge in advance, and we want to design a direct mechanism in which each agent reports his true cost.

Translated in the language of mechanism design, an outcome o is an ordered list of agents indices (the edges that are included in the shortest path); agent i has type θ_i (the cost of his edge), and valuation function $\nu_i(o, \theta_i) = -\theta_i$ if $i \in o$ and zero otherwise; and the social choice function $f(\tilde{\theta})$ is an algorithm (e.g., Dijkstra) that solves (7.6) (computes the shortest path given the reported costs).

A Clarke mechanism solves the above problem by providing nonzero payments to all agents i that are included in a shortest path solution. These payments are computed from (7.7) and (7.8):

$$p_i(f(\tilde{\theta})) = \sum_{j \neq i} \nu_j(f(\tilde{\theta}), \tilde{\theta}_j) - \sum_{j \neq i} \nu_j(f'(\tilde{\theta}_{-i}), \tilde{\theta}_j) = \tilde{\theta}_i - C + C' \qquad (7.10)$$

where C is the additive cost (length) of the shortest path solution, and C' is the length of the shortest path solution after edge i is removed from the graph. From (7.2) and (7.10), the payoff of agent i under truth-telling is $u_i = -\theta_i + \theta_i - C + C'$, which is always nonnegative since removing an edge from a graph can never generate a shorter path. It is therefore individually rational for an agent to participate in this mechanism, and because Groves-Clarke mechanisms are strategy-proof, each agent will gladly report his true cost.

7.5 Notes and further reading

Our exposition was partly based on Osborne and Rubinstein (1994, chap. 10), Parkes (2001, chap. 2), Nisan (1999), and Sandholm (1999). The papers of Vickrey (1961), Clarke (1971), and Groves (1973) are seminal. The revelation principle is due to Gibbard (1973). Computational issues in mechanism design are discussed by Nisan (1999), Parkes (2001), and Papadimitriou (2001). The latter writes: "All design problems [in computer science] are now mechanism design problems".

Chapter 8

Learning

In this chapter we briefly address the issue of learning, in particular reinforcement learning which allows agents to learn from delayed rewards. We outline existing techniques for single-agent systems, and show how they can be extended in the multiagent case.

8.1 Reinforcement learning

Reinforcement learning (RL) is a generic name given to a family of techniques in which an agent tries to learn a task by directly interacting with the environment. The method has its roots in the study of animal behavior under the influence of external stimuli. In the last two decades, RL has been extensively studied in artificial intelligence, where the emphasis is on how agents can improve their performance in a given task by perception and trial-and-error. The field of single-agent RL is nowadays mature, with well-understood theoretical results and many practical techniques (Sutton and Barto, 1998).

On the contrary, the field of **multiagent reinforcement learning** in which many agents are simultaneously learning by interacting with the environment and with each other, is less mature. The main reason is that many theoretical results for single-agent RL do not directly apply in the case of multiple agents. There are also computational issues like the difficulty of dealing with exponentially large state/action spaces, and the intractability of several distributed decision making algorithms (Bernstein et al., 2000). Recent efforts involve linking multiagent RL with game-theoretic models of learning, with promising results (Claus and Boutilier, 1998; Wang and Sandholm, 2003).

In the rest of the chapter we will present two popular learning algorithms for single-agent systems, value iteration and Q-learning, and show how they can be extended to the multiagent case. For simplicity, we will focus on cooperative multiagent systems only.

8.2 Markov decision processes

In this section we address the single-agent case. The **sequential decision making** of an agent in an observable stochastic world with Markovian transition model is called a **Markov decision process** (MDP). An MDP extends the decision making model of section 2.4 by allowing an agent to take consecutive actions $a_t, a_{t+1}, \ldots$, one action per time step t. In an MDP we assume that in each state s_t at time t the agent receives from the environment an immediate **reward** or **reinforcement** $R(s_t) \in \mathbb{R}$. The task of the agent is to maximize its total **discounted future reward** $R(s_t) + \gamma R(s_{t+1}) + \gamma^2 R(s_{t+2}) + \cdots$, where $\gamma \in [0, 1]$ is a discount rate that ensures that even with infinite sequences the sum is finite. Clearly, the discounted future reward will depend on the particular policy of the agent, because different policies result in different paths in the state space.

Given the above, the optimal utility of a state s for a particular agent can be defined as the maximum discounted future reward this agent can receive in state s by following some policy:

$$U^*(s) = \max_\pi E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s\right] \tag{8.1}$$

where the expectation operator $E[\cdot]$ averages over rewards and stochastic transitions. Similarly, we can define an optimal action value $Q^*(s, a)$ as the maximum discounted future reward the agent can receive after taking action a in state s. A policy $\pi^*(s)$ that maximizes the above expression is called an **optimal policy**, for which, as we already saw in section 2.4, the greedy property (2.5) holds. We should note that there can be many optimal policies in a given task, but they all share a unique $U^*(s)$ and $Q^*(s, a)$.

If we combine (8.1) with (2.5) we get a recursive definition of optimal utility:

$$U^*(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s, a) U^*(s'). \tag{8.2}$$

This is called the **Bellman equation**, and the solutions of this set of equations (one for each state) define the optimal utility of each state. A similar recursive definition holds for action values:

$$Q^*(s, a) = R(s) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q^*(s', a'). \tag{8.3}$$

Value iteration

A simple and efficient method for computing optimal utilities in an MDP when the transition model is available is **value iteration**. We start with random utility values $U(s)$ for each state and then iteratively apply (8.2)

turned into an assignment operation:

$$U(s) := R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)U(s'). \tag{8.4}$$

We repeat until convergence which is measured in relative increase in $U(s)$ between two successive update steps. Value iteration converges to the optimal $U^*(s)$ for each state. As we described in section 2.4, from the optimal utilities we can then easily compute an optimal policy $\pi^*(s)$ by using (2.5). For example, using value iteration in the world of Fig. 2.1 with a reward of $R(s) = -1/30$ for each nonterminal state and $\gamma = 1$ (no discounting), we get the optimal utilities and the optimal policy shown in Fig. 2.2.

Q-learning

One of the disadvantages of value iteration is that it assumes knowledge of the transition model $P(s'|s,a)$. However, in many applications the transition model is unavailable, and we would like to have a learning method that does not require a model. **Q-learning** is such a **model-free** method in which an agent repeatedly interacts with the environment and tries to estimate the optimal $Q^*(s,a)$ by trial-and-error. In particular, the agent starts with random estimates $Q(s,a)$ for each state-action pair, and then begins exploring the environment. During exploration it receives tuples in the form (s, R, a, s') where s is the current state, R is the current reward, a is an action taken in state s, and s' is the resulting state after executing a. From each tuple, the agent updates its action value estimates as

$$Q(s,a) := (1-\lambda)Q(s,a) + \lambda\big[R + \gamma \max_{a'} Q(s',a')\big], \tag{8.5}$$

where $\lambda \in (0,1)$ is a learning rate that controls convergence. Note that the maximization in (8.5) is over all actions a' from the resulting state s'.

If all state-action pairs are visited infinitely often and λ decreases slowly with time, Q-learning can been shown to converge to the optimal $Q^*(s,a)$. Moreover, this holds irrespective of the particular **exploration policy** by which the agent selects its actions a above. A common choice is the so-called **ϵ-greedy** policy by which in state s the agent selects a random action with probability ϵ, and action $a = \arg\max_{a'} Q(s,a')$ with probability $1-\epsilon$, where $\epsilon < 1$ is a small number. Alternatively, the agent can choose exploration action a in state s according to a **Boltzmann** distribution

$$p(a|s) = \frac{\exp(Q(s,a)/\tau)}{\sum_{a'} \exp(Q(s,a')/\tau)}, \tag{8.6}$$

where τ controls the smoothness of the distribution (and thus the randomness of the choice), and is decreasing with time.

8.3 Multiagent reinforcement learning

In this section we examine how RL techniques, like value iteration and Q-learning, can be extended to the case of multiple agents. Extending RL to multiagent systems involves several issues, for instance whether the agents receive the same rewards, whether they know the payoffs of each other in a local game, whether they observe the selected joint actions, whether they model each other, etc. For simplicity, we will only deal with cooperative multiagent systems, in which all agents receive the same reward (or a reward sampled from the same distribution) in each time step.

As in chapters 3 and 4, we assume here that each state $s \in S$ is fully observable to all agents, and defines a local strategic game G_s with corresponding payoffs for each joint action of the agents. Since we restrict attention to cooperative systems, the true payoff of a joint action is assumed to be the same for all agents, reflecting average discounted future reward if this action is taken from the particular state. However, an agent will not be aware of the true payoffs of G_s, and this is what he must learn.

We additionally assume that the world evolves with a stochastic transition model $p(s'|s,a)$, where s is the current state, a is the *joint* action of the agents, and s' is the resulting state after a is executed. The transition model will typically be unknown to the agents. As in single-agent RL, the task of the agents is to compute an optimal **joint policy** $\pi^*(s) = (\pi_i^*(s))$ that maximizes discounted future reward in the specific environment $p(s'|s,a)$. The challenge in multiagent RL is to guarantee that the individual optimal policies $\pi_i^*(s)$ are *coordinated*, that is, they indeed define an optimal joint policy $\pi^*(s)$.

Independent learning

The simplest case is when the agents learn independently of each other. That is, each agent treats the other agents as part of the environment, and does not attempt to model them or predict their actions. In this case it is more natural for an agent to use Q-learning to compute its optimal policy, because even knowledge of $p(s'|s,a)$ does not imply knowledge of $p(s'|s,a_i)$, where a_i is the action of agent i. Note that the way the world changes when agent i takes action a_i in state s depends also on the actions of the other agents in s, and since agent i does not model the other agents there is no way for him to compute $p(s'|s,a_i)$ (or an informative approximation of it). But even Q-learning may not result in coordinated individual policies because its convergence relies on an underlying transition model that is stationary, i.e., does not change with time. This is not the case with $p(s'|s,a_i)$ which is affected by the policy of the other agents who are also simultaneously learning.

Although the use of Q-learning by independent agents cannot be justi-

fied theoretically, the method has been employed in practice with reported success (Tan, 1993; Sen et al., 1994; Matarić, 1994). Claus and Boutilier (1998) examine the conditions under which independent Q-learning leads to individual policies that form a Nash equilibrium in a single state coordination problem, concluding that under general conditions the method converges. However, the resulting equilibrium may not be Pareto optimal (see chapter 4). Similarly, Wolpert et al. (1999) have shown that for a constrained class of problems independent Q-learning may converge to a Nash equilibrium.

Joint action learning

Better results can be obtained if the agents attempt to model each other. In this case, each agent maintains an action value function $Q^{(i)}(s, a)$ for all state and *joint* action pairs. Every time a joint action a is taken in state s and a new state s' is observed, each agent i updates his $Q^{(i)}(s, a)$. This Q-value reflects the value of joint action a in state s as modeled by the particular agent. Note that this framework requires that each agent observes the taken joint actions. If all agents receive exactly the same reward in each state, then clearly all $Q^{(i)}(s, a)$ will be identical during Q-learning. However, it may happen that the rewards the agents receive are different samples from the same distribution, and thus not identical. In this case the individual $Q^{(i)}(s, a)$ may differ during Q-learning.

In general, in joint action learning we have to consider the following issues:

Representation: how an agent should represent $Q^{(i)}(s, a)$.

Optimization: how he can compute $\arg\max_{a'} Q(s', a')$ in (8.5).

Exploration: how he should select an exploration action a in (8.5).

With regard to representation, each agent can in principle choose his own representation. The simplest choice is to use a *tabular* representation in which each $Q^{(i)}(s, a)$ is a matrix with as many entries as the pairs of states $s \in S$ and joint actions $a \in \times_i A_i$. Alternatively, if many agents are involved, a coordination graph can be used (see section 4.5). In this case the assumption is that the true payoffs can be decomposed as $Q(s, a) = \sum_j Q_j(s, a_j)$ where a_j denotes the joint action of a subset of agents. Moreover, if communication is available, each agent can only maintain those local payoff functions $Q_j(s, a_j)$ that involve him, as in the communication-based coordination of chapter 6. If the state space is large, a *functional* representation of $Q_j(s, a_j)$ using, for instance, a neural network, may be more appropriate than a tabular representation (Guestrin et al., 2002b).

Concerning the maximization step $\arg\max_{a'} Q(s', a')$ in (8.5), an agent needs to search over all a' for the joint action that maximizes $Q^{(i)}(s', a')$

(hoping that the other estimates $Q^{(j\neq i)}(s', a')$ do not differ too much from $Q^{(i)}(s', a')$, see also (Wang and Sandholm, 2003)). In a tabular representation the maximization involves just a pass through all entries (s', a') for given s'. In a coordination graph representation this can be done with variable elimination as we explained in section 4.5.

Finally, in the next section we discuss the issue of optimal exploration.

8.4 Exploration policies

An important issue in multiagent RL is how an agent chooses his exploration policy. We focus again on the cooperative case only. If all observed rewards are exactly equal, a simple method is to select a joint action in state s according to a Boltzmann distribution over joint actions using the current $Q^{(i)}(s, a)$ (which will be the same for all agents). Each agent can sample a joint action from this distribution by using the same random number generator (and same seed). This ensures that all agents will sample the same exploration action a. Then each agent can select his action a_i as the component i of the selected a.

Q-learning with an exploration policy like the above and common knowledge assumptions about parameters like the random number generator and seed, implies in effect that each agent runs Q-learning over joint actions identically and in parallel. This guarantees the convergence of the algorithm, under conditions similar to those in single-agent Q-learning. Equivalently, if a transition model is available, value iteration can also be performed by each agent identically. In this way, the whole multiagent system is effectively treated as a 'big' single agent, and the learning algorithm is simply reproduced by each agent. The approach resembles the communication-free coordination protocols of chapter 4 where each agent runs the same algorithm identically and in parallel.

When the individual rewards differ and common knowledge assumptions cannot be guaranteed, the following scheme for exploratory action selection can be used. At time t and state s, agent i chooses at random k joint actions from the last m ($m > k$) observed joint actions $H = \{a^{t-m}, \ldots, a^{t-1}\}$ taken by the agents in state s. Then each agent i computes the relative frequency of each a_{-i}, i.e., how many times out of k each a_{-i} was played by the other agents at state s. This results in an empirical distribution $\mu_i(s, a_{-i})$ that reflects the *belief* of agent i over the joint action a_{-i} of all other agents at state s. Using such an empirical belief, agent i can now compute his *expected* payoff $U_i(s, a_i)$ for an action a_i at state s as

$$U_i(s, a_i) = \sum_{a_{-i}} \mu_i(s, a_{-i}) Q^{(i)}(s, (a_{-i}, a_i)), \tag{8.7}$$

and then randomly choose a best-response action from $\arg\max_{a_i} U_i(s, a_i)$.

Variations of the above approach have been used by Peyton Young (1993), Claus and Boutilier (1998), and Wang and Sandholm (2003). In particular, under some additional technical conditions, Q-learning with such an exploration scheme can be shown to converge to a coordinated joint policy for the agents (Wang and Sandholm, 2003). The approach bears resemblance to **fictitious play**, a learning method in games in which a number of (less than fully rational) agents interact repeatedly with each other aiming at reaching an equilibrium (Fudenberg and Levine, 1998).

8.5 Notes and further reading

Single-agent reinforcement learning is treated thoroughly in the book of Sutton and Barto (1998). Fudenberg and Levine (1998) provide a detailed treatment of learning in games. The article of Tan (1993) was one of the first attempts to extend RL to multiagent systems, where independent learning was applied on the predator-prey problem. Other references on independent Q-learning include (Sen et al., 1994; Matarić, 1994). The idea of decomposing the global Q-function into a sum of local functions has been proposed by Schneider et al. (1999) and Guestrin et al. (2002b). The papers of Claus and Boutilier (1998), Wolpert et al. (1999), and Wang and Sandholm (2003) discuss the case of cooperative multiagent RL and examine the conditions under which Q-learning converges. Multiagent RL among self-interested agents has been treated by Littman (1994) and Lagoudakis and Parr (2003). Learning in general-sum games has been studied by Hu and Wellman (1998), Littman (2001), and Bowling and Veloso (2002).

Bibliography

Aumann, R. J. (1976). Agreeing to disagree. *Ann. Statist.*, 4(6):1236–1239.

Bellman, R. (1961). *Adaptive Control Processes: a Guided Tour*. Princeton University Press.

Bernstein, D. S., Zilberstein, S., and Immerman, N. (2000). The complexity of decentralized control of Markov decision processes. In *Proc. 16th Int. Conf. on Uncertainty in Artificial Intelligence*, Stanford, CA.

Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *Proc. Conf. on Theoretical Aspects of Rationality and Knowledge*.

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

Castelpietra, C., Iocchi, L., Nardi, D., Piaggio, M., Scalzo, A., and Sgorbissa, A. (2000). Coordination among heterogenous robotic soccer players. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Takamatsu, Japan.

Clarke, E. H. (1971). Multipart pricing of public goods. *Public choice*, 11:17–33.

Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *Proc. 15th Nation. Conf. on Artificial Intelligence*, Madison, WI.

Cohen, P. R. and Levesque, H. J. (1991). Teamwork. *Nous*, 25(4):487–512.

Decker, K. and Lesser, V. R. (1995). Designing a family of coordination algorithms. In *Proc. 1st Int. Conf. on Multi-Agent Systems*, San Francisco, CA.

Durfee, E. H. and Lesser, V. R. (1987). Using partial global plans to coordinate distributed problem solvers. In *Proc. 10th Int. Joint Conf. on Artificial Intelligence*, Milan, Italy.

Fagin, R., Halpern, J., Moses, Y., and Vardi, M. (1995). *Reasoning about Knowledge*. The MIT Press, Cambridge, MA.

Ferber, J. (1999). *Multi-Agent Systems: an Introduction to Distributed Artificial Intelligence*. Addison-Wesley.

Fudenberg, D. and Levine, D. K. (1998). *The theory of learning in games*. MIT Press.

Geanakoplos, J. (1992). Common knowledge. *J. of Economic Perspectives*, 6(4):53–82.

Gibbard, A. (1973). Manipulation of voting schemes: a general result. *Econometrica*, 41:587–601.

Gibbons, R. (1992). *Game Theory for Applied Economists*. Princeton University Press.

Gilbert, N. and Doran, J., editors (1994). *Simulating Societies: the computer simulation of social phenomena*. UCL Press, London.

Gmytrasiewicz, P. J. and Durfee, E. H. (2001). Rational communication in multi-agent environments. *Autonomous Agents and Multi-Agent Systems*, 4:233–272.

Groves, T. (1973). Incentives in teams. *Econometrica*, 41:617–631.

Guestrin, C., Koller, D., and Parr, R. (2002a). Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems 14*. The MIT Press.

Guestrin, C., Lagoudakis, M., and Parr, R. (2002b). Coordinated reinforcement learning. In *Proc. 19th Int. Conf. on Machine Learning*, Sydney, Australia.

Harsanyi, J. C. and Selten, R. (1988). *A General Theory of Equilibrium Selection in Games*. MIT Press.

Hu, J. and Wellman, M. P. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proc. 15th Int. Conf. on Machine Learning*, San Francisco, CA.

Huhns, M. N., editor (1987). *Distributed Artificial Intelligence*. Pitman, Morgan Kaufmann.

Jennings, N. R. (1996). Coordination techniques for distributed artificial intelligence. In O'Hare, G. M. P. and Jennings, N. R., editors, *Foundations of Distributed Artificial Intelligence*, pages 187–210. John Wiley & Sons.

Kitano, H., Tambe, M., Stone, P., Veloso, M., Coradeschi, S., Osawa, E., Matsubara, H., Noda, I., and Asada, M. (1997). The RoboCup synthetic agent challenge 97. In *Proc. Int. Joint Conf. on Artificial Intelligence*, pages 24–29.

Kok, J. R., Spaan, M. T. J., and Vlassis, N. (2003). Multi-robot decision making using coordination graphs. In *Proc. 11th Int. Conf. on Advanced Robotics*, Coimbra, Portugal.

Labrou, Y., Finin, T., and Peng, Y. (1999). Agent communication languages: The current landscape. *IEEE Intelligent Systems*, 14(2):45–52.

Lagoudakis, M. G. and Parr, R. (2003). Learning in zero-sum team Markov games using factored value functions. In *Advances in Neural Information Processing Systems 15*, Cambridge, MA. MIT Press.

Lesser, V. R. and Erman, L. D. (1980). Distributed interpretation: a model and experiment. *IEEE Trans. Computers*, 29(12):1144–1163.

Lewis, D. K. (1969). *Conventions: A Philosophical Study*. Harvard University Press, Cambridge.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proc. 11th Int. Conf. on Machine Learning*, San Francisco, CA.

Littman, M. L. (2001). Friend-or-foe Q-learning in general-sum games. In *Proc. 18th Int. Conf. on Machine Learning*, San Francisco, CA.

Matarić, M. J. (1994). Reward functions for accelerated learning. In *Proc. 11th Int. Conf. on Machine Learning*, San Francisco, CA.

Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36:48–49.

Nisan, N. (1999). Algorithms for selfish agents. In *Proc. 16th Symp. on Theoret. Aspects of Computer Science*, Trier, Germany.

Noriega, P. and Sierra, C., editors (1999). *Agent Mediated Electronic Commerce*. Lecture Notes in Artificial Intelligence 1571. Springer.

O'Hare, G. M. P. and Jennings, N. R., editors (1996). *Foundations of Distributed Artificial Intelligence*. John Wiley & Sons.

Osborne, M. J. (2003). *An Introduction to Game Theory*. Oxford University Press.

Osborne, M. J. and Rubinstein, A. (1994). *A Course in Game Theory*. MIT Press.

Papadimitriou, C. H. (2001). Algorithms, games, and the Internet. In *Proc. 33rd Ann. ACM Symp. on Theory of Computing*, Crete, Greece.

Parkes, D. C. (2001). *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, Computer and Information Science, University of Pennsylvania.

Peyton Young, H. (1993). The evolution of conventions. *Econometrica*, 61(1):57–84.

Roumeliotis, S. I. and Bekey, G. A. (2002). Distributed multi-robot localization. *IEEE Trans. Robotics and Automation*, 18(5):781–795.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: a Modern Approach*. Prentice Hall, 2nd edition.

Sandholm, T. (1999). Distributed rational decision making. In Weiss, G., editor, *Multiagent Systems: A Modern Introduction to Distributed Artificial Intelligence*, pages 201–258. MIT Press.

Schneider, J., Wong, W.-K., Moore, A., and Riedmiller, M. (1999). Distributed value functions. In *Proc. Int. Conf. on Machine Learning*, Bled, Slovenia.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, UK.

Sen, S., Sekaran, M., and Hale, J. (1994). Learning to coordinate without sharing information. In *Proc. 12th Nation. Conf. on Artificial Intelligence*, Seattle, WA.

Shoham, Y. and Tennenholtz, M. (1992). On the synthesis of useful social laws for artificial agent societies. In *Proc. 10th Nation. Conf. on Artificial Intelligence*, San Diego, CA.

Singh, M. P. (1994). *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications*. Springer Verlag.

Smith, R. G. (1980). The contract net protocol: high-level communication and control in a distributed problem solver. *IEEE Trans. Computers*, C-29(12):1104–1113.

Stone, P. and Veloso, M. (2000). Multiagent systems: a survey from a machine learning perspective. *Autonomous Robots*, 8(3).

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Sycara, K. (1998). Multiagent systems. *AI Magazine*, 19(2):79–92.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proc. 10th Int. Conf. on Machine Learning*, Amherst, MA.

Tanenbaum, A. S. and van Steen, M. (2001). *Distributed Systems: Principles and Paradigms*. Prentice Hall.

Terzopoulos, D. (1999). Artificial life for computer graphics. *Commun. ACM*, 42(8):32–42.

Vickrey, W. (1961). Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37.

von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. John Wiley & Sons.

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 15*, Cambridge, MA. MIT Press.

Weiss, G., editor (1999). *Multiagent Systems: a Modern Approach to Distributed Artificial Intelligence*. MIT Press.

Wolpert, D., Wheeler, K., and Tumer, K. (1999). General principles of learning-based multi-agent systems. In *Proc. 3rd Int. Conf. on Automomous Agents*, Seattle, WA.

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. John Wiley & Sons.