Delft University of Technology

Delft Center for Systems and Control

Technical report 05-019

Learning and coordination in dynamic multiagent systems*

L. Buşoniu, B. De Schutter, and R. Babuška

October 2005

Delft Center for Systems and Control
Delft University of Technology
Mekelweg 2, 2628 CD Delft
The Netherlands
phone: +31-15-278.51.19 (secretary)
fax: +31-15-278.66.79
URL: http://www.dcsc.tudelft.nl

*This report can also be downloaded via http://pub.deschutter.info/abs/05_019.html

Delft University of Technology

Delft Center for Systems and Control

Interactive Collaborative Information Systems

Collaborative Decision Making

Technical report 05-019

Learning and Coordination in Dynamic Multiagent Systems

L. Buşoniu, B. De Schutter, and R. Babuška

October 2005

Delft Center for Systems and Control
Delft University of Technology
Mekelweg 2, 2628 CD Delft
The Netherlands
phone: +31-15-278.51.19 (secretary)
fax: +31-15-278.66.79
URL: `http://www.dcsc.tudelft.nl`

Learning and Coordination in Dynamic Multiagent Systems

L. Buşoniu, B. De Schutter, and R. Babuška

October 2005

Contents

List of Figures

List of Tables

List of Symbols

Common notations

$\cdot$	generic placeholder for (list of) argument(s)
$\boldsymbol{v}$	joint variable collected from several agents
v^{T}	vector transpose
$[v_1, v_2]$	vector concatenation
$\langle v_1, v_2 \rangle$	tuple
(v_1, v_2)	joint collection of strategies or policies
$\overline{f}$	deterministic variant of originally stochastic function
$\tilde{f}$	parameterized function
v'	next value
$\tilde{v}$	alternate variable
v^*	optimal value
$\widehat{v}$	estimated value
$P(\cdot)$	probability
$P(\cdot \mid \cdot)$	conditional probability
$\Pi(\cdot)$	probability distribution over argument set

Operators

$\|\cdot\|$	vector norm
$\|\cdot\|$	set cardinality
$E\{\cdot\}$	expectation
$E_f\{\cdot\}$	expectation conditioned on function
$E_f\{\cdot \mid \cdot\}$	expectation conditioned on function and initial condition
$\times\cdot$	product over sets
$\mathrm{d}\cdot$	discrete difference
$\mathrm{d}^n\cdot$	discrete n^{th} order difference

Dynamic agents and multiagent systems

k, l	discrete time index
A	set of agents

a	agent variable
i, j	agent index; agent with index i (j)
n	number of agents in the system
X	environment state space
x	environment state variable
f	environment dynamics
S	agent internal state space
s	agent internal state variable
$\mathcal{S}$	agent internal state space adaptation mapping
p	agent internal dynamics
θ	parameters vector for parameterized agent internal dynamics
Θ	parameter space for parameterized agent internal dynamics
$\mathcal{P}$	adaptation mapping for agent internal dynamics
Y	agent observation space
y	agent observation variable
ω	agent observation distribution
U	agent action space
u	agent action variable
h	agent policy
$\boldsymbol{U}$	joint action space
$\boldsymbol{u}$	joint action variable
$\boldsymbol{h}$	joint agent policy
ϕ	parameters vector for parameterized agent policy
Φ	parameter space for parameterized agent policy
$\mathcal{H}$	agent policy adaptation mapping

Communication

X^{c}	communication channel state space
X^{e}	state space of the environment excluding the communication channel
x^{c}	communication channel state variable
x^{e}	state variable for the environment excluding the communication channel
f^{c}	communication channel dynamics
x^{e}	dynamics of the environment excluding the communication channel
M	set of agent messages
m^{rcv}	message received by the agent
m^{snd}	message sent by the agent
$\boldsymbol{m}$	joint message sent by the agents

Y^{e}	agent's space of environment observations
y^{e}	agent's environment observation variable
ω^{e}	agent's environment observation distribution
ω^{rcv}	agent's message receival distribution
U^{e}	agent domain-level action space
u^{e}	agent domain-level action
h^{e}	agent domain-level policy
h^{snd}	agent message sending policy
$\boldsymbol{U}^{\mathrm{e}}$	joint domain-level action space
$\boldsymbol{u}^{\mathrm{e}}$	joint domain-level action
p^{e}	agent domain-level observations handling dynamics
p^{rcv}	agent messages handling dynamics

Roles

U^{r}	set of agent roles
U^{o}	space of ordinary agent actions
u^{r}	agent role
u^{o}	agent ordinary action
h^{r}	agent role choice policy
h^{o}	agent ordinary action choice policy
C	role constraints function

Reinforcement learning

ρ	reward function
r	reward variable
V	state value function
Q	action value function
E	eligibility trace
γ	discount factor
α	learning rate
λ	recency factor
ε	exploration probability
σ	agent strategy
$\boldsymbol{\sigma}$	joint strategy
$\boldsymbol{\sigma}_{-i}$	reduced strategy profile excluding agent i

List of Abbreviations

MAS	multiagent system
DMAS	dynamic multiagent system
AL-MAS	adaptive learning multiagent system
RL	reinforcement learning
MA-RL	multiagent reinforcement learning
MDP	Markov decision process
MMDP	multiagent Markov decision process
COM-MTDP	communicative Markov team decision problem
ICIS	Interactive Collaborative Information Systems
SARSA	state-action, reward-state-action
WoLF-PHC	Win-or-Learn-Fast policy hill-climbing
L-ALLIANCE	Learning ALLIANCE
STEAM	Shell for TEAMwork

Chapter 1

Introduction

1.1 Multiagent systems and adaptive learning

An *agent* is, informally, any entity that can perceive its environment through sensors and act upon it through actuators (Russell and Norvig, 2003). This definition is very broad. It includes robotic agents, that perceive their environment through cameras and distance sensors, and act upon it with motors and grippers. It includes process controllers, that use sensors to measure the outputs of the process, which in this case is the environment, and act upon it with command signals. It even includes humans, perceiving the world with their five senses and acting upon it using their motor and verbal skills.

A *multiagent system (MAS)* is, informally, a collection of agents that interact with each other (Vlassis, 2003). Examples of MAS involving the three types of agents mentioned above are, in order, teams of robots, distributed networks of controllers, and social groups of humans.

We formalize these, and the other notions that we use in this foreword, in Section 1.2. For now, we focus on motivating MAS and learning in MAS. Specifically, we are interested in *synthetic* MAS such as the robotic teams and controller networks exemplified above.

MAS are useful in the modeling, analysis and design of systems where control is distributed among several autonomous decision makers. MAS can arise naturally as the most viable solution of representing the considered problem. For instance, they are the most natural way of looking at distributed systems such as the robotic teams or controller networks above. A centralized perspective on such systems typically does not help, as it ignores the interactions between the autonomous decision makers, interactions that can either help or damage, perhaps up to the point of rendering it impossible, the activity of the system as a whole.

MAS can also provide an alternative perspective on systems that are originally represented in other ways. For instance, resource allocation can be seen as a centralized decision-making problem, where a single, central scheduler assigns resources to users. It can, however, also be seen as a set of autonomous, proactive users, that compete to gain access to the resources. The latter perspective can provide valuable insight into the problem and its possible solutions.

MAS offer the following potential advantages over centralized systems (Stone and Veloso, 2000):

- Speed-up of the system activity, due to parallel computation.
- Robustness and reliability, when the capabilities of the agents overlap. The system resists to failures in one or several agents, by having other agents take over the activity of the faulty ones.

- Scalability and flexibility. In principle, since MAS are inherently modular, adding and removing agents to the system should be easy. In this way, the system could adapt to a changing task on-the-fly, without ever needing to shutdown or be redesigned.

- Ease of design, development, and maintenance. This also follows from the inherent modularity of the MAS.

It should be noted at this point that MAS are not a panacea. The potential benefits described above should be carefully weighed with the simplicity of a centralized solution, considering the characteristics of the task. For instance, a task where parallel, distributed control is not possible is unlikely to benefit from a MAS solution (Stone and Veloso, 2000).

Learning is, informally, the acquisition and incorporation of knowledge and skills by an agent, leading to an improvement in the agent's performance. Learning is necessary in MAS because many times the environment of the MAS is large, complex, open and time-varying (Sen and Weiss, 1999). The first two properties imply that designing a good agent behaviour, that takes into consideration all the possible circumstances the agents might encounter, is a very difficult, if not impossible, undertaking. The second two properties, openness and variation over time, imply that even if such a behaviour were somehow designed, it would quickly become obsolete as the environment changes.

This behavioural improvement view is the focus of this survey. A complementary view, that pursues insight into multiagent learning as opposed to traditional, isolated machine learning, is also possible. Such insight could lead to novel machine learning techniques and algorithms (Sen and Weiss, 1999).

Why, then, is adaptation necessary on top of learning? The reason stems from the fact that, due to difficulties in dealing with open and time-varying environments, most multiagent learning algorithms are designed for unchanging environments. They typically involve some fixed learning structures that are updated by a set of rules involving some fixed or scheduled parameters. We call such algorithms "static" learning.

By allowing the learning parameters or structures of the static algorithms to adapt, the learning processes of the agents should be able to regain their ability of handling open and time-varying environments.

Note that adaptive learning is not a radically different process from learning. It can be viewed as a kind of "meta-learning" – that is, a special case of "learning how to learn".

This review is organized in the following way. The rest of this chapter is dedicated to formalizing the discussion above. First, we introduce a consistent framework focusing on the dynamic nature of the learning agents, and adding other important elements such as inter-agent communication. We then present a taxonomy of MAS, formalized using this framework. We close the chapter by presenting some examples of single-agent and multiagent control, and showing how they fit into our framework.

Chapter 2 deals with the application of reinforcement learning (RL) techniques to MAS. We introduce the single-agent RL framework and solutions, and continue with presenting their extension to the multiagent case. We analyze a number of adaptive RL techniques, and close the chapter with a detailed presentation of conclusions and research opportunities.

Chapter 3 briefly presents other methods of multiagent learning: evolutionary techniques, and heuristic approaches.

Chapter 4 deals with the issue of coordination in MAS, focusing on its relationship with learning. We review the most prominent coordination techniques encountered in the literature, focusing the discussion on learned variants of these techniques. We then outline some promising research opportunities in multiagent coordination.

Chapter 5 reviews the relevant application domains of MAS, from the perspective of the Interactive Collaborative Information Systems large-scale traffic incident scenario.

Chapter 6 concludes the review. It summarizes our main conclusions on the multiagent learning and coordination techniques analyzed in the preceding chapters, and distils the identified research opportunities into a concise set of research questions.

1.2 Theoretical framework

We begin by defining the concepts of dynamic agent and dynamic MAS. Our goal is to isolate the elements belonging to the agents' "mind" (such as reasoning and learning processes and the data they rely on), for the purpose of analyzing them. We also emphasize the dynamic nature of these elements. We make the definition general so as to encompass as many models as possible from the variety used by the learning and coordination algorithms addressed throughout the review.

The two concepts are tightly interrelated: the agents are embedded in the multiagent world, which in its turn is influenced by the decisions of the agents.

Definition 1.1 *A* dynamic agent *is a tuple* $\langle S, Y, U, p, h, s_0 \rangle$*, where:*

- S *is the internal state space of the agent.*
- Y *is the observation space of the agent.*
- U *is the action space available to the agent.*
- $p : S \times Y \to S$ *is the agent transition function, describing how the agent evolves as a result of its observations of the environment.*
- $h : S \times Y \times U \to [0, 1]$ *is the decision probability distribution of the agent, describing its behaviour.*
- s_0 *is the agent's initial state.*

Definition 1.2 *A* dynamic multiagent system (DMAS) *is a tuple* $\langle A, X, f, \{\omega_i\}_{i \in A}, x_0 \rangle$*, where:*

- A *is the set of dynamic agents,* $n = |A|$ *being their number.*
- X *is the environment state space.*
- $f : X \times \boldsymbol{U} \times X \to [0, 1]$*, with* $\boldsymbol{U} = \times_{i \in A} U_i$ *the joint action space, is the environment transition probability distribution, describing how the environment evolves as a result of the agents' actions.*
- $\omega_i : X \times Y_i \to [0, 1], i \in A$ *are the observation probability distributions, describing how the state of the environment is translated into agent observations.*
- x_0 *is the initial environment state.*

We denote an agent in the set A simply by its index i (hence the notation "$i \in A$"). We also indicate that an element belongs to agent i by subscript index i (e.g., S_i).

In our interpretation, the agent is in complete control of its state space S_i, dynamics p_i and behaviour h_i. These are part of the agent's "mind". The observation distribution, ω_i, is not. Strictly speaking, it is a part of the environment. The observation space Y_i and the set of available actions U_i are the agent's "interface" with the environment. Together with ω_i, they also can be considered, in a sense, a part of the agent's sensory and actuating "body", with which it is endowed and that it cannot change. Moreover, the observation function ω_i is not known by the agent; all the agents sees are the results of applying this function to the state of the environment. The observation functions, though part of the environment, are distinct for each agent; each agent sees the world "through its own eyes".

The environment state, observation, internal state and action spaces may be continuous or discrete, and in the latter case infinite or finite. The spectra of observations Y_i the agents may experience and the sets of actions U_i available to the agents may depend on the state of the environment. E.g., if a robotic agent moves along a very narrow straight underground tunnel, it cannot observe the position of the sun, nor can it move left or right. However, to prevent the notation from becoming cluttered, the definition above does not capture this dependence.

We work with discrete time, as virtually all learning and coordination algorithms work on a per iteration basis. We denote the current value of the discrete time variable by k. For readability, wherever the context clearly indicates that the exposition refers to the "current" time step, we may omit the time variable.

We denote the environment state value by x and agent i's action by u_i. The joint agent action is defined by $\boldsymbol{u} = [u_1, \ldots, u_n]^{\mathrm{T}}, \boldsymbol{u} \in \boldsymbol{U}$.

We denote the probability that agent i observes $y_{i,k} \in Y_i$ at time step k when the environment state is x_k by $P(y_{i,k} \,|\, x_k) = \omega_i(x_k, y_{i,k})$. The definition allows for incomplete and/or uncertain observations. When the observations are deterministic, the observation probability distribution changes into a function $\overline{\omega}_i : X \to Y_i$. If the agent is viewed as a controller of the environment, then its observation may also be interpreted as its *input*, or *control feedback*.

We denote the fact that agent i changes its internal state to $s_{i,k+1}$ as a result of observing $y_{i,k}$ in state $s_{i,k}$, by $s_{i,k+1} = p_i(s_{i,k}, y_{i,k})$.

We denote the probability of agent i taking action $u_{i,k}$ at step k, given its updated internal state $s_{i,k+1}$ and its observation $y_{i,k}$, by $P(u_{i,k} \,|\, s_{i,k+1}, y_{i,k}) = h_i(s_{i,k+1}, y_{i,k}, u_{i,k})$. We refer to h_i as the *policy* of the agent. The model thus assumes stochastic policies in the general case. When the policy is deterministic, it changes into a function $\overline{h}_i : S_i \times Y_i \to U_i$. If the agent is interpreted as a controller of the environment, then its action may also be interpreted as its *command output*.

At first sight, it appears strange that the value of the *updated* agent state $s_{i,k+1}$ is used in the action choice. The reason is that it is essential for the agent to take into account and adapt to the environment's state as soon as it can be observed. Because of this, the first thing that happens after the agent receives its observation is the *update of its internal state*: $s_{i,k+1} = p_i(s_{i,k}, y_{i,k})$. Hence, the notation $s_{i,k+1}$ is conventional and should be understood as an "updated" value rather than a "next" value that, intuitively, cannot exist yet.

We denote the probability of the environment changing its state to $x_{k+1} \in X$ as a result of the application of joint action $\boldsymbol{u}_k$ in state $x_k \in X$ at time step k, by $P(x_{k+1} \,|\, \boldsymbol{u}_k, x_k) = f(x_k, \boldsymbol{u}_k, x_{k+1})$. The definition allows for stochastic environment evolution. When the environment evolution is deterministic, the transition probability distribution changes into a function $\overline{f} : X \times \boldsymbol{U} \to X$.

With these notations, the description of the interaction between the agents and their environment is given in Algorithm 1.1. We use deterministic functions to keep the notation simple. We also use assignments instead of equal signs where statements are operational rather than declarative. All the system components have the same clock – i.e., evolve along the same discrete time axis.

Algorithm 1.1 DMAS evolution

1: $k \leftarrow 0$
2: **loop**
3: $\quad y_{i,k} \leftarrow \overline{\omega}_i(x_k), \quad \forall i \in A$ ▷ observe state
4: $\quad s_{i,k+1} \leftarrow p_i(s_{i,k}, y_{i,k}), \quad \forall i \in A$ ▷ evolve
5: $\quad u_{i,k} \leftarrow \overline{h}_i(s_{i,k+1}, y_{i,k}), \quad \forall i \in A$ ▷ take action
6: $\quad x_{k+1} \leftarrow \overline{f}(x_k, \boldsymbol{u}_k)$, where $\boldsymbol{u}_k = [u_{1,k}, \ldots, u_{n,k}]^{\mathrm{T}}$ ▷ environment evolves
7: $\quad k \leftarrow k + 1$
8: **end loop**

Figure 1.1 presents a schematic view over a dynamic multiagent system. The operator z^{-1} delays the discrete signal with one time step.

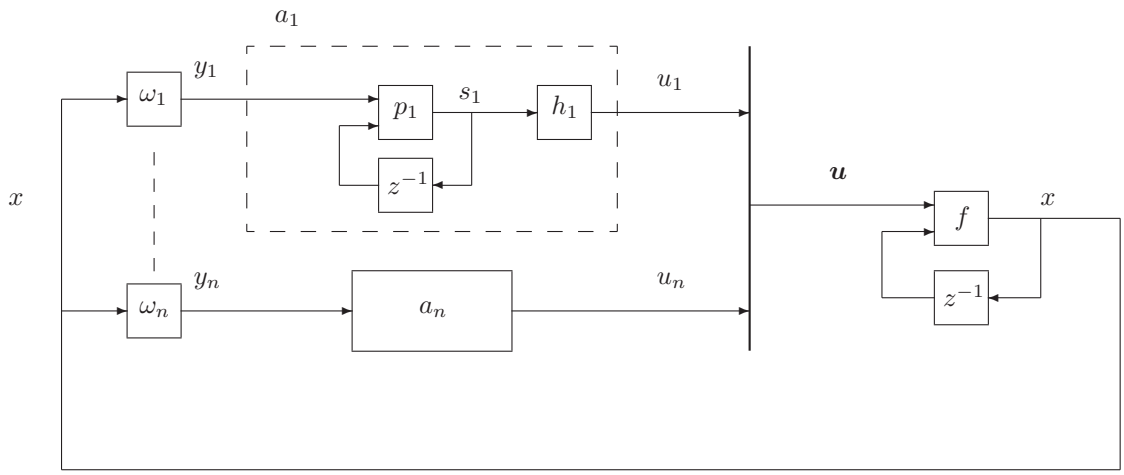


Figure 1.1: Schematic view of DMAS.

The evolution of the DMAS world can also be interpreted in the following way. The environment and agent states are components of a global state $\boldsymbol{x}_k = [x_k, s_{1,k}, \ldots, s_{n,k}]^{\mathrm{T}}$. The agents choose actions as their decision making functions dictate, $u_{i,k} = \overline{h}_i(s_{i,k+1}, y_{i,k})$. The world changes state as described by a global transition function, $\boldsymbol{x}_{k+1} = \overline{\boldsymbol{f}}(\boldsymbol{x}_k, \boldsymbol{u}_k)$, and then the cycle repeats. The observation function together with the environment and internal transition functions are then all incorporated into the global transition function $\overline{\boldsymbol{f}}$. This is a very general interpretation, but it does not provide the explicit separation between the agents' "mind" and their environment, that we are interested in.

We define the concepts of a rational agent and learning agent within the introduced framework.

Definition 1.3 *A* rational agent *is a dynamic agent that continuously strives to optimize a performance measure expressed in terms of its observations.*

This performance measure encodes, or represents, the goal of the agent. The performance measure is expressed in terms of observations because that is the only way the agent can access the environment's state. The goal of the agent can be (and most of the times will be) originally expressed in terms of the environment's state, but still the agent can only rely on its observations to estimate this state.

Definition 1.4 *A* learning agent *is a rational agent that makes use of its internal state s and dynamics p in an attempt to improve its ability of optimizing the aforementioned performance measure.*

The meaning of the internal state and dynamics of the agent in the context of learning is the following: the internal state s_i incorporates the agent's *learned knowledge*, and the dynamics p_i include the *learning processes* of the agent (i is the agent's index). These processes strive to improve the knowledge of the agent, and through it, its decision making abilities. Historical data can be stored into the agent's internal state via the agent dynamics. The internal state is also the structure where any prior knowledge should be stored. Among other things, the dynamics p_i can also perform predictions to help with action selection. The result of these predictions are saved onto the internal state and can be "read" by the agent policy.

Many models employed by multiagent learning algorithms do not explicitly account for the internal state of the agents, but instead consider the agent learning processes separate from the environment evolution processes and give them access to certain learning data structures. Our model integrates these learning structures into the generic concept of internal agent state and allows the agents and the environment to flow together in their evolution. Thus, the dynamic nature of a learning agent clearly stands out. An agent can, however, also be dynamic without necessarily learning.

Definition 1.5 *A* learning multiagent system *is a* DMAS *within which at least one of the agents is a learning agent.*

By allowing the agent to alter the elements it has control over – its internal state space, transition function, and policy – we obtain an adaptive agent.

Definition 1.6 *An* adaptive agent *is a tuple $\langle S_0, Y, U, p_0, \omega, h_0, s_0, \mathcal{S}, \mathcal{P}, \mathcal{H} \rangle$, where:*

- *Y, U, ω have the same meaning as in Definition 1.1.*
- *$\mathcal{S}$ is the internal state adaptation mapping of the agent. This mapping specifies how the internal state space of the agent adapts at each step as a result of the agent's internal state and observation: $S_{k+1} = \mathcal{S}(S_k, s_k, y_k)$. Thus:*
- *$S_k, k \geq 0$ is the adapting internal state space of the agent, with S_0 being the initial state space.*
- *$\mathcal{P}$ is the transition adaptation mapping of the agent. This mapping specifies how the transition function of the agent adapts at each step as a result of the agent's internal state and observation: $p_{k+1} = \mathcal{P}(p_k, s_k, y_k)$. Thus:*
- *$p_{k+1} : S_k \times Y \to S_{k+1}, k \geq 0$ is the adapting transition function of the agent, with $p_0 : S_0 \times Y \to S_0$ being the initial transition function.*

- $\mathcal{H}$ *is the policy adaptation mapping of the agent. This mapping specifies how the decision distribution probability of the agent adapts at each step as a result of the agent's internal state:* $h_{k+1} = \mathcal{H}(h_k, s_k, y_k)$. *Thus:*
- $h_k : S_k \times Y \times U \to [0, 1], k \geq 0$ *is the adapting decision probability distribution of the agent, with* h_0 *being the initial distribution.*

An adaptive agent is an enhancement of a dynamic agent. An adaptive agent that is also learning is able at each time step not only to use its dynamics to improve its knowledge, but also to alter the structure of this knowledge and the way its learning algorithm and policy work. We call such an agent an *adaptive learning agent.*

Definition 1.7 *An* adaptive learning multiagent system (AL-MAS) *is a DMAS within which at least one of the agents is an adaptive learning agent.*

Algorithm 1.2 describes the interaction between the agents and their environment in an AL-MAS. We use deterministic functions and assume all agents are adaptive to keep the notation simple. Prior to doing anything else, the agent adapts and then learns, hence the presence of the incremented time indices where adaptive elements are used. As explained before, these incremented indices should be interpreted as "updated" rather than "next" elements.

Algorithm 1.2 AL-MAS evolution

1: $k \leftarrow 0$
2: **loop**
3: $y_{i,k} \leftarrow \overline{\omega}_i(x_k), \quad \forall i \in A$ ▷ observe state
4: $S_{i,k+1} \leftarrow \mathcal{S}_i(S_{i,k}, s_{i,k}, y_{i,k}), \quad \forall i \in A$
5: $p_{i,k+1} \leftarrow \mathcal{P}_i(p_{i,k}, s_{i,k}, y_{i,k}), \quad \forall i \in A$ ▷ adapt
6: $\overline{h}_{i,k+1} \leftarrow \mathcal{H}_i(\overline{h}_{i,k}, s_{i,k}, y_{i,k}), \quad \forall i \in A$
7: $s_{i,k+1} \leftarrow p_{i,k+1}(s_{i,k}, y_{i,k}), \quad \forall i \in A$ ▷ learn
8: $u_{i,k} \leftarrow \overline{h}_{i,k+1}(s_{i,k+1}, y_{i,k}), \quad \forall i \in A$ ▷ decide
9: $x_{k+1} \leftarrow \overline{f}(x_k, \boldsymbol{u}_k)$, where $\boldsymbol{u}_k = [u_{1,k}, \ldots, u_{n,k}]^{\mathrm{T}}$ ▷ environment evolves
10: $k \leftarrow k + 1$
11: **end loop**

Note that we do not require all the agents in an (adaptive) learning DMAS to be (adaptive) learning agents. This allows for greater flexibility. We also do not forbid adaptation in non-learning agents.

One of the most important reasons for adaptation in an agent is keeping pace with a time-varying environment: $f = f_k$ and / or $\omega_i = \omega_{i,k}$. This happens when the DMAS is not closed, but interacts with the outside world: e.g., objects might be inserted or removed in a spatial domain, or temperature drifts might affect parameters in the process controlled by the agents.

We can distinguish two special, very interesting cases of adaptation. The first occurs when the agent adapts some parameters of its learning and / or policy function; the second, when it adapts its state space.

Definition 1.8 Parametric adaptation *is the process of adaptation occurring within an adaptive learning agent that:*

1. *uses parametric representations for its learning and policy functions:* $p_k(\cdot) = \tilde{p}(\cdot, \theta_k), \tilde{p} : S \times Y \times \Theta \to S$, $h_k(\cdot) = \tilde{h}(\cdot, \phi_k), \tilde{h} : S \times Y \times U \times \Phi \to [0, 1]$, $k \geq 0$, *where* $\theta_k \in \Theta$ *and* $\phi_k \in \Phi$ *are parameter vectors with some given initial values* θ_0 *and* ϕ_0.

2. *adapts the parameter vectors instead of the functions themselves:* $\theta_{k+1} = \tilde{\mathcal{P}}(\theta_k, s_k, y_k)$, $\phi_{k+1} = \tilde{\mathcal{H}}(\phi_k, s_k, y_k)$.

Definition 1.9 Structural adaptation *is the process of adaptation occurring within an adaptive learning agent that makes use of the state space adaptation function* $\mathcal{S}$ *to adapt its internal state space, and of the transition and policy adaptation mappings* $\mathcal{P}$ *and* $\mathcal{H}$ *to allow its transition and policy functions to accommodate the alterations in the structure of the internal state space.*

These definitions do not limit the adaptation behaviour of the agent. For instance, an agent using parametric adaptation may also use structural adaptation, and viceversa. The most restrictive type of structural adaptation is where $\mathcal{P}$ and $\mathcal{H}$ are used *only* to accommodate the alterations caused by $\mathcal{S}$, such as changes in dimensionality and constraints of the state space. The most restrictive type of parametric adaptation is where adaptation of the state space is forbidden. A broader type of adaptation of which parametric adaptation is a special case can be imagined, where the learning and policy functions are directly adapted but the state space may or may not be adapted.

It is important to note that adaptive learning can always be seen as a static learning process over an augmented static agent state space $\tilde{S}$. E.g., for parametric adaptation, this augmented state space incorporates the parameter spaces Θ, Φ, and the augmented state variable is $\tilde{s}_k = [s_k, \theta_k, \phi_k]^{\mathrm{T}}$. For structural adaptation, the augmented state space incorporates the structural parameters of the adapting state space S_k, such as number of dimensions, component ranges, discretization grain.

1.2.1 Communication

We introduce a special case of DMAS to deal with communication. Throughout the following, when stating "environment" we mean "environment excluding the communication channel".

Definition 1.10 *A* communicative multiagent system without interference *is a* DMAS $\langle A, X, f, x_0 \rangle$ *within which:*

(i) $X = X^{\mathrm{e}} \times X^{\mathrm{c}}$, *where* X^{e} *is the state space of the environment, and* X^{c} *is the state space of the communication channel. Thus,* $\forall x \in X, x = [x^{\mathrm{e}}, x^{\mathrm{c}}]^{\mathrm{T}}$, *with* $x^{\mathrm{e}} \in X^{\mathrm{e}}, x^{\mathrm{c}} \in X^{\mathrm{c}}$.

(ii) $\forall i \in A, Y_i = Y_i^{\mathrm{e}} \times M_i$, *where* Y_i^{e} *is the space of environment observations, and* M_i *is the message space of agent* i. *Thus,* $\forall y_i \in Y_i, y_i = [y_i^{\mathrm{e}}, m_i^{\mathrm{rcv}}]^{\mathrm{T}}$, *with* $y_i \in Y_i, m_i^{\mathrm{rcv}} \in M_i$. *The variable* m_i^{rcv} *is called the* received message.

(iii) $\forall i \in A$, *there exist two distributions* $\omega_i^{\mathrm{e}} : x^{\mathrm{e}} \times Y_i^{\mathrm{e}} \to [0, 1]$ *and* $\omega_i^{\mathrm{rcv}} : X^{\mathrm{c}} \times M_i \to [0, 1]$, *such that* $\forall x = [x^{\mathrm{e}}, x^{\mathrm{c}}]^{\mathrm{T}} \in X, y_i = [y_i^{\mathrm{e}}, m_i^{\mathrm{rcv}}]^{\mathrm{T}} \in Y_i$, $P(y_i \mid x) = \omega_i(x, y_i)$ *iff* $P(y_i^{\mathrm{e}} \mid x^{\mathrm{e}}) = \omega_i^{\mathrm{e}}(x^{\mathrm{e}}, y_i^{\mathrm{e}})$ *and* $P(m_i^{\mathrm{rcv}} \mid x^{\mathrm{c}}) = \omega_i^{\mathrm{rcv}}(x^{\mathrm{c}}, m_i^{\mathrm{rcv}})$. *The distribution* ω_i^{e} *is the environment observation distribution, whereas* ω_i^{rcv} *is the* message receival distribution.

(iv) $\forall i \in A, U_i = U_i^{\mathrm{e}} \times M_i$, *where* U_i^{e} *is the space of domain-level (environment) actions, and* M_i *is the message space of agent* i. *Thus,* $\forall u_i \in U_i, u_i = [u_i^{\mathrm{e}}, m_i^{\mathrm{snd}}]^{\mathrm{T}}$, *with* $u_i \in U_i, m_i^{\mathrm{snd}} \in M_i$. *The variable* m_i^{snd} *is called the* sent message.

(v) $\forall i \in A$, *there exist a distribution* $h_i^{\mathrm{e}} : S_i \times U_i^{\mathrm{e}} \to [0,1]$ *and a function* $h_i^{\mathrm{snd}} : S_i \to M_i$, *such that* $\forall s \in S, u_i = [u_i^{\mathrm{e}}, m_i^{\mathrm{snd}}]^{\mathrm{T}} \in U_i$, $P(u_i \mid s_i) = h_i(s_i, u_i)$ *iff* $P(u_i^{\mathrm{e}} \mid s_i) = h_i^{\mathrm{e}}(s_i, u_i^{\mathrm{e}})$ *and* $m_i^{\mathrm{snd}} = h_i^{\mathrm{snd}}(s_i)$. *The distribution* h_i^{e} *is the domain-level (environment) policy, whereas the function* h_i^{snd} *is the* message sending policy.

(vi) Defining $\boldsymbol{U}^{\mathrm{e}} = \times_{i \in A} U_i^{\mathrm{e}}$, $\boldsymbol{u}^{\mathrm{e}} = [u_1^{\mathrm{e}}, \ldots, u_n^{\mathrm{e}}]^{\mathrm{T}}$, $\boldsymbol{M} = \times_{i \in A} M_i$, $\boldsymbol{m} = [m_1^{\mathrm{snd}}, \ldots, m_n^{\mathrm{snd}}]^{\mathrm{T}}$, *there exist two distributions* $f^{\mathrm{e}} : X^{\mathrm{e}} \times \boldsymbol{U}^{\mathrm{e}} \times X^{\mathrm{e}} \to [0,1]$ *and* $f^{\mathrm{c}} : X^{\mathrm{c}} \times \boldsymbol{M} \times X^{\mathrm{c}} \to [0,1]$, *such that* $\forall x = [x^{\mathrm{e}}, x^{\mathrm{c}}]^{\mathrm{T}} \in X, x' = [x^{\mathrm{e}\prime}, x^{\mathrm{c}\prime}]^{\mathrm{T}} \in X, \boldsymbol{u} = [u_1^{\mathrm{e}}, m_1, \ldots, u_n^{\mathrm{e}}, m_n]^{\mathrm{T}} \in \boldsymbol{U}$, $P(x' \mid x, \boldsymbol{u}) = f(x, \boldsymbol{u}, x')$ *iff* $P(x^{\mathrm{e}\prime} \mid x^{\mathrm{e}}, \boldsymbol{u}^{\mathrm{e}}) = f^{\mathrm{e}}(x^{\mathrm{e}}, \boldsymbol{u}^{\mathrm{e}}, x^{\mathrm{e}\prime})$ *and* $P(x^{\mathrm{c}\prime} \mid x^{\mathrm{c}}, \boldsymbol{m}) = f^{\mathrm{c}}(x^{\mathrm{c}}, \boldsymbol{m}, x^{\mathrm{c}\prime})$. *The distribution* f^{e} *is the environment dynamics, whereas* f^{c} *is the communication channel* transmission dynamics.

So, we assume an explicit communication channel X^{c} embedded in the world (i), whose state is "read" by the message receival distribution ω_i^{rcv} at each time step to produce a message (iii). This message becomes a distinct part of the agent's observation, usable by the agent's internal processes p_i (ii). Similarly, at each time step the agent is given the opportunity to create and send a message, besides the usual action applied to the environment (v). The message is a distinct part of the agent's output, and the set of messages that can be sent and received are identical (iv).

The definition ensures that there are no interferences between the communication channel and the environment: neither during sending and transmission, as the evolutions of the environment and of the communication channel are independent (vi); nor during receival, as the environment observation distribution and the message receival distribution are independent (iii). Hence, the characterization of the communication as "without interference". By removing these requirements, we allow the environment to interfere with the communication, but the analysis becomes more difficult and the distinction between the communicative and general DMAS becomes blurred.

Note however that we allow both the message transmission and receival to be noisy in the general case, since both f^{c} and ω_i^{rcv} are stochastic.

The complex Definition 1.10, intended to clearly expose the way communication emerges within the natural DMAS framework, can be taken further. By taking advantage of the assumption of no interference and separating the agents' dynamics in two parts, one that handles messages, $p_i^{\mathrm{rcv}} : S_i \times M_i \to S_i$, and another handling observations, $p_i^{\mathrm{e}} : S_i \times Y_i^{\mathrm{e}} \to S_i$, we can distinguish an independent communication process running in the DMAS, as described by Algorithm 1.3. The algorithm assumes noise-free communication (i.e., deterministic receival functions and channel dynamics).

Algorithm 1.3 Communication in an interference-free communicative DMAS

loop
  $m_i^{\mathrm{rcv}} \leftarrow \overline{\omega}_i^{\mathrm{rcv}}(x^{\mathrm{c}}) \quad \forall i \in A$ ▷ read channel for input message
  $s_i \leftarrow p_i^{\mathrm{rcv}}(s_i, m_i^{\mathrm{rcv}}), \quad \forall i \in A$ ▷ process input message
  $m_i^{\mathrm{snd}} \leftarrow h_i^{\mathrm{snd}}(s_i), \quad \forall i \in A$ ▷ create and send output message
  $x^{\mathrm{c}} \leftarrow \overline{f}^{\mathrm{c}}(x^{\mathrm{c}}, \boldsymbol{m})$, where $\boldsymbol{m} = [m_1^{\mathrm{snd}}, \ldots, m_n^{\mathrm{snd}}]^{\mathrm{T}}$ ▷ channel processes messages
end loop

An important observation is that communication is not instantaneous. There is a delay of at least one time step between the placing of a message onto the communication channel and

its receival by the agents. This is because in our model, all interactions must pass through the environment. However, if the processes in Algorithm 1.1 and Algorithm 1.3 take place simultaneously with different clocks, then the communication process can be made faster than the environmental interaction process, and in the limiting case communication can be considered instantaneous.

A agent using the described communication mechanism can also be adaptive. The extension is intuitively straightforward, and involves endowing the agent with the possibility to alter its communication policy. Because it is part of the observation space, however, the available set of messages must remain constant.

1.2.2 Agent roles

Informally, a role is a restriction imposed on the space of available domain-level actions for the agent taking that role. The explicit representation of roles leads to a special case of DMAS. We extend from the definition of a plain DMAS, but we do not introduce conflicts between communication and roles, so that a DMAS can have either of the two mechanisms or both of them. This definition was inspired by Jung et al. (2002).

Definition 1.11 *A* multiagent system with roles *is a tuple* $\langle A, X, f, U^{\mathrm{r}}, x_0 \rangle$ *where* $\langle A, X, f, x_0 \rangle$ *is a DMAS and:*

- U^{r} *is a set of available roles.*
- *The action space of each agent* $i \in A$ *is of the form* $U_i = U_i^{\mathrm{r}} \cup U_i^{\mathrm{o}}$*, with* $U_i^{\mathrm{r}} \in U^{\mathrm{r}}$*. Here,* U_i^{r} *are the roles available to agent* i*, and* U_i^{o} *is the space of ordinary actions available to agent* i*.*
- *The policy of each agent* $i \in A$ *can be decomposed in two parts:*
  - *a* role choice *policy* $h_i^{\mathrm{r}} : S_i \times Y_i \times U_i^{\mathrm{r}} \to [0, 1]$*, indicating how the agent chooses roles given its internal state and observation.*
  - *the time-varying* ordinary action choice *policy* $h_{i,k}^{\mathrm{o}} : S_i \times Y_i \times C_i(u_{i,k}^{\mathrm{r}}) \to [0, 1]$*, with* $C_i : U_i^{\mathrm{r}} \to U_i^{\mathrm{o}}$ *the role constraints function and* $u_{i,k}^{\mathrm{r}}$ *the role chosen at step* k *by* h_i^{r} *given* $s_{i,k}$ *and* $y_{i,k}$*.*

In the interpretation of this definition, roles are something the agents choose to employ, rather than something that is imposed to them. Rational agents will therefore choose to use roles because it benefits their performance. At each time step, the agent decides whether to maintain its current role or change it, and then decides on an ordinary action from the set indicated by the role via the constraint function C_i.

Roles will typically be used in cooperative multiagent systems. The set of roles is a characteristic of the multiagent system, but each agent uses its own subset U_i^{r} of this set. The interpretation of the roles (the constraints imposed on the action space) is also particular to each agent. Typically, however, agents will have some information on the roles used by other agents, and perhaps even on the constraints imposed by the roles on these agents' actions.

Since it has full control over its role choice policy and its role constraints, an adaptive agent can, in principle, adapt both these elements. However, due to the fact that other agents may have information on these elements and may rely on that information in making decisions, adaptation in the role choice and constraints area should be done cautiously.

1.2.3 The credit-assignment problem

The credit-assignment problem is the problem of determining the DMAS activities that were responsible for a detected change in the performance of an agent. This problem has two aspects: structural, and temporal credit assignment.

The *structural* credit-assignment problem is the problem of an agent determining the components of the DMAS that were responsible for a detected change in its performance. Structural credit assignment is performed at two levels: inter-agent, when the agent tries to determine the size of its own contribution to the performance change relative to the contribution of the others; and intra-agent, when the agent subsequently distributes this partial contribution among its internal components responsible with decision-making (Sen and Weiss, 1999).

The *temporal* credit-assignment problem arises because the DMAS environment is dynamic, and sometimes (in fact, most of the times) the effects of an action prolong far into the future. E.g., if a robotic agent goes right at a junction, passing into a straight corridor that leads it to its destination without any other turns, the robot has to somehow determine that the decision of going right earlier at the junction enabled it to reach its destination. Formally, then, the temporal credit-assignment problem is the problem of determining which of the past actions of an agent were responsible for a detected change in its performance.

1.3 Taxonomy of multiagent systems

Using the framework introduced in Section 1.2, we describe here several taxonomy dimensions for multiagent systems. We select the dimensions on the importance of which researchers agree, and that bear relevance in this review. Wherever the taxonomy categories impose restrictions with respect to the adaptivity of the agents, these restrictions are explicitly mentioned.

1.3.1 Agent-related dimensions

Degree of heterogeneity

A *homogeneous* dynamic multiagent system is a DMAS $\langle A, X, f, x_0 \rangle$ where $\forall i, j \in A$, $S_i = S_j$, $Y_i = Y_j$, $U_i = U_j$, $p_i \equiv p_j$, and $h_i \equiv h_j$.

Note that we do not require the observation distributions nor the initial states of the agents to be identical. The former allows for local agent perspectives and the latter for different initial knowledge. Thus, though homogeneous, the agents will not be identical and will gain different knowledge during their lifetime.

A *heterogeneous* dynamic multiagent system is a DMAS for which at least one of the above set of identities is not satisfied for at least one pair $i, j \in A$.

Both homogeneous and heterogeneous agents can be adaptive. An additional requirement for homogeneous agents is that they should use the same internal state, transition and policy adaptation mappings: $\mathcal{S}_i \equiv \mathcal{S}_j, \mathcal{P}_i \equiv \mathcal{P}_j, \mathcal{H}_i \equiv \mathcal{H}_j, \forall i, j \in A$.

Compatibility of agent goals

A *common goal* MAS is one within which the agents strive to maximize a common performance measure. Such a MAS is typically referred to as a multiagent *team*.

At the other end of the spectrum, the goal of any agent is completely opposite to the goals of the other agents. This is a *fully competitive* setting.

Intermediate situations exist, where the goals of the agents, without being completely opposite, sometimes come into conflict. Many times, even a common goal may lead the agents to partially conflicting secondary goals on the way to the solution (for example, taking possession of a shared resource).

Inter-agent cooperation

A *cooperative* multiagent system is one within which the agents are willing to help other agents obtain their goals. *Self-interested* agents, on the other hand, act only towards achieving their own goals.

This dimension is strongly related to goals compatibility. Multiagent teams will always be cooperative, whereas fully competitive agents will always be self-interested. In the grey area in between, cooperation is optional.

An agent helping other agents achieving their goals, whatever these goals are, may cost the agent temporary losses in its own performance, or it may not. On the long run, cooperation must be beneficial, or a rational agent does not have the incentive to cooperate.

The simple situation where the agents share a common goal and their temporary performance is never, at any time step, affected negatively by cooperation, is called *fully cooperative*.

Degree of control decentralization

When the actions of the all the agents are dictated by a single, central authority, we say that the control is *centralized*. In this case, the multiagent system degenerates into a single agent. Most of the control schemes in control engineering are centralized (see Section 1.4).

At the opposite end of the spectrum, the agents are the only arbiters of their own actions. This is *decentralized* (or "distributed") control. Decentralized control does not mean that an agent cannot coordinate its actions with other agents; rather, it means that the agent is free to choose whether and how to coordinate.

In between sits *hierarchical* control. In a hierarchical control scheme, agents retain part of their autonomy. However, agents on higher levels in the hierarchy impose goals onto the agents residing on lower levels. These lower-level agents are then bound to try realizing those goals. Higher-level agents may also judge the quality of the actions of their subordinates and offer them feedback in this respect. It is not necessary that there is a single top-level ("root") agent; several may exist.

We refer to hierarchical and decentralized control by the generic term of "distributed" control.

Reactivity versus deliberation

A *reactive agent* is a tuple $\langle Y, U, \omega, h \rangle$, where the policy is $h : Y \times U \to [0, 1]$ and the other elements have the same meaning as in Definition 1.1.

The above says that a reactive agent does not have an internal state, and takes actions only on the basis of its observation.

For an agent to be deliberative, the literature typically requires that it uses its internal dynamics to search through the space of available actions, predict their effects, and finally choose an action on the basis of these predictions. This requirement is, however, not strictly necessary for attaining a good performance, even for a learning agent. For instance, Q-learning agents act in a reflex fashion on the basis of their Q-function (see Section 2.2), but update

this Q-function on the basis of feedback from the environment. Hence, we could argue that since the Q-function encodes the experience of the agent – in effect, predictions on the results of its actions – the Q-learning agent is, in a sense, deliberative.

Wee see then that a clear definition of a deliberative agent is difficult to give. Consequently, the distinction between reactive and deliberative agents is not clear-cut. An intermediate example could be a deliberative agent that uses reflexes to perform certain simple tasks.

1.3.2 Environment-related dimensions

Degree of communication

The communicative multiagent system has already been defined in Section 1.2, Definition 1.10.

We say a DMAS $\langle A, X, f, x_0 \rangle$ has no interference-free communication if there exist no sets X^{c}, $M_i, i \in A$, distributions $f^{\mathrm{c}}, \omega_i^{\mathrm{rcv}}, i \in A$, and functions $h_i^{\mathrm{snd}}, i \in A$ such that the decomposition in Definition 1.10 is possible.

By dropping the requirement of no interference, as explained in Section 1.2, the distinction between communicative and non-communicative multiagent systems blurs. In this case, we can look at the interaction between the communication channel dynamics f^{c} and environment dynamics f^{e} in two ways. First, we can see it as an interference from the environment affecting the channel. By taking another point of view that does not perform the separation between the communication channel and the environment, we can look at communication as a process of transmitting information indirectly by affecting the environment. This is the concept of *stigmergy*. An example of stigmergy, for multiple robots performing a cleaning task, is one of the robots passing over an area that has already been cleaned. The robot can infer that another robot has passed through that area earlier, without any explicit communication taking place.

Both communicating and non-communicating agents can be adaptive. The special characteristics of the adaptation process for a communicating agent were described in Section 1.2.

Scope of the agent perspective

The agents in a DMAS $\langle A, X, f, x_0 \rangle$ have a *common perspective* if $\forall i, j \in A, \omega_i \equiv \omega_j \equiv \omega$, where ω is a common observation distribution. This immediately implies the equality of the agents' observation spaces: $Y_i = Y_j = Y$, where Y is a common observation space.

A *global perspective* offers the agents the same quantity of information about the environment. This doesn't mean that the environment state must be translated into observations identically for all the agents. This last statement only holds for the global perspective when the agents are homogeneous. In this case, the notions of global and common perspective coincide.

In general, the common perspective is a special case of global perspective.

If the quantities of information offered to the agents differ, we say that the agents have a *local perspective*.

Degree of measurability

The DMAS environment is *completely measurable* (or, simply, measurable) if $\forall x \in X$, and for any $i \in A$, the corresponding observation y_i can uniquely identify the state, i.e., $\forall \tilde{x} \in X, \tilde{x} \neq x, P(\tilde{x} \mid y_i) = 0$.

When the above is not satisfied, the environment is *partially measurable*.

These definitions are inspired by Pynadath and Tambe (2002). The researchers identified another interesting level of measurability[1]. When the observations of one agent are not enough to uniquely identify the environment state, but the collective observation of all the agents is, the environment is *collectively completely measurable* (or collectively measurable).

Formally, the DMAS environment is collectively measurable if $\forall x \in X$, the corresponding observations $y_1, \ldots, y_n$ uniquely identify the state, i.e., $\forall \tilde{x} \in X, \tilde{x} \neq x, P(\tilde{x} \mid y_1, \ldots, y_n) = 0$. This last level of measurability is useful only in the presence of communication.

We must stress here that the virtually all the MAS literature identifies measurability with observability, and speaks of the property calling it observability. This is not entirely accurate, as we shall see below; measurability is a special case of a broader notion of observability. We treated it separately because it carries weight in the literature.

Degree of observability

The DMAS environment is *completely observable* if for any possible sequence of environment states, the current state can be determined in finite time from the observations.

Formally, the DMAS environment is completely observable (or, simply, observable), if for all possible sequences of states $x_1, x_2, \ldots, x_k, \ldots$, for any k, and for any agent $i \in A$, there exists a finite $l_i \geq 0$ such that x_k can be uniquely determined from the sequence $y_{i,k}, \ldots, y_{i,k+l_i}$, i.e., $\forall \tilde{x} \neq x_k, P(\tilde{x} \mid y_{i,k}, \ldots, y_{i,k+l_i}) = 0$.

When the above is not satisfied, the environment is *partially observable*.

Similarly to the above, when the observations of one agent are not enough to uniquely identify the environment state in finite time, but the collective observation of all the agents is, the environment is *collectively completely observable* (or collectively observable).

These definitions are inspired from the control-engineering notion of observability.

The observability dimension is not orthogonal to the measurability dimension, being instead its generalization for sequences of observations: if all l_i are zero, observability reduces to measurability. For instance, if the environment is measurable, then by necessity it is observable. However, it can be observable without being measurable. Similarly, if the environment is collectively measurable, it is also collectively observable, without the converse being necessarily true.

Episodic vs. continuing tasks

The task of the MAS is *episodic*, if the interaction between the agents and the environment breaks naturally into subsequences, each terminating in one of a set of terminal states, and at a final time step. Each of these subsequences is an "episode" or "trial", and after each episode termination, the MAS is "reset" to an initial state or to a state drawn from a distribution of initial states.

When the interaction between the agents and the environment does not break naturally into identifiable episodes, but goes on indefinitely, the MAS task is *continuing*. These definitions are borrowed from Sutton and Barto (1998).

[1] In fact, they also identified a fourth level. When the system receives no feedback at all from the environment (in control terminology, the control is open-loop), then the environment is *not measurable*. This level is however of little use in our setting, so we do not refer to it further.

An example of episodic task is navigating through a maze: when all the agents exit the maze, the task ends. Many times, the goal of a MAS in an episodic task is to reach any of a subset of desirable terminal states in minimum time.

An example of continuing task is process control.

Along the presented taxonomy dimensions, we are interested in:

1. both homogeneous and heterogeneous agents;
2. agent teams, that are
3. cooperative (though not necessarily fully cooperative);
4. distributed agent control;
5. both reactive and deliberative agents;
6. all levels of inter-agent communication;
7. both global and local perspectives, especially the latter;
8. both complete and partial measurability, especially the latter;
9. both complete and partial observability;
10. both episodic and continuing tasks.

1.4 Examples

We present in this section several frameworks from control engineering and the multiagent systems literature, analyzing how they fit into the DMAS – AL-MAS models introduced in Section 1.2. In addition to illustrating the generality of our framework, the analysis will also provide helpful insight into its characteristics.

1.4.1 The conventional control scheme

We begin with the simplest feedback control scheme from control engineering: a linear, time invariant single-input, single-output plant controlled by a linear, time invariant controller (Figure 1.2). The evolution of the plant can be described by the discrete state space model:

$$\begin{cases} x_{k+1} = \Phi_p x_k + \Gamma_p u_k \\ y_k = C_p x_k \end{cases} \tag{1.1}$$

where $\Phi_p \in \mathbb{R}^{n_p \times n_p}$ is the discrete transfer matrix, $\Gamma_p \in \mathbb{R}^{n_p \times 1}$ the discrete input matrix, $C_p \in \mathbb{R}^{1 \times n_p}$ the discrete output matrix, with n_p the size of the plant's state space. The signal x is the plant's state, u is the command input, and y is the plant's output.

The controller is given by:

$$\begin{cases} s_{k+1} = \Phi_c s_k + \Gamma_c y_k \\ u_k = C_c s_k + D_c y^{\text{ref}} \end{cases} \tag{1.2}$$

where $\Phi_c \in \mathbb{R}^{n_c \times n_c}$, $\Gamma_c \in \mathbb{R}^{n_c \times 1}$ and $C_c \in \mathbb{R}^{1 \times n_c}$ have similar meanings, this time for the controller, D_c is the discrete direct feedthrough matrix, s is the controller's state, and y^{ref} is the reference signal. The basic control goal is to make the plant's output follow the reference signal as closely as possible.

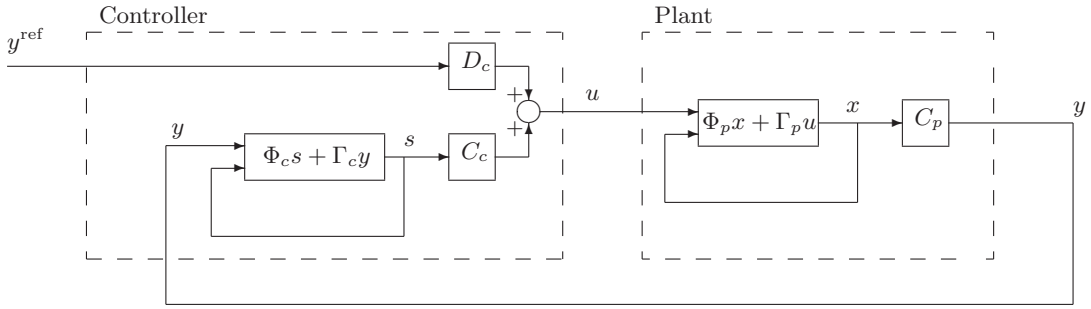


Figure 1.2: The conventional control scheme.

It can be seen immediately that this scheme bears resemblance to the DMAS representation in Figure 1.1. The most notable difference is the reduction to a single agent / controller.

In fact, the control scheme is equivalent to a single-agent, deterministic DMAS. Clearly, as the notations indicate, y corresponds to the agent's observation, s to its state, u to the agent's action, and x to the environment's state. The correspondence between the linear relations (1.1 – 1.2) and the functional elements of the DMAS is the following:

$$\overline{f}(x_k, u_k) = \Phi_p x_k + \Gamma_p u_k \tag{1.3}$$

$$\overline{\omega}(x_k) = C_p x_k \tag{1.4}$$

$$p(s_k, y_k) = \Phi_c s_k + \Gamma_c y_k \tag{1.5}$$

$$\overline{h}(s_k) = C_c s_k + D_c y_k^{\text{ref}} \tag{1.6}$$

An interesting difference can be observed: the DMAS has no explicit element similar to the reference signal (the control goal) in Figure 1.2. This opens the way for two interpretations:

- The goal is internal to the agent. The agent knows what it wants to achieve, pursues its goal, and is the only arbiter of its own actions.
- The goal is a component of the agent's observation (input). It is imposed by an external authority, such as a human, or another agent one level higher in a hierarchical control scheme.

The scheme in Figure 1.2 isolates the plant. In DMAS, the plant corresponds to the environment, and nothing places itself between the agents and the environment, such as noisy transmission channels for the command or feedback signals. These are all modeled as part of the environment.

If we allow the controller to be time varying:

$$\begin{cases} s_{k+1} = \Phi_{c,k} s_k + \Gamma_{c,k} y_k \\ u_k = C_{c,k} s_k + D_{c,k} y_k^{\text{ref}} \end{cases} \tag{1.7}$$

we can identify it with an adaptive agent:

$$\begin{cases} p_k(s_k, y_k) = \Phi_{c,k} s_k + \Gamma_{c,k} y_k \\ \overline{h}_k(s_k) = C_{c,k} s_k + D_{c,k} y_k^{\text{ref}} \end{cases} \tag{1.8}$$

1.4.2 Model-reference adaptive control

The model reference adaptive system is an adaptive control scheme that expresses the control goal in terms of a reference model. The reference signal is passed through this model to generate the desired response of the plant. The control law is parameterized, and the parameters are adapted by an adjustment mechanism in such a way that the difference between the actual and the desired output of the plant is minimized (Figure 1.3).

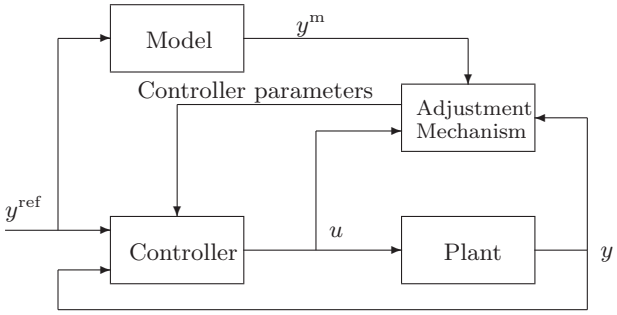


Figure 1.3: The model-reference adaptive control scheme.

We assume the most general form for the plant and the (parameterized) controller:

$$\begin{cases} x_{k+1} = f_p(x_k, u_k) \\ y_k = h_p(x_k) \end{cases} \tag{1.9}$$

$$\begin{cases} s_{k+1} = \tilde{f}_c(s_k, y_k^{\text{ref}}, y_k, \theta_k) \\ u_k = \tilde{h}_c(s_k, y_k^{\text{ref}}, \phi_k) \end{cases} \tag{1.10}$$

where y^{ref} is the command input. This signal generates the desired plant output y^{m} via the reference model.

Similarly to what was done in Section 1.4.1, we can establish a correspondence between a single-agent, deterministic AL-MAS and (1.9 – 1.10):

$$\overline{f}(x_k, u_k) = f_p(x_k, u_k) \tag{1.11}$$

$$\overline{\omega}(x_k) = h_p(x_k) \tag{1.12}$$

$$\tilde{p}(s_k, y_k, \theta_k) = \tilde{f}_c(s_k, y_k^{\text{ref}}, y_k, \theta_k) \tag{1.13}$$

$$\tilde{h}(s_k, \phi_k) = \tilde{h}_c(s_k, y_k^{\text{ref}}, \phi_k) \tag{1.14}$$

The controller is now an adaptive agent using parametric adaptation. Again, the goal representation y^{ref} is in our framework internal to the agent.

The adjustment mechanism adapts the controller parameters:

$$\theta_{k+1} = \tilde{\theta}(\theta_k, u_k, y_{m,k}, y_k) \tag{1.15}$$

$$\phi_{k+1} = \tilde{\phi}(\phi_k, u_k, y_{m,k}, y_k) \tag{1.16}$$

and corresponds to the adaptation mappings in Definition 1.8:

$$\tilde{\mathcal{P}}(\theta_k, s_k, y_k) = \tilde{\theta}(\theta_k, u_k, y_{m,k}, y_k) \tag{1.17}$$

$$\tilde{\mathcal{H}}(\phi_k, s_k, y_k) = \tilde{\phi}(\phi_k, u_k, y_{m,k}, y_k) \tag{1.18}$$

The model of the plant is in the DMAS interpretation part of the agent. Its state vector x^{m} can be interpreted as part of an extended state signal $\overline{s} = [s, x^{\mathrm{m}}]^{\mathrm{T}}$. Similarly, its dynamics f^{m} can be seen as part of extended agent dynamics. The output function of the model is integrated into the adaptation mappings $\tilde{\mathcal{P}}$ and $\tilde{\mathcal{H}}$, which use it to compute y^{m}.

1.4.3 A communicative multiagent team model

Our last example is a multiagent teamwork model developed by Pynadath and Tambe (2002) with the purpose of analyzing the tradeoff between optimality and complexity in teamwork models and theories. We choose to illustrate this model for two reasons: first, it is very general and representative for the multiagent literature, and second, it has been one of the sources of inspiration for defining our (communicative) DMAS model. The model subsumes among others Markov decision processes and team Markov games with deterministic rewards, and is easily extensible to handle stochastic rewards and conflicting goals situations. These frameworks are, as we shall see later, very important for multiagent learning.

The name of the model is the *communicative Markov team decision problem (COM-MTDP)*. Given a team of agents A, a COM-MTDP is a tuple $\langle X, U_A, M_A, f, Y_A, \omega_A, B_A, \rho \rangle$ where:

- X is a set of world states.
- $U_A = \times_{i \in A} U_i$ is a set of combined actions, where U_i is the set of actions for agent i.
- $M_A = \times_{i \in A} M_i$ is a set of combined messages, where M_i is the set of messages for agent i.
- f is the state transition probability distribution, $f(x, \boldsymbol{u}, x') = P\left(x_{k+1} = x' \,|\, x_k = x, \boldsymbol{u}_k = \boldsymbol{u}\right)$.
- $Y_A = \times_{i \in A} Y_i$ is a set of observations, where U_i is the set of observations for agent i.
- ω_A is the joint observation distribution, $\omega_A(x, \boldsymbol{u}, \boldsymbol{y}) = P\left(\boldsymbol{y}_k = \boldsymbol{y} \,|\, x_k = x, \boldsymbol{u}_k = \boldsymbol{u}\right)$. The observation distribution ω_i for agent i is given by $\omega_i(x, \boldsymbol{u}, y_i) = P\left(y_{i,k} = y_i \,|\, x_k = x, \boldsymbol{u}_k = \boldsymbol{u}\right)$.
- $B_A = \times_{i \in A} B_i$ is a set of combined belief states, where B_i is the set of possible belief states for agent i.
- ρ is the common reward function of the team, representing its joint preference over states and the cost of actions and communication, $\rho : X \times M_A \times U_A \to \mathbb{R}$. The reward function is the sum of two rewards: (i) $\rho^{\mathrm{e}} : X \times U_A \to \mathbb{R}$, representing the utility of the domain actions, and (ii) $\rho^{\mathrm{c}} : X \times M_A \to \mathbb{R}$, representing the cost of communication.

The agent decides which actions to take and which messages to send on the basis of its belief state $b_i \in B_i$. The domain actions are dictated by a domain-level policy $h_i^{\mathrm{e}} : B_i \to U_i$. Similarly, the sent messages are dictated by a communication policy $h_i^{\mathrm{c}} : B_i \to M_i$.

The belief state is updated in two stages at each time step: once when the agent observes the world ("pre-communication"), and a second time when it receives the joint message sent by the team ("post-communication"). The updates are performed via so-called

"state-estimators": pre-communication, $\beta_i^{\text{pre}} : B_i \times Y_i \to B_i$, and post-communication, $\beta_i^{\text{post}} : B_i \times M_A \to B_i$, respectively.

At every step, each agent executes an action and sends a message as indicated by its domain-level and communication policies, respectively, and the world evolves in consequence.

We proceed to show that a COM-MTDP can be described by a communicative DMAS (Definition 1.10), and to enumerate the differences between the two frameworks.

The correspondences between the COM-MTDP and communicative DMAS elements are summarized in Table 1.1.

COM-MTDP	**Communicative DMAS**
world state space X	environment state space X^{e}
domain-level action space U_i	domain-level action space U_i^{e}
set of messages M_i	set of messages M_i
transition distribution f	environment transition distribution f^{e}
set of observations Y_i	set of environment observations Y_i^{e}
observation distributions ω_i	environment observation distributions ω_i^{e}
belief state space B_i	(included in) agent internal state space S_i
pre-communication estimators β_i^{pre}	(included in) observations agent dynamics p_i^{e}
post-communication estimators β_i^{post}	messages handling agent dynamics p_i^{rcv}
reward function ρ	(implicit in) the env. observation distributions ω_i^{e}
domain-level policy h_i^{e}	domain-level policy h_i^{e}
communication policy h_i^{c}	message sending policy h_i^{snd}
–	channel state space X^{c}
–	channel dynamics f^{c}
–	message receival distributions ω_i^{rcv}

Table 1.1: Correspondences between the COM-MTDP and the communicative DMAS elements

Many of these correspondences are immediate. The following differences exist:

- The DMAS model does not assume that an authority providing a reward signal exists in the environment; instead, the agent must maintain some kind of goal representation in its internal state s_i, and judge the quality of its actions only on the basis on their effect on the environment, monitored via its observations ω_i^{e}. Nevertheless, an explicit reward function can be assumed to be part of the agent's observation function ω_i.

- In the view of COM-MTDP, the agent's internal state is used only to represent beliefs on the world's state, and consequently, the agent dynamics are used only to update these beliefs. The DMAS view is broader: the agent's internal state includes all the information relevant to the decision making process of the agent, possibly including among others a-priori knowledge, learned knowledge, and prediction results. Consequently, the agent's dynamics can include, among other things, learning and prediction processes.

- COM-MTDP does not have any explicit representation of the communication channel dynamics or of the message receival process. As such, it cannot describe transmission or receival noise. In contrast to the COM-MTDP, in the communicative DMAS communication is not added on top of preexisting structures, but arises naturally from the basic DMAS framework.

Therefore, the communicative DMAS is a more general model than the COM-MTDP, offering additional expressiveness in the form of agent internal dynamics and explicit communication dynamics.

Chapter 2

Multiagent reinforcement learning

2.1 Introduction

Reinforcement learning (RL) is the problem faced by an agent that must learn behaviour through trial-and-error interactions with a dynamic environment (Kaelbling et al., 1996; Sutton and Barto, 1998). The RL field was born from the junction of several research threads in various disciplines, among which the most important are trial-and-error learning in animal psychology and the dynamic programming approach to optimal control in control engineering (Sutton and Barto, 1998).

Reinforcement learning assumes the following interaction model between the agent and its environment (we use the terms "world" and "environment" interchangeably in the RL context): at each discrete time step, the agent observes the current state of the environment, and chooses an action. As a result, the environment transitions into a new state, and the agent receives a scalar reward signal (see Figure 2.1). This signal is a measure of the quality of the agent's actions, as determined by the environment (this placement of the evaluation process in the environment is characteristic to RL).

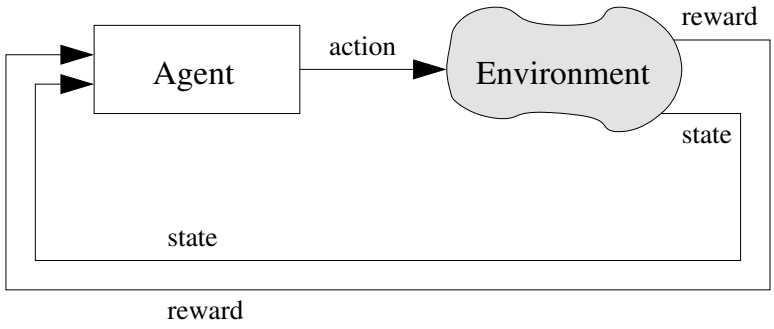


Figure 2.1: The reinforcement learning model.

Consider for instance, a "mouse" agent in a maze, who needs to find the exit. The actions of the mouse are its movements; the state consists of its position, and changes as a result of the mouse's movements. The mouse receives zero reward while it is inside the maze, and a positive reward ("cheese") when it finds the exit. An added incentive to find the exit quickly can be provided to the mouse by giving it a small negative reward ("penalty" in the RL terminology) instead of zero, at each time step it spends inside the maze.

This simple example problem illustrates the two fundamental characteristics of reinforcement learning: trial-and-error search and delayed reinforcement (Sutton and Barto, 1998). The mouse does not receive any indication as to what movement would have been best at some junction in the maze, as it would be the case in supervised learning. Instead, it is only informed of the relative quality of its movement via the reward signal, and it needs to find out by itself which moves are best – this is *trial-and-error search*. Assume now that the mouse made a crucial decision at a maze junction close to its start point. That decision enabled the agent to reach the maze exit after a certain time τ; however, the positive reward that was obtained in a significant part in consequence of that decision, is given to the agent only after the interval τ has elapsed. It is up to the agent to infer which decisions along its trajectory contributed to it obtaining the final reward, and with what weight – this is *delayed reinforcement*.

It is clear that learning techniques based on such a model are very attractive, regardless of whether one or more agents are involved. When the agents are reinforcement learners, the task of the designer reduces in a first approximation to specifying a reinforcement signal that accurately represents the goals set to the multiagent system (this is not an easy task, however). Of course, when more than one agent is involved, the environment is no longer the static, reactive machine represented in Figure 2.1.

Due to the strong appeal of RL, much of the research in multiagent learning focuses on this field. As such, this chapter constitutes the main body of the part of our review dealing with learning in multiagent systems. A survey of other multiagent learning methods can be found in Chapter 3.

The chapter is organized in the following way: the RL problem is formally introduced in the single-agent case, together with its solution concepts and techniques. We proceed then to present some of the formal representations and solution concepts of the multiagent reinforcement learning (multiagent reinforcement learning (MA-RL)) problem explored in the literature . We present a selection of methods used by researchers to solve this problem, and conclude with some remarks and research opportunities.

2.2 Single agent reinforcement learning

Most multiagent reinforcement learning algorithms build on single-agent methods. We therefore thoroughly introduce the theory and techniques employed by these methods.

2.2.1 Formal model

In the independent learning agent case, the reinforcement learning task is typically formalized as a Markov decision process.

Definition 2.1 *A* finite Markov decision process (MDP) *is a tuple* $\langle X, U, \rho, f \rangle$ *where* X *is the discrete and finite state space,* U *is the discrete and finite action space,* $\rho : X \times U \times X \times \mathbb{R} \to [0, 1]$ *is the reward probability distribution, and* $f : X \times U \times X \to [0, 1]$ *is the state transition probability distribution.*

There is also a general type of MDP with possibly infinite / continuous state and action spaces. There exist methods of handling such types of problems, though the theoretical guarantees valid for approaches dealing with the finite case typically do not carry over to those methods. For the remainder of the presentation, we consider only finite MDPs.

The behaviour of the RL agent in an MDP is described by a possibly stochastic policy mapping states into actions, $h : X \times U \to [0, 1]$. This policy is changed over time by the reinforcement learning algorithm.

Using this formal model, the RL iteration can be described as follows: the agent observes the current state x_k of the environment, and chooses an action u_k. As a result, the environment transitions into a new state x_{k+1} with probability $f(x_k, u_k, x_{k+1})$ and the agent receives a reward r_{k+1} with probability $\rho(x_k, u_k, x_{k+1}, r_{k+1})$.

The MDP fits into our DMAS model (Definition 1.2) by reduction to one agent (denoted as agent 1) and assumption of full measurability. The MDP represents only the learning tasks, so the DMAS elements internal to the agent are missing. Furthermore, our model does not assume that an authority providing a reward signal exists in the environment, as RL does; instead, in the general case the agent must maintain some kind of goal representation in its internal state s_1, and judge the quality of its actions only on the basis on their effect on the environment. Nevertheless, an explicit reward function can be assumed to be part of the agent's observation function ω_1. Hence, as the environment is fully measurable, the observation of the agent is of the form $y_{1,k} = [x_k, r_k]^{\mathrm{T}}$.

These correspondences are summarized in Table 2.1.

MDP	**DMAS**
state space X	environment state space X
action space U	action space U_1
transition distribution f	transition distribution f
reward distribution ρ	(implicit in) the observation distribution ω_1
policy h	policy h_1

Table 2.1: Correspondences between the MDP and the DMAS elements

The RL policy h is time-varying as described before, under the influence of the learning algorithm. In our model, the policy h_1 is not time-varying, but is a function of the time-varying internal state s_1, which is changed by the learning processes p_1.

Since most of our discussion on RL does not explicitly address the internal agent state, throughout this chapter by simply stating "state" we mean "environment state". When discussing the internal agent state, we declare it explicitly.

2.2.2 Learning goal

The RL agent is a rational agent that uses an optimality measure expressed in terms of its rewards. The most common ways of expressing this optimality measure are (Kaelbling et al., 1996):

(i) The *finite horizon expected return*:

$$R_k = E\left\{\sum_{l=0}^{T} r_{k+l+1}\right\}. \tag{2.1}$$

This measure can only be used for episodic tasks.

(ii) The infinite horizon *expected discounted return*:

$$R_k = E\left\{\sum_{l=0}^{\infty} \gamma^l r_{k+l+1}\right\}. \tag{2.2}$$

This measure is suitable for both episodic and continuing tasks. The variable $\gamma \in [0, 1)$ is the *discount factor*, and may be interpreted in several ways:

- a probability that the agent survives to the next step;
- a measure of how "far-sighted" the agent is in considering its rewards;
- a mathematical trick to bound the otherwise infinite sum.

(iii) The infinite horizon *expected average return*:

$$R_k = \lim_{h\to\infty} E\left\{\frac{1}{h}\sum_{l=0}^{h} r_{k+l+1}\right\}. \tag{2.3}$$

This measure is also appropriate for both episodic and continuing tasks, and can be seen as the limiting case of the discounted expected return when the discount factor approaches 1 (Kaelbling et al., 1996).

We use the expected discounted return throughout our review, since it is the most popular and can handle both continuing and episodic tasks (including episodic tasks repeated along the agent's life).

2.2.3 The Markov property

The most important assumption that many theoretical results on reinforcement learning rely on is the Markov property. This property refers to the state signal and is satisfied if this signal contains at each step all the information relevant to the agent's decision making process. The state signal should also be, of course, as compact as possible.

Formally, the Markov property is expressed by the following identity (Sutton and Barto, 1998):

$$P\left(x_{k+1} = x, r_{k+1} = r \,|\, x_k, u_k, x_{k-1}, u_{k-1}, \ldots, x_0, u_0\right) = P\left(x_{k+1} = x, r_{k+1} = r \,|\, x_k, u_k\right). \tag{2.4}$$

That is, it is sufficient to know the last state and action to determine the next state and reward. We call a task that satisfies this property a "Markov" task.

2.2.4 Value functions and the Bellman equations

Almost all RL algorithms rely on estimating how good it is for the agent to be in a given state, or to take a given action in a given state. These estimates are embodied by the *state value function* and *action value function*, respectively (Sutton and Barto, 1998).

Definition 2.2 *The value of a state x under a policy h is the expected return when starting in x and following h thereafter:*

$$V^h(x) = E_h\left\{R_k \,|\, x_k = x\right\}. \tag{2.5}$$

Under the expected discounted return (2.2), this is:

$$V^h(x) = E_h\left\{\sum_{l=0}^{\infty}\gamma^l r_{k+l+1}\,\middle|\, x_k = x\right\}. \tag{2.6}$$

Definition 2.3 *The value of taking action u in state x under a policy h is the expected return when starting in x, taking u and following h thereafter:*

$$Q^h(x,u) = E_h\left\{R_k \,|\, x_k = x, u_k = u\right\}. \tag{2.7}$$

Under the expected discounted return (2.2), this is:

$$Q^h(x,u) = E_h\left\{\sum_{l=0}^{\infty}\gamma^l r_{k+l+1}\,\middle|\, x_k = x, u_k = u\right\}. \tag{2.8}$$

Action-values are also called "Q-values". We also sometimes call the Q-function "Q-table" since most of the results presented in this chapter rely on a complete, tabular representation of the Q-function, indexed by states and actions.

The fundamental property of value functions is that they satisfy the following recursive equations for any policy h (assuming deterministic rewards to prevent the notation from becoming cluttered):

$$V^h(x) = \sum_{u\in U} h(x,u) \sum_{x'\in X} f(x,u,x')\left[\rho(x,u,x') + \gamma V^h(x')\right] \quad \forall x \in X, \tag{2.9}$$

$$Q^h(x,u) = \sum_{x'\in X} f(x,u,x')\left[\rho(x,u,x') + \gamma \sum_{u'\in U} h(x',u')Q^h(x',u')\right] \quad \forall x \in X, u \in U. \tag{2.10}$$

Definition 2.4 (Optimality in MDPs)

(i) The optimal state value function is the function associating each state with the maximal attainable value from that state:

$$V^*(x) = \max_h V^h(x), \quad \forall x \in X. \tag{2.11}$$

(ii) The optimal action value function is the function associating each state-action pair with the maximal attainable value after taking that action in that state:

$$Q^*(x,u) = \max_h Q^h(x,u), \quad \forall x \in X, u \in U. \tag{2.12}$$

(iii) An optimal policy h^* is a policy attaining the optimal state value function or the optimal action value function.

An MDP always has a deterministic optimal policy, given by:

$$h^*(x) = \arg\max_{u\in U} Q^*(x,u). \tag{2.13}$$

There may in fact be more optimal policies, but all attain the same state and action value functions – the optimal ones.

Clearly, the optimal value functions must satisfy the recursive relations (2.9), (2.10). The obtained expressions are the *Bellman optimality equations*:

$$V^*(x) = \max_{u \in U} \sum_{x' \in X} f(x, u, x') \left[\rho(x, u, x') + \gamma V^*(x') \right] \quad \forall x \in X, \tag{2.14}$$

$$Q^*(x, u) = \sum_{x' \in X} f(x, u, x') \left[\rho(x, u, x') + \gamma \max_{u' \in U} Q^*(x', u') \right] \quad \forall x \in X, u \in U. \tag{2.15}$$

These equations are the cornerstone of most RL solution techniques.

2.2.5 The exploration issue

Equation (2.13) suggests the agent can choose at each step the action that, according to the action value function, yields the maximum value. This is called *greedy* action selection; if the agent chooses the greedy action, we say it is "exploiting". The greedy action would, of course, be the best idea if the agent knew the optimal value function. During learning, however, the agent has only an estimate of this value function. It might be that some action is underestimated and is actually better than the action with the highest value in the current value estimate. At each step, the learning agent must weigh the direct benefits of choosing the greedy action with the possible benefit of choosing some other action to find out whether it is not, in fact, better. The latter course of action is called *exploration*, and when an agent follows it we say that it is "exploring".

In the maze example, assume the mouse agent has chosen at a T junction to go left, but that the exit is very close to the right path, while also reachable via the left path. If the mouse actually reaches it, chances are that the estimated value of the left path will become greater than that of the right path. Then, the next time the mouse is at the same junction, according to its estimates it will prefer going left; this is the greedy action. However, if the mouse chooses to explore instead and goes right, it will find out that this underestimated path actually gets it to the goal much faster.

When reinforcement is delayed, we must rely on heuristics or approximations in balancing exploration with exploitation. Thrun (1992) classified exploration strategies in two categories, directed and undirected. Exploration is directed when it takes into account some measure of the expected gain in information obtained by following exploratory moves, and attempts to maximize this measure. When such a measure is not used, exploration is undirected.

Undirected exploration

Though the efficiency of exploration has tremendous consequences on the learning performance, most RL researchers use simple stochastic strategies. The two most common of these are (for a detailed discussion, see Singh et al., 2000a):

(i) *ε-greedy* exploration. This strategy chooses at each step the greedy action with probability $1 - \varepsilon$, and some other random action with probability ε, $\varepsilon \in (0, 1]$. The exploration probability ε is typically initialized at a relatively high value and decays as learning progresses.

(ii) *softmax* (or "Boltzmann") exploration. The basic mechanism is the same as for ε-greedy exploration, but when exploring, this strategy does not select a random action. Instead, it ranks actions based on their values, and then chooses one of them stochastically. Specifically, the probability of selecting action u when exploring from state x is:

$$P(u \mid x) = \frac{e^{Q(x,u)/\tau}}{\sum_{u' \in U} e^{Q(x,u')/\tau}}. \tag{2.16}$$

The "temperature" parameter τ controls the randomness of the action selection. When at high values, the actual value of the actions has little influence on what action is selected. When τ has low values, however, the softmax strategy approaches greedy action selection.

Another option is optimism in the face of uncertainty: the action value function is initialized with overestimated values, such that "disappointment" experienced from the already tried actions drives the agent to explore new ones (Sutton and Barto, 1998). If the initial values are sufficiently high (for an example of a special case where the initial values are exactly computable, see Sen et al., 1994) and given enough time, the agent will explore the entire state-action space. This is not always desirable; it might be better to initialize the action value function with moderate values so that the agent doesn't waste time exploring everything.

Directed exploration

Thrun (1992) further separates directed exploration strategies into three classes:

(i) *frequency-based* exploration remembers how often states or state action pairs were visited. An illustrative variant of this counts the number of times each state has been visited, and chooses in every state the action that will move the agent into the least-visited adjacent state (Thrun, 1992). The goal is to improve the agent's knowledge in states about which it has (presumably) the least information.

(ii) *recency-based* exploration relies on the elapsed time since an action was last tried. The exploration is biased towards actions that weren't tried in a long time, in order to improve the knowledge of the agent on their effects.

(iii) *error-based* exploration uses second order information to estimate the uncertainty in the estimated action values. Exploration is biased towards actions that have higher potential. One such method is Kaelbling's interval estimation heuristic (Kaelbling et al., 1996). This method computes an upper bound of a confidence interval (for instance, a 95% interval) for all actions and chooses the action with the highest upper bound.

De Jong (1997) attempted to combine the advantages of recency-based and error-based methods into the exploration buckets method. In this method, all actions have an associated exploration "bucket". At each step, a quantity related to the error in the reward predicted by the action last time it was chosen, is added to its bucket. Buckets of chosen actions are emptied. The error-based character of the method is obvious; the recency-based character is given by the fact that biases accumulate in the exploration buckets.

2.2.6 Solution techniques

Model-based techniques

Model based methods assume a model of the environment (reward and transition functions) is available, and compute the optimal course of action given that model. It is easy to do this indirectly, by first computing an optimal value function and then using greedy action selection. The method is called value iteration, and is presented in Algorithm 2.1.

Algorithm 2.1 Value iteration

Require: f, the transition model; ρ, the reward model
Input: threshold $\theta \geq 0$, discount factor $\gamma \in [0, 1)$
Output: an optimal policy h^*

1: $Q(x, u) \leftarrow 0, \quad \forall x \in X, u \in U$
2: **repeat**
3: $\quad \delta \leftarrow 0$
4: $\quad$ **for all** $x \in X, u \in U$ **do**
5: $\quad\quad q \leftarrow Q(x, u)$
6: $\quad\quad Q(x, u) \leftarrow \sum_{x' \in X} f(x, u, x') \left[\rho(x, u, x') + \gamma \max_{u' \in U} Q(x', u')\right]$
7: $\quad\quad \delta \leftarrow \max\{\delta, |Q(x, u) - q|\}$
8: $\quad$ **end for**
9: **until** $\delta \leq \theta$
10: $h^*(x) \leftarrow \arg\max_{u \in U} Q(x, u), \quad \forall x \in X$ ▷ breaking ties randomly

The algorithm turns the Bellman equation (2.15) into an assignment. Convergence to the optimal value function can be proven under certain conditions.

Policy iteration operates directly on the policy of the agent (Algorithm 2.2). It works by iteratively executing two interdependent processes: policy evaluation, where the value function of the current policy is computed, and policy improvement, where this value function is used to compute a new policy. It can be shown that policy evaluation converges to the true value of the policy and that policy improvement yields a better policy except when the policy is optimal – hence the stopping condition of the algorithm.

Equivalent value and policy iteration algorithms using the state value function exist (see Sutton and Barto, 1998). Since most practical solution methods rely on action values, we focus here on the action value function

These techniques are directly derived from dynamic programming in optimal control, and therefore bear its name.

Model-free techniques

In many cases, it is difficult or impossible to obtain the accurate models of the world required by model-based techniques. Moreover, the state space of some problems is very large and processing all the states might not be practical. These problems are solved by model-free methods.

The Monte Carlo class of model-free methods evaluates a policy by actually executing large numbers of episodes with that policy and averaging over the obtained returns (Sutton and Barto, 1998). Since returns are obtained by direct experience in the environment, a model is no longer needed. Moreover, the computation can be focused on the interesting areas of

Algorithm 2.2 Policy iteration

Require: f, the transition model; ρ, the reward model
Input: threshold $\theta \geq 0$, discount factor $\gamma \in [0, 1)$
Output: an optimal policy h^*

$h(x) \leftarrow$ a random action $u \in U, \quad \forall x \in X$
repeat
  $Q(x, u) \leftarrow 0, \quad \forall x \in X, u \in U$
  repeat ▷ policy evaluation
    $\delta \leftarrow 0$
    for all $x \in X, u \in U$ **do**
      $q \leftarrow Q(x, u)$
      $Q(x, u) \leftarrow \sum_{x' \in X} f(x, u, x') \left[\rho(x, u, x') + \gamma Q(x', h(x'))\right]$
      $\delta \leftarrow \max\{\delta, |Q(x, u) - q|\}$
    end for
  until $\delta \leq \theta$
  $h_{\text{stable}} \leftarrow$ true ▷ policy improvement
  for all $x \in X$ **do**
    $v \leftarrow h(x)$
    $h(x) \leftarrow \arg\max_{u \in U} Q(x, u)$ ▷ breaking ties randomly
    if $h(x) \neq v$ **then**
      $h_{\text{stable}} \leftarrow$ false
    end if
  end for
until h_{stable}
$h^* \leftarrow h$

the state space where useful policies evolve. For instance, if the mouse in our example starts somewhere near the exit of a very large maze, it is not interested in computing accurate values for states far back from its starting position.

The advantages of Monte Carlo methods are obvious; the disadvantage is that they require large numbers of episodes, which may be costly to obtain. Moreover, continuing tasks do not have any well-defined "end". *Temporal difference* methods are able to perform useful updates at each step taken in the world. They combine the ideas of dynamic programming and Monte Carlo methods. The actual reward obtained by experience in the world is used, like in Monte Carlo methods – except that temporal difference performs updates using this reward at each step, not only after long experience sequences. The update rule moves the current value estimate towards a target formed by combining the observed reward with the current estimated value of the next state – like value iteration, but using actual experience instead of the model to obtain the next state and reward:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha \left[r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)\right]. \tag{2.17}$$

Here, the agent chose u_k in state x_k at time step k, and as a result the world changed state to x_{k+1}, rewarding the agent by r_{k+1}. The action u_{k+1} is that chosen by the agent at time step $k + 1$. A few interesting things are worth noting about this update rule:

- The estimate $Q(x_k, u_k)$ is not moved all the way to the target $[r_{k+1} + \gamma Q(x_{k+1}, u_{k+1})]$, as value iteration would do. Instead, it is moved only a fraction α of the distance. This

fraction is called "step-size" or *learning rate*. It typically varies decreasingly during the learning process.

- The estimate $Q(x_k, u_k)$ is updated using another estimate, $Q(x_{k+1}, u_{k+1})$. This technique is called bootstrapping.
- The agent uses its next action u_{k+1} to determine the value of the next state. Hence, the estimates will converge to the value of the actually used policy. The update rule is said for this reason to be "on-policy".

The on-policy algorithm resulting from the combination of (2.17) with a policy derived from the current action value estimates is called SARSA (from the structure of the experience instance used by the update rule: state-action, reward-state-action). Under certain conditions, it can be shown that state-action, reward-state-action (SARSA) converges to the optimal policy. These conditions notably include an exploratory policy with decaying exploration so that in the limit the policy becomes greedy; and a learning rate series that sums up to infinity, but whose squares sum up to a finite value.

A disadvantage of SARSA is that the agent cannot know the actual optimal action values until the policy has converged to the optimal policy, whereas the convergence conditions require the policy to be exploratory and thus non-optimal. It turns out that temporal difference methods can actually estimate the optimal value function while following a sub-optimal policy – such methods are characterized as "off-policy". The modification required for the update rule is straightforward:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha \left[r_{k+1} + \gamma \max_{u' \in U} Q(x_{k+1}, u') - Q(x_k, u_k) \right]. \tag{2.18}$$

The algorithm resulting from the combination of (2.18) with a policy derived from the current action value estimates is called Q-learning and is presented in Algorithm 2.3 (Watkins and Dayan, 1992).

Algorithm 2.3 Q-learning

Input: learning rate α, discount factor γ

1: $Q(x, u) \leftarrow 0, \quad \forall x \in X, u \in U$
2: observe initial state x
3: **loop**
4: $u \leftarrow h(x)$ where h is a policy derived from Q (e.g., ε-greedy)
5: apply u, observe r and x'
6: $Q(x, u) \leftarrow Q(x, u) + \alpha \left[r + \gamma \max_{u' \in U} Q(x', u') - Q(x, u) \right]$
7: $x \leftarrow x'$
8: **end loop**

It has been shown that Q-learning converges to the optimal policy under more relaxed conditions than SARSA (Watkins and Dayan, 1992) (the only condition imposed on the policy is that all state-action pairs should continue to be updated). The initialization of Q-values to 0 is arbitrary, any finite initial values satisfy the convergence properties. An immediate advantage of this is that optimistic initial values can be used (Section 2.2.5).

In the multiagent context, we sometimes refer to this original variant of Q-learning as "basic" or "plain" Q-learning.

Q-learning and its derivations are the most widely used reinforcement learning algorithms. We therefore analyze how they fit into the DMAS framework. Table 2.1 and the accompanying exposition already explained how the basic MDP components fit into DMAS. What we need to investigate deeper is the place of the elements specific to Q-learning: the Q-table and the update rule (2.18).

In the DMAS interpretation, the Q-table is part of the agent's knowledge – that is, of its internal state. So, the Q-table corresponds to a part of the agent's internal state vector s (since there is only one agent, we omit the agent index). Denote this "flat" Q-vector by $q \in \mathbb{R}^m$, where $m = |X| \cdot |U|$. Denote its value at step k by q_k.

In order to bring forth the agent dynamics in the temporal difference update rules, we introduce the selector function:

$$J : X \times U \to \mathbb{R}^m,\ J_i(x,u) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}, \quad i = 1, \dots, m \tag{2.19}$$

where j is the position of the element corresponding to x and u in q. So, given x and u, the multiplication $J^{\mathrm{T}}(x,u)q$ produces a scalar equal to the Q-value of the pair (x,u). Also, $J(x,u)$ multiplied by a scalar produces a vector with 0 on all positions except that corresponding to the pair (x,u) where the scalar is placed.

Let us first rewrite the SARSA update rule (2.17) using J:

$$\begin{aligned} q_{k+1} &\leftarrow q_k + \alpha J(x_k,u_k)\left[r_{k+1} + \gamma J^{\mathrm{T}}(x_{k+1},u_{k+1})q_k - J^{\mathrm{T}}(x_k,u_k)q_k\right] \\ &= q_k + \alpha\gamma J(x_k,u_k)J^{\mathrm{T}}(x_{k+1},u_{k+1})q_k - \alpha J(x_k,u_k)J^{\mathrm{T}}(x_k,u_k)q_k + \alpha J(x_k,u_k)r_{k+1} \\ &= \left[I + \alpha\gamma J(x_k,u_k)J^{\mathrm{T}}(x_{k+1},u_{k+1}) - \alpha J(x_k,u_k)J^{\mathrm{T}}(x_k,u_k)\right] q_k + \alpha J(x_k,u_k)r_{k+1} \end{aligned} \tag{2.20}$$

where I is the $m \times m$ identity matrix.

These *learning dynamics* are linear in q. The agent needs to remember the previous state of the environment x_k and its previous action u_k. It does that by storing them on its internal state. The state of the agent is then composed of the Q-vector together with the previous environment state and chosen action: $s_{k+1} = [q_k, x_k, u_k]^{\mathrm{T}}$. The index shift appears because of the notation convention in the DMAS model: the knowledge q_k on the basis of which the agent decides at time step k is already updated with the observation y_k; hence, it is denoted by index $k+1$ (see Section 1.2 for the detailed discussion of this aspect).

Therefore, (2.20) is not linear in the agent internal state, but the nonlinearity appears in elements that only serve the purpose of short-term memory. Remembering that the observation of the agent has the form $y_k = [x_k, r_k]^{\mathrm{T}}$, (2.20) can be written (considering the explained index shift):

$$q_{k+1} \leftarrow \Phi(s_{k+1}, y_{k+1})q_k + \Gamma(s_{k+1}, y_{k+1}). \tag{2.21}$$

This equation describes the learning dynamics of the agent, which form a part of the agent's internal dynamics:

$$s_{k+2} \leftarrow p(s_{k+1}, y_{k+1}). \tag{2.22}$$

The interpretation of the Q-learning dynamics is very similar, but the learning dynamics are nonlinear in q and would involve messy notation. We give here just the rewritten version of the Q-learning update rule (2.18):

$$q_{k+1} \leftarrow q_k + \alpha J(x_k, u_k) \left[r_{k+1} + \gamma \max_{u' \in U} J^{\mathrm{T}}(x_{k+1}, u') q_k - J^{\mathrm{T}}(x_k, u_k) q_k \right]. \tag{2.23}$$

The Q-learning update rule (2.18) propagates information one step back along the trajectory of the agent in the state space. In our maze example, this means that when the mouse reaches the cheese for the first time, information about the large obtained reward is propagated only to the visited state nearest to the cheese, say denoted by x. If the mouse is then put back in some other position in the maze, it will have no idea of the way to the cheese, until it somehow reaches a state near x and then moves into x. At this point, information is propagated back another step, and so on. Clearly this is quite inefficient. This issue is actually an instantiation of the temporal credit assignment problem.

There is actually a much more efficient way of propagating value information. The agent marks its trajectory with a so-called "eligibility trace" E as it passes, and updates at each step not only the last Q-value but an entire set of Q-values along its path, proportionally to their eligibility values. The trace decays exponentially by a factor of λ called the recency factor. The resulting algorithm is called Q(λ) and is presented in Algorithm 2.4.

Algorithm 2.4 Q(λ) (with accumulating trace)

Input: learning rate α, discount factor γ, recency factor λ

1: $Q(x,u) \leftarrow 0, \quad E(x,u) \leftarrow 0, \quad \forall x \in X, u \in U$
2: observe initial state x
3: **loop**
4: $u \leftarrow \arg\max_{u' \in U} Q(x,u')$
5: modify u to exploratory move if and as indicated by the exploration strategy
6: **if** $u \neq \arg\max_{u' \in U} Q(x,u')$ **then**
7: $E(x,u) \leftarrow 0, \quad \forall x \in X, u \in U$ ▷ reset trace
8: **end if**
9: apply u, observe r and x'
10: $\delta \leftarrow r + \gamma \max_{u' \in U} Q(x',u') - Q(x,u)$ ▷ compute temporal difference
11: $E(x,u) \leftarrow E(x,u) + 1$ ▷ mark trace
12: **for all** $\tilde{x} \in X, \tilde{u} \in U$ **do**
13: $Q(\tilde{x},\tilde{u}) \leftarrow Q(\tilde{x},\tilde{u}) + \alpha\delta E(\tilde{x},\tilde{u})$
14: $E(\tilde{x},\tilde{u}) \leftarrow \lambda E(\tilde{x},\tilde{u})$ ▷ decay trace
15: **end for**
16: $x \leftarrow x'$
17: **end loop**

"Accumulating" refers to the fact that the trace is not reset to 1, but incremented by 1 in line 11. Line 7 resets the trace whenever an exploratory action is taken, because at that step the causality in the agent's path is interrupted. Credit should not be assigned to actions preceding the exploratory move, as their influence was cut at that moment. A version of $Q(\lambda)$ that avoids this issue is given by Peng and Williams (1996).

Another model-free method, related to the above but more complex and heuristic in nature, is the learning classifier system. We introduce it here as it is used by some of the work surveyed later in the chapter.

A learning classifier system is a "parallel, message-passing, rule-based systems designed to permit nontrivial modifications and reorganizations of its knowledge as it performs a task" (Booker, 1988). A population of classifier rules represents the knowledge of the classifier system. Each of the classifiers consists of a condition part and an message part. The condition part is in the basic variant of the learning classifier system a bit string where each bit can take values in the set $\{0, 1, \#\}$, the latter with the meaning of "don't care". The behaviour of the classifier system is given by three subsystems: performance, credit assignment, and rule discovery. A messages list circulates information within the system.

At each iteration, the environmental state is processed via an input interface to produce a set of binary messages, that are placed on the messages list. The messages on the list are matched with the condition input strings of the classifier population. The performance module then probabilistically chooses a set of classifiers for activation, based on their matching degree with the messages and their strength. The strength measures the overall value of the classifier for the system.

The classifiers chosen for activation generate their corresponding messages, which are placed on a new messages list that replaces the old one. The messages list is processed via an output interface to produce the actions of the learning classifier system. Some of the messages may remain on the list after processing and be fed back to the classifiers directly without passing through the environment, thus endowing the system with memory.

The credit assignment system, in the "bucket brigade" version, works in the following way: each classifier bids a fraction of its strength for the purpose of activation. If it is activated, it pays this bid to the classifiers that were active at the previous time step, and it receives the bids of the classifiers that are activated at the next time step. Direct reward from the environment is distributed to the classifiers that were activated prior to its receival.

The rule discovery system is responsible with generating new classifiers that should enhance the performance of the system, typically by genetic algorithms.

The credit assignment system is similar to the temporal difference update rule (2.17). The classifier strengths are similar to action values, with classifiers encoding relations between states (processed by the input interface to obtain messages) and actions (obtained from messages by the output interface). Bids play the role of value increments, with rewards being used in the updates as well when they are received from the environment.

An example will clarify. For the very simple classifier system introduced by (Dorigo and Bersini, 1994), where messages are identified with overt actions, a classifier exists for each state-action combination, and exactly one classifier is active at each time step, the bucket brigade strength redistribution reduces to:

$$S(c_k, m_k) \leftarrow S(c_k, m_k) + r_{k+1} + \alpha \left[S(c_{k+1}, m_{k+1}) - S(c_k, m_k) \right], \tag{2.24}$$

where c and m denote respectively the condition and message parts, and the time indices implicity point to the corresponding active classifiers. The relation clearly bears a strong resemblance to (2.17). In fact, (Dorigo and Bersini, 1994) argued that, with certain modifications, the very simple classifier system is equivalent to Q-learning.

The general version of the learning classifier system has some important advantages over temporal difference learning. The presence of "don't care" symbols offers generalization abilities. The system natively incorporates memory via direct message passing from one iteration to another, and memory helps in situations where the learning task does not satisfy the Markov property. Learning classifier systems also natively perform structural adaptation, via the rule discovery mechanism. However, due to their complexity, a mathematical formulation

of the learning classifier systems that could provide theoretical convergence and optimality guarantees has not yet been found.

Mixing model-based with model-free methods

Sutton (1991) introduced the Dyna architecture (from Dynamic programming). This architecture uses experience in a much more efficient way than model-free methods. Besides using the conventional Q-learning rule (2.18) to perform model-free updates, Dyna constructs a model of the environment from experience and uses it to perform model-based updates in between interactions with the real world. Thus, Dyna combines Q-learning with value iteration ideas, resulting in less experiences in the real world necessary to reach a good behaviour. The computational cost of Dyna is, however, greater than that incurred by model-free methods. As such, it should be used when computation is cheap and interaction with the real world is expensive.

The model-based updates of Dyna are performed on randomly chosen state-action pairs. A further improvement can be obtained if the state-action pairs are queued in decreasing order of the impact their update may have on their predecessors. At each iteration, a few state-action pairs are popped from the top of the queue and updated. From the sweeps performed by value iteration through the state space, this algorithm is called prioritized sweeping (Moore and Atkeson, 1993).

Direct policy search

Another important class of methods for solving RL problems is direct search in the policy space. These methods do not use the dynamic programming machinery introduced above, relying instead on finding an appropriate policy directly. This is typically done by gradient ascent or genetic algorithms.

The main advantage is, of course, greater generality. A disadvantage is that policy search takes more time than the classical methods. Gradient methods were introduced that converge in situations where dynamic programming was shown to diverge due to violation of its convergence assumptions (Baird, 1995; Baird and Moore, 1998). On the other hand, gradient methods can be shown to converge only to local optima, which do not necessarily give good policies. The locality problem can be alleviated by the use of global optimization techniques such as genetic algorithms (see Section 3.1 for a brief description), but the complexity of this method makes theoretical results very difficult. Also, genetic algorithms are more computationally intensive than gradient methods.

2.3 The multiagent case

The RL field is a mature one, with well understood theoretical results and proven practical applications. Due to this, and to the relaxed assumptions it makes on the learning task, RL seems a very attractive solution to the multiagent learning problem.

The extension from single-agent RL to multiagent reinforcement learning (MA-RL) is, however, not trivial. The main difficulty comes from the fact that, from the viewpoint of any agent in the MAS, the environment is not any longer Markovian. This is because the other agents are part of this environment, and each agent is a dynamic system that changes its behaviour as it learns (as explained in Section 2.2.1 and exemplified for SARSA and Q-learning

in Section 2.2.6). It is not, therefore, sufficient to know the state of the environment to reason, even in principle, on how it will evolve. Stated differently, the difficulty is that the results of an agent's action depend not only on the state of the environment when the action is taken, but also on the actions of the other agents performed at the same time.

This violation of the Markov assumption destroys the theoretical convergence guarantees of single-agent RL. In order to achieve results, an agent needs to reason explicitly on the behaviour of the other agents as it learns – their action choices and the mechanisms that led to those choices. This is, unfortunately, not an easy thing to do, and it will take the rest of this chapter and the next one to present the difficulties that arise and their explored solutions.

The Markov assumption violation is not the only problem of MA-RL. Another important problem is the exponential explosion of the state space in the number of agents. This renders simple, tabular representations of value functions completely impractical for all but the simplest of problems. The credit assignment problem is also more difficult in MAS than for a single agent. This follows from the interdependence between the agents' actions.

There are also benefits to agents learning in a MAS. These come mainly from knowledge sharing. For instance, if several agents learn to perform similar tasks, they can share their experience to speed up learning (Tan, 1993). Or, when a new agent arrives, "older", more skilled agents may serve as teachers (Clouse, 1995). If teaching is not desirable, the newcomer might learn by watching and imitating the skilled agents as they perform their tasks (Price and Boutilier, 2003).

The behaviour of multiple rational agents in interaction has been extensively studied in game theory. In consequence, much of the research in MA-RL is based on game theoretic notions. In fact, the influence of game theory extends beyond MA-RL to general multiagent learning and coordination methods.

In this section, we present the formal model of the MA-RL task, detailing its stateless version which bears relevance for game-theoretic techniques. We then review the controversy in the literature regarding the MA-RL learning goal. In the following sections, we review the solutions to the MA-RL task explored by the literature. We start with the direct application of single-agent RL, continue with special cases of the multiagent setting (fully cooperative, followed by stateless MAS), and then present techniques for dealing with the general MA-RL case.

2.3.1 Formal model

Many of the notions introduced below are denoted by several names in the literature. When this is the case, we give the alternatives in parentheses after the chosen notion name.

The concept of Markov game stands at the basis of most MA-RL task models. Prior to defining a Markov game, we define its stateless version, the strategic game.

Definition 2.5 *A* strategic game *(matrix game) is a tuple* $\langle A, \{U_i\}_{i\in A}, \{\rho_i\}_{i\in A}\rangle$ *where* A *is the set of agents,* $|A| = n$ *being their number,* $\{U_i\}_{i\in A}$ *are the discrete sets of actions available to each agent, yielding the joint action set* $\boldsymbol{U} = \times_{i\in A} U_i$*, and* $\overline{\rho}_i : \boldsymbol{U} \to \mathbb{R}, i \in A$*, are the reward functions of the agents.*

Agents take actions as indicated by their *strategies* $\sigma_i : U_i \to [0, 1]$, and receive rewards on the basis of the joint action $\boldsymbol{u} = [u_1, \ldots, u_n]^{\mathrm{T}}$: $r_i = \overline{\rho}_i(\boldsymbol{u})$. Strategies can be stochastic (also called "randomized") or deterministic (also called "pure"). We denote the joint strategy

of the agents by $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$, $\boldsymbol{\sigma} \in \Pi(\boldsymbol{U})$, where $\Pi(\cdot)$ denotes the space of probability distributions over the set given as argument.

Any reward function $\overline{\rho}_i$ can be written as an n-dimensional matrix with discrete actions as indices and reward values as content – hence the alias "matrix-game". We sometimes refer to rewards in strategic games by the game-theoretic term "payoff".

Some problems in multiagent learning and coordination are modeled by *repeated* strategic games. They are strategic games that are played repeatedly by the same agents. These agents incrementally learn how to solve the strategic game.

Definition 2.6 *A* Markov game *(stochastic game) is a tuple* $\langle A, X, \{U_i\}_{i \in A}, f, \{\rho_i\}_{i \in A} \rangle$ *where:*

- A *is the set of agents,* $|A| = n$ *being their number.*
- X *is the discrete set of states.*
- $\{U_i\}_{i \in A}$ *are the discrete sets of actions available to each agent, yielding the joint action set* $\boldsymbol{U} = \times_{i \in A} U_i$.
- $f : X \times \boldsymbol{U} \times X \to [0, 1]$ *is the state transition probability distribution.*
- $\rho_i : X \times \boldsymbol{U} \times X \times \mathbb{R} \to [0, 1], i \in A$ *are the reward probability distributions of the agents.*

At each time step k, each agent i observes the state x_k and takes action $u_{i,k}$ as indicated by its policy $h_i : X \times U_i \to [0, 1]$. As a result of the joint agent action $\boldsymbol{u}_k = [u_{1,k}, \ldots, u_{n,k}]^{\mathrm{T}}$, the world changes state to x_{k+1} with probability $P(x_{k+1} \,|\, x_k, \boldsymbol{u}_k) = f(x_k, \boldsymbol{u}_k, x_{k+1})$ and each agent i receives a reward $r_{i,k+1}$ with probability $P(r_{i,k+1} \,|\, x_k, \boldsymbol{u}_k, x_{k+1}) = \rho_i(x_k, \boldsymbol{u}_k, x_{k+1}, r_{i,k+1})$. Similarly, policies can be stochastic or deterministic (also called "pure"). A "stationary" policy is one that does not change its action selection probabilities over time. We denote the joint policy of the agents by $\boldsymbol{h} = (h_1, \ldots, h_n)$.

The Markov game is an extension of the strategic game to multiple states, and also to stochastic rewards. Each state of the Markov game is a different strategic game with stochastic rewards played by the same agents A. In fact, many multiagent reinforcement learning algorithms deal with stochastic games by separately solving the strategic games that arise in each state of the stochastic game.

Similarly, a policy is an extension of a strategy to multiple states, and conversely, a strategy is the reduction of a policy to a single state.

A Markov game is an extension of an MDP; conversely, an MDP is a Markov game with $n = 1$.

The Markov game is also a special case of a DMAS (Definition 1.2). A similar argument to that presented in Section 2.2.1 holds: the DMAS environment is fully measurable, and the rewards are included in the agents' observations. The correspondences between the Markov game and the DMAS elements are summarized in Table 2.2.

We sometimes use the game-theoretic term "play" to denote the evolution of an agent within a Markov game. If the agents are homogeneous (Section 1.3), i.e., they use the same learning algorithm, we call the learning process taking place in the Markov game *self-play*. Homogeneity is equivalent to identical learning algorithms because the agents already have a common perspective, due to the complete measurability assumed by the Markov game. Self-play is typically defined by the literature in the context of (repeated) strategic games, but the definition above includes those as special cases.

Markov game	DMAS
state space X	environment state space X
action space U_i	action space U_i
transition distribution f	transition distribution f
reward distribution ρ_i	(implicit in) the observation distribution ω_i
policy h_i	policy h_i

Table 2.2: Correspondences between the Markov game and the DMAS elements

We are particularly interested in two special cases of Markov games: when the agents have a common goal (i.e., the same reward functions), and when two agents act one against the other. We use deterministic reward functions following the literature, but the extension to stochastic rewards is trivial.

Definition 2.7 *A* multiagent Markov decision process (MMDP) *(fully cooperative game, collaborative multiagent MDP, team Markov game) is a Markov game* $\langle A, X, \{U_i\}_{i\in A}, f, \overline{\rho}\rangle$*, where all agents share the same reward function* $\overline{\rho}$*.*

Definition 2.8 *A* fully competitive game *(zero-sum game) is two-players Markov game* $\langle \{a_1, a_2\}, X, U_1, U_2, f, \overline{\rho}_1, \overline{\rho}_2\rangle$*, where* $\overline{\rho}_1(x, u_1, u_2) = -\overline{\rho}_2(x, u_1, u_2), \forall x \in X, u_1 \in U_2, u_2 \in U_2$*.*

The name "multiagent Markov decision process" is justified as follows: if all the agents are considered together as a centralized decision maker (the control is centralized – see Section 1.3), then the MMDP reduces to an MDP with an action space given by the joint action space of the MMDP. A significant part of the work on multiagent learning and coordination focuses on this type of Markov game.

Kok et al. (2005b) use a slightly different version of multiagent Markov decision process:

Definition 2.9 *A multiagent Markov decision process is a stochastic game* $\langle A, X, \{U_i\}_{i\in A}, f, \{\overline{\rho}_i\}_{i\in A}, \overline{\rho}\rangle$*, where* $\overline{\rho}$ *is the global reward function and is given by* $\overline{\rho}(x, \boldsymbol{u}) = \sum_{i\in A} \overline{\rho}_i(x, \boldsymbol{u}), \forall x \in X, \boldsymbol{u} \in \boldsymbol{U}$*.*

The agents in this version of MMDP attempt to maximize the global return at each step, so this definition is equivalent to Definition 2.7.

The fully competitive game is also called "zero-sum" because the rewards of the two agents always sum to 0. When discussing fully competitive games, and also two-agent games in general, from the perspective of one agent, we sometimes refer to the other agent by the game-theoretic term "opponent", and to the agent whose perspective we assume as the "player".

2.3.2 Solution concepts

Currently, there does not exist a consensus in the literature as to the learning goal of MA-RL agents. We summarize the controversy developed around this issue in Section 2.3.3. For now, we simply present the building blocks used in defining the various learning goals proposed by researchers and reviewed in this chapter. Many of these building blocks are game-theoretic equilibria.

The basic concept standing at the basis of MA-RL solution concepts is, similarly to single-agent RL, the *value function*. The state and action value functions are defined similarly to the

single-agent case, but taking into account the policies (and, for the action value function, the actions) of all the agents. We give the definitions directly in terms of the expected discounted return (2.2).

Definition 2.10 *The value of a state x for agent i under a joint policy $\boldsymbol{h}$ is the expected return of agent i when the environment starts in x and the agents follow $\boldsymbol{h}$ thereafter:*

$$V_i^{\boldsymbol{h}}(x) = E_{\boldsymbol{h}}\left\{\sum_{l=0}^{\infty}\gamma^l r_{i,k+l+1}\,\middle|\, x_k = x\right\}. \tag{2.25}$$

Definition 2.11 *The value of the joint action $\boldsymbol{u}$ in state x for agent i under a joint policy $\boldsymbol{h}$ is the expected return when the environment starts in x, the agents take $\boldsymbol{u}$ and follow $\boldsymbol{h}$ thereafter:*

$$Q_i^{\boldsymbol{h}}(x,\boldsymbol{u}) = E_{\boldsymbol{h}}\left\{\sum_{l=0}^{\infty}\gamma^l r_{i,k+l+1}\,\middle|\, x_k = x, \boldsymbol{u}_k = \boldsymbol{u}\right\}. \tag{2.26}$$

As noted in Section 2.3.1, many multiagent learning algorithms work by solving the strategic games given by each state of the Markov game. We therefore give the formalization below in terms of stateless games. The state values under joint policies $V_i^{\boldsymbol{h}}(x)$ (i.e., expected returns) become then, simply, values under joint strategies $V_i^{\boldsymbol{\sigma}}$ (i.e., expected rewards), as the concept of state loses its meaning:

$$V_i^{\boldsymbol{\sigma}} = E_{\boldsymbol{\sigma}}\left\{r_i\right\}. \tag{2.27}$$

The extension to the multi-state case is formally straightforward: simply solve for the joint action in state values $V_i^{\boldsymbol{h}}(x)$ instead of strategy values $V_i^{\boldsymbol{\sigma}}$.

The joint strategy of the agents $\boldsymbol{\sigma}$ is also known as a "strategy profile". We denote the joint strategy of all the agents except agent i (the "reduced strategy profile") by $\boldsymbol{\sigma}_{-i} = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_n)$. We say σ_i^* is a "best response" of agent i to the reduced strategy profile $\boldsymbol{\sigma}_{-i}$ if $V_i^{(\sigma_i, \boldsymbol{\sigma}_{-i})} \leq V_i^{(\sigma_i^*, \boldsymbol{\sigma}_{-i})}, \forall \sigma_i \in \Pi(U_i)$.

The most widely used game-theoretic solution concept for multiagent learning is the Nash equilibrium.

Definition 2.12 *A* Nash equilibrium *is a joint strategy $\boldsymbol{\sigma}^*$ such that $\forall i \in A$, σ_i^* is a best response for $\boldsymbol{\sigma}_{-i}^*$.*

In fully competitive games, the Nash equilibrium takes a special meaning: each strategy σ_1 is evaluated with respect to the opponent strategy σ_2 that, in combination with σ_1, yields the least value. The agent will behave so that it maximizes its payoff in the worst case:

$$\sigma_1^* = \arg\max_{\sigma_1 \in \Pi(U_1)} \min_{\sigma_2 \in \Pi(U_2)} V_1^{(\sigma_1, \sigma_2)}. \tag{2.28}$$

This principle is called minimax (Littman, 2001b).

It is unknown whether Nash equilibria can be computed in polynomial time. The strategies forming a Nash equilibrium are uncorrelated, in the sense that the probability distributions $\sigma_i,\ i \in A$ are mutually independent. By removing the independence requirement, a more general class of equilibria called *correlated equilibria* is obtained. The set of correlated equilibria

is convex and contains the set of Nash equilibria. An advantage is that, being convex, the set of correlated equilibria is computable in polynomial time by linear programming.

In their general definition, Markov games are symmetric: no agents are favoured over others. If we allow some agents ("leaders") to possess information on how other agents ("followers") will act, then a new notion of equilibrium arises. We restrict the definition to two agents, for simplicity (Başar, 1985).

Definition 2.13 *A pair of strategies* (σ_1^*, σ_2^*) *is a* Stackelberg equilibrium *(leader-follower equilibrium) with unique follower responses of the two-players strategic game* $\langle \{a_1, a_2\}, U_1, U_2, \overline{\rho}_1, \overline{\rho}_2 \rangle$ *if there exists a unique mapping* $T : \Pi(U_1) \to \Pi(U_2)$ *satisfying:*

$$V_2^{(\sigma_1, T(\sigma_1))} \geq V_2^{(\sigma_1, \sigma_2)}, \quad \forall \sigma_1 \in \Pi(U_1), \sigma_2 \in \Pi(U_2) \tag{2.29}$$

$$V_1^{(\sigma_1^*, T(\sigma_1^*))} \geq V_1^{(\sigma_1, T(\sigma_1))}, \quad \forall \sigma_1 \in \Pi(U_1) \tag{2.30}$$

with $\sigma_2^* = T(\sigma_1^*)$.

Agent 1 is the leader, agent 2 the follower, and the mapping T describes how the follower reacts to the actions of the leader. Condition (2.29) ensures that the rational follower agent will obey the mapping T; (2.30) is the equilibrium condition.

Another type of solution concept is *regret*. Regret measures the difference between the maximum total reward that could have been achieved by any fixed, deterministic policy, and the actual reward obtained by the agent:

$$\mathcal{R}_{i,k} = \max_{\overline{h}_i} \sum_{l=0}^{k-1} \left[\rho_i(x_l, \overline{h}_i(x_l)) - r_{i,l+1} \right], \tag{2.31}$$

where we assumed deterministic rewards to keep the notation simple.

A good agent performance is associated with low (or negative) regret.

2.3.3 Learning goal

Many MA-RL algorithms set as goal for the learning agent, convergence to a game-theoretic equilibrium, most often the Nash equilibrium (e.g., Littman, 2001b; Hu and Wellman, 2003; Greenwald and Hall, 2003). An explanation for this is that the Nash equilibrium is self-enforcing – i.e., seems to be the "natural" long-term consequence of the best-response play of rational agents.

However, Shoham et al. (2003) put forth a tough criticism on the focus on game-theoretic equilibria in general, and on the Nash equilibrium in special. They argued that the Nash equilibrium presents important problems:

- In general, a strategic game can have multiple Nash equilibria. This leads to awkward convergence guarantees, requiring that the agents somehow coordinate their selection among these equilibria, perhaps by an external mechanism.

- The meaning and desirability of the Nash equilibrium, defined in terms of stateless games, are doubtful in the full Markov game setting, where delayed reward has an important role.

It is important to note that these objections extend to other equilibrium notions, as well. The problem of the agents consistently selecting the same equilibrium when multiple equilibria exist in a state of the Markov game is a recurring theme in MA-RL, and is known as *the equilibrium selection problem.*

Shoham et al. (2003) interpreted this unjustified focus on Nash equilibria as a symptom of a missing clearly defined problem statement in the MA-RL field.

A first attempt at a problem definition had already been done by Bowling and Veloso (2002), in their definition of the *rationality* and *convergence* criteria. A learning algorithm is "rational" if, given that all other agents converge to stationary policies, the learning agent converges to a policy that is a best response to those stationary policies. A learning algorithm is "convergent" if, given that the other agents use learning algorithms from a given set, the learning agent will necessarily converge to a stationary policy. In the authors' interpretation, it is desirable that learning algorithms are both rational and convergent.

However, Powers and Shoham (2004) argued that these criteria are dissatisfying, for the following reasons:

- They unjustifiably require that both the learner and the other agents converge to stationary policies, whereas non-stationary policies might very well be interesting for the designer of the multiagent system.
- Both rationality and convergence are required to hold in the limit, without offering any guarantees of a reasonable performance in finite time.

Moreover, the authors argued, both properties are defined in terms of the policy of the agent, rather than of its actual performance measure, the reward; they address the effect instead of going directly to the cause.

The authors then defined three new criteria as a learning goal for MA-RL algorithms. In terms of repeated games, the learning agent must, with arbitrarily high probability and in finite time, achieve the following:

- *Targeted optimality*: when the learning algorithms of other agents are in a given set, an average reward that is arbitrarily close to the best-response value.
- *Compatibility*: in self-play, an average reward that is arbitrarily close to the value of the "best"[1] Nash equilibrium.
- *Safety*: when the other agents use any other learning algorithms, an average reward that is arbitrarily close to the minimax value (worst-case value).

These criteria completely drop the convergence requirements from the learning goal. They do, however, have a shortcoming: they only impose requirements on the *average* reward. This means that, at any given moment in time, the performance of the agent may be arbitrarily poor.

In order to avoid this shortcoming, Bowling (2004) introduced the requirement of no regret: if an agent does not converge to a stationary policy, then its regret (2.31) should be negative or zero. Convergence was still deemed a desirable characteristic of the learning algorithm, because it leads to the stability of the Markov game evolution, and stability allows for accuracy

[1]The technical statement of this condition is slightly more complicated; see (Powers and Shoham, 2004) for its exact form.

in estimating value functions. When delayed reward is involved, from a practical point of view the ability of properly estimating value functions is critical for the RL agent.

It seems, therefore, that converging and achieving good rewards are not two mutually exclusive goals. Instead, a tradeoff between these two properties may exist. In order to obtain high rewards, an agent needs to predict the value function accurately; but to do that, stability is required, and thus the agent's learning process must convergence, sacrificing reward!

In any case, the debate on the issue of the multiagent reinforcement learning goal is still open, and no definitive answers have been given yet.

2.4 Application of single agent techniques to multiagent reinforcement learning

The easiest way of dealing with the consequences of the presence of other agents in the RL task is, of course, to disregard them. Some good results have been reported with this approach in certain problems ranging from simple simulations to real, complex tasks.

The work of Sen et al. (1994) lies at the first end of the spectrum. In this study, two agents using Q-learning learned complementary policies for pushing a block on a two-dimensional surface. The authors used a very specific type of problem where the goal state had the same horizontal coordinate with the start state. They were thus able to quantify the position information in vertical "stripes" and to provide instantaneous reinforcement to the agents on the basis of the horizontal distance between the block and the goal position. This particularity simplified the RL problem in two ways: first, the number of dimensions of the state space was halved (from two to one), and second, the problem of delayed reward was eliminated.

The specificity of this learning problem does not allow us to conclude whether the approach would extend to general settings. This issue is representative for the work with single-agent learning in simple multiagent simulations.

A large part of the body of work in complex settings comes from the field of multirobot systems. E.g., Matarić (1996) presented results on a multirobot foraging task, where the robots learned to collect pucks scattered in the world and bring them to a designated home region. The state and action space of the agents were highly abstracted, such that they only needed to learn a mapping from a small number of high-level conditions (such as *have-puck*) to a small number of high-level behaviours (such as *homing*). The reinforcement signal was composed in a complex way from several separate goals and instantaneous parts called progress estimators (e.g., an intruder avoidance estimator rewarded an agent for maintaining an appropriate distance from its teammates).

Another representative work is the simulated robotic soccer application of a fuzzy learning classifier system by Bonarini and Trianni (2001). In this work state information was abstracted as fuzzy linguistic variables. For example, the agent knows whether there are zero, one, or more teammates in his vicinity, with the closest being close to the right and the farthest far to the left (with "zero", "one", "more", "close", "far", "left", "right" linguistic values). This information is repeated for agents from the opponent team. Similar fuzzy values were defined for the ball and the internal agent state (containing for instance stamina). The latter is in the view of our DMAS model only a part of the internal agent state, which also contains learned knowledge. Other complex tools such as heterogeneous reinforcement with progress estimators and high-level behaviours were used.

Crites and Barto (1998) applied reinforcement learning to the task of elevator scheduling.

The model of this task is a discrete event system with continuous time. Though the reinforcement signal is global, yielding a common-goal task, different RL controllers are allocated to each elevator, so the system is indeed a DMAS. Further complexities arise from the partial observability of the environment (some parts of the state are hidden, e.g., the destinations of the passengers waiting at each floor). The researchers used returns defined as continuous time integrals and neural networks to represent the value function. The results obtained in simulations outperformed commercial scheduling algorithms.

It is characteristic of these realistic multiagent applications to make use of empirical and heuristic machinery in order to make the complex considered problems learnable. This machinery makes difficult the analysis of the techniques and their usefulness in other settings.

2.5 Fully cooperative multiagent teams

The theoretical model that describes the fully cooperative setting is the multiagent Markov decision process (Definition 2.7). In an MMDP, the optimal solution can in principle be found by treating the multiagent system as a single agent, as noted in Section 2.3.1, and by estimating the optimal joint action values with Q-learning. For n agents, a Q-table of the form $Q(x, u_1, \ldots, u_n)$ is learned. This approach then yields the optimal policy by simple greedy action choice.

The problem is, of course, that in a multiagent system the agents always have a certain degree of autonomy in choosing their actions (if the MAS is decentralized, they have complete autonomy). A solution might be to duplicate the value function and the learning algorithm in each agent. This is possible as long as the actions are measurable among the agents. The algorithm relying on this solution has been called "team-Q" or "friend-Q" in the literature (Littman, 2001b). There is one further difficulty coming from the distributed nature of the decision making process. For some world state, there might be several joint actions yielding the best Q-value. The classical greedy action choice would break ties randomly. However, if each agent breaks the tie randomly, different agents may break the tie in different ways. This would lead the agents to select different joint actions and execute their part, with the actual resulting joint action being suboptimal! This is a special case of the equilibrium selection problem.

In fact, Littman (2001b) offered team-Q convergence guarantees only for the value function: the Q-functions maintained by the agents will converge to the optimal Q^*. This guarantee is complemented by convergence of policies only when the optimal joint actions achieving Q^* (termed in his work "coordination equilibria") are unique in every state.

In summary, team-Q requires the following assumptions:

(i) the Markov game is fully cooperative (it is an MMDP).

(ii) the actions are measurable among the agents.

(iii) the optimal joint actions are unique in every state of the world

Lauer and Riedmiller (2000) gave an algorithm they call "distributed Q-learning" that removes assumption (ii). An agent i maintains a Q-table indexed only on its own action – $Q_i(x, u_i)$, and uses a modified temporal difference update rule:

$$Q_i(x_k, u_{i,k}) \leftarrow \max \left\{ Q_i(x_k, u_{i,k}),\ r_{k+1} + \gamma \max_{u'_i \in U_i} Q_i(x_{k+1}, u'_i) \right\}. \tag{2.32}$$

This update rule moves the estimate $Q_i(x_k, u_{i,k})$ all the way to the new estimate $[r_{k+1} + \gamma \max_{u_i' \in U_i} Q_i(x_{k+1}, u_i')]$ (in contrast with (2.18), that moves it only a fraction α of the way), but only if the update leads to an increase in the value of the estimate.

The authors proved that with this update rule, under the assumptions that the reward function is positive and all Q-values are initialized to 0, the Q-values the agents learn are the maxima of the joint Q-values:

$$Q_i(x, u_i) = \max_{\substack{\boldsymbol{u}=[u_1,\ldots,u_n]^{\mathrm{T}} \\ u_j \in U_j, j \neq i}} Q(x, \boldsymbol{u}), \quad \forall x \in X,\ u_i \in U_i. \tag{2.33}$$

Though requiring the reward function to be positive seems strange, it actually does not restrict the generality of the algorithm. This is because in RL agents always act on the basis of relative differences in the value estimates, and not on the basis of the absolute values of these estimates: an agent compares the Q-values of its actions and takes the action with the highest Q-value.

The agents maintain explicit estimates $\tilde{h}_i$ of the optimal policy. By modifying the greedy policy so that modifications are allowed only when they lead to improvements in the Q-values:

$$\tilde{h}_i(x_k) \leftarrow \begin{cases} \tilde{h}_i(x_k) & \text{if } \max\limits_{u_i \in U_i} Q_i(x_k, u_i) \text{ has not changed by (2.32)} \\ u_{i,k} & \text{otherwise} \end{cases} \tag{2.34}$$

the authors showed that the joint policy $\tilde{\boldsymbol{h}} = (\tilde{h}_1, \ldots, \tilde{h}_n)$ always attains the greedy value with respect to the joint Q-table. In (2.34), $u_{i,k}$ is the action actually taken by the agent at time step k. Since distributed Q-learning is off-policy and the agents need to follow exploratory policies, this action need not always be taken according to $\tilde{h}_i$, and learning does take place in (2.34).

Using the greediness of the joint policy with respect to the joint Q-table, the authors show that the convergence of distributed Q-learning follows from that of basic Q-learning.

2.6 General multiagent systems

The full cooperation assumption is in many cases too restrictive. This is true especially for self-interested agents, but even cooperating teams of agents may sometimes encounter situations where the immediate interests of some of the agents are in conflict. An example is a shared resource: though the high-level goals of the agents might be the same, if they all need at some point a resource that is too expensive to be multiplied for each agent, they will compete to obtain that resource. The theoretical model describing this type of tasks is the Markov game (Definition 2.6).

Algorithms for handling such cases typically make several assumptions:

(i) The actions are measurable among the agents.

(ii) The rewards are measurable among the agents.

Sometimes, algorithms make additional assumptions. We state these assumptions explicitly wherever such is the case.

Most of the times the agents learn value functions on the basis of the joint action and use stochastic policies. The latter is motivated by the existence of Markov games for which the targeted solutions cannot be expressed by deterministic policies.

Many approaches dealing with general multiagent systems come from the field of game theory.

2.6.1 Stateless problems

There exist MA-RL approaches stemming from game theory that handle only repeated, strategic games (Definition 2.5). As such, they do not possess the concept of state, and lose one of the main characteristics of RL: delayed reward. Nevertheless, these approaches are relevant for our discussion on MA-RL, and one of them (infinitesimal gradient ascent) forms the basis for a more general method we review in Section 2.7.

All the algorithms presented in this section strengthen assumption (ii) to:

(iii) The reward functions of the agents are common knowledge (formally, common knowledge means that all agents know the fact, all the agents know that all the agents know the fact, and so on indefinitely).

They also add an extra assumption:

(iv) The agents play a repeated strategic game (no world states).

Moreover, almost all such approaches are designed for games with only two agents.

Littman and Stone (2001) focused on two agent repeated games and introduced two aggressive "leader" strategies that attempt to induce follower behaviour in the opponent. By using these strategies, the agent tries to create an asymmetric, Stackelberg situation in an originally symmetric game (see Section 2.3.2). The authors' motivation was that the best-responses usually considered in game-theoretic approaches to multiagent learning are essentially follower strategies, and that important benefits might be achieved by employing aggressive leader-like strategies that implicitly induce cooperation in the opponent.

The two strategies are called "Bully" and "Godfather", and rely respectively on stubbornness and threats. We look at the game from the perspective of the first agent. Bully is a deterministic strategy choosing its action by:

$$u_{1,k} = \arg \max_{u_1 \in U_1} \overline{\rho}_1(u_1, \arg \max_{u_2 \in U_2} \overline{\rho}_2(u_1, u_2)), \quad \forall k \geq 0. \tag{2.35}$$

So, Bully assumes the opponent will play a best response to its strategy and plays so that it will maximize its reward under this assumption.

The Godfather strategy chooses a joint action that yields to both agents more than their minimax ("security-level") payoffs. It then executes its part of this joint action. If the opponent doesn't execute its corresponding component, Godfather takes the action that will reduce the payoff of the opponent to its security level, basically telling it "play your half, or no matter what you do you will not obtain more than your security level". Formally, if $[u_1^*, u_2^*]^{\mathrm{T}}$ is the joint action chosen by Godfather:

$$u_{1,k} = \begin{cases} u_1^* & \text{if } k = 0 \text{ or } u_{2,k-1} = u_2^* \\ \arg \min_{u_1 \in U_1} \max_{u_2 \in U_2} \overline{\rho}_2(u_1, u_2) & \text{otherwise} \end{cases} \tag{2.36}$$

The authors tested the leader strategies in several repeated games showing they work better than best-response strategies when paired with follower opponents.

Conitzer and Sandholm (2003) targeted the rationality and convergence learning goals defined by Bowling and Veloso (2002) (Section 2.3.3). The algorithm presented by the authors is called AWESOME, from "adapt when everybody is stationary, otherwise move to equilibrium" . The agent switches between the equilibrium and best-response strategies on the basis of its experience. It acts according to its beliefs in two hypotheses: (i) that all other agents are stationary, and (ii) that all other agents play their part of a precomputed equilibrium. The beliefs are updated on the basis of the analysis of changes in empirical distributions of the other agents' actions over consecutive epochs of learning iterations.

The method was shown to (asymptotically) learn a best response against stationary agents and converge in self-play for all repeated games.

In addition to defining targeted optimality, compatibility and safety (Section 2.3.3), Powers and Shoham (2004) also gave an algorithm that according to the authors provably meets the three requirements. The algorithm was thoroughly tested on stateless games against a broad set of reinforcement learning techniques, and fared well against all of them in asymptotic performance.

An important result for our forthcoming investigation on adaptive MA-RL falls in the area of direct policy search. Singh et al. (2000b) focused on gradient ascent incremental policy update in two-agents, two-actions, repeated strategic games. In such games, the stochastic strategy of an agent can be modeled by a real number in the interval $[0, 1]$. Say, the strategy of agent 1 is represented by $\alpha \in [0, 1]$, then the agent chooses its first action with probability α and its second action with probability $1 - \alpha$. Similarly, the second agent's stochastic strategy is described by a parameter $\beta \in [0, 1]$. The gradient ascent update rule is then:

$$\begin{cases} \alpha_{k+1} = \alpha_k + \delta \dfrac{\partial V_1(\alpha_k, \beta_k)}{\partial \alpha} \\ \beta_{k+1} = \beta_k + \delta \dfrac{\partial V_2(\alpha_k, \beta_k)}{\partial \beta} \end{cases} \tag{2.37}$$

where $V_i(\alpha_k, \beta_k)$ is the expected payoff of agent i given that the agents play strategies α_k, β_k and δ is the gradient ascent step size.

The main result proven by the authors was that assuming an infinitesimal step size ($\lim_{\delta \to 0}$) the payoffs of the agents converge in the limit to the Nash payoffs.

2.6.2 Multiple-state problems

In order to be of practical interest, a MA-RL algorithm must handle problems with a non-empty state space and delayed reward, i.e., full-blown Markov games. Such algorithms typically make an extra requirement beside the assumptions (i) – (ii) listed in the beginning of Section 2.6:

(v) All the agents use the same learning algorithm (a self-play situation exists).

Many MA-RL algorithms for the Markov games are derived from Q-learning, and share a common skeleton. This skeleton is given in Algorithm 2.5 for one agent, identified as agent i.

The dots "$\cdot$" stand for "all values of the corresponding argument". Algorithm 2.5 is clearly similar to basic Q-learning in Algorithm 2.3. Several very important differences exist. We analyze them in turn.

Algorithm 2.5 Generic multiagent Q-learning for agent i

Input: learning rate α, discount factor γ

1: $Q_i(x, \boldsymbol{u}) \leftarrow 0, \quad \forall x \in X, \boldsymbol{u} \in \boldsymbol{U}$, where $\boldsymbol{U} = \times_{j \in A} U_j$
2: $h_i(x, u_i) = 1/\left|U_i\right|, \quad \forall u_i \in U_i$
3: observe initial state x
4: **loop**
5: $\quad h_i(x, \cdot) \leftarrow \mathbf{solve}_i\left(Q_1(x, \cdot), \ldots, Q_n(x, \cdot)\right)$
6: $\quad$ draw u_i according to $h_i(x, u_i)$
7: $\quad$ apply u_i, with suitable exploration
8: $\quad$ observe other actions $u_j, j \in A, j \neq i$, rewards $r_j, j \in A$ and next state x'
9: $\quad Q_i(x, \boldsymbol{u}) \leftarrow Q_i(x, \boldsymbol{u}) + \alpha\left[r_i + \gamma \cdot \mathbf{eval}_i\left(Q_1(x', \cdot), \ldots, Q_n(x', \cdot)\right) - Q_i(x, \boldsymbol{u})\right]$
10: $\quad$ **for** $j \in A, j \neq i$ **do**
11: $\quad\quad Q_j(x, \boldsymbol{u}) \leftarrow Q_j(x, \boldsymbol{u}) + \alpha\left[r_j + \gamma \cdot \mathbf{eval}_j\left(Q_1(x', \cdot), \ldots, Q_n(x', \cdot)\right) - Q_j(x, \boldsymbol{u})\right]$
12: $\quad$ **end for**
13: $\quad x \leftarrow x'$
14: **end loop**

Line 5 updates the policy of the agent in state x, $h_i(x, \cdot)$, with the strategy computed by $\mathbf{solve}_i$ from the Q-tables of all the agents. So, instead of the usual greedy policy, an agent maintains and uses an explicit stochastic policy computed on the basis of all the agents' Q-tables.

Line 9 is a temporal difference update, but the next state value used in the update target is not the value of the greedy action in that state, as in basic Q-learning. Instead, this value is given by an evaluation function $\mathbf{eval}_i$ that takes into account the Q-tables of the other agents.

Both steps described above need the Q-tables of all the agents in the MAS. So, an agent needs to maintain the Q-tables of the other agents along with its own Q-table. This is done by the *modeling* step in lines 10–12. The necessity of the measurable rewards (ii) and self-play (v) assumptions is obvious in line 11: the rewards of the other agents are needed to perform the update, and the other agents must use the same learning algorithm so that the update reflects the reality.

Line 7 also deserves some comments. This is where the agent acts using its stochastic policy. The action of the agent is applied to the environment simultaneously with the other agents' actions. The agent follows the strategy for state x computed by **solve**, but sometimes takes exploratory actions. It is important to note that exploration is still necessary, though in the general case the policies of the agents are stochastic. This is because nothing stops any of the strategies returned by **solve** to assign zero probability weight to some (or all except one) of the actions. A suitable exploratory policy, on the other hand, may never assign zero probability weight to any action, except perhaps in the limit.

Many times, **solve** searches for a certain type of game theoretic equilibrium (a joint strategy) for the strategic game given by state x' of the Markov game, and **eval** gives the value of this solution. The two functions are in such cases closely interrelated, and we may regard them as a single subroutine returning a strategy in the game state and the value of the state under that strategy. Moreover, in these cases the separate strategy and state value solutions of the agents should form a consistent joint solution, so the subroutine can be thought of as running identically and in parallel in all the agents, but returning to each agent its part of the solution. The equilibrium selection problem arises when the solution is not unique, and

in this context is the necessity of the **solve** and **eval** returning complementary parts of the same, consistent solution to all the agents.

Many multiagent learning algorithms can be understood as instantiations of the skeleton in Algorithm 2.5.

If the game is fully competitive (Definition 2.8), the minimax principle (2.28) can be applied (Littman, 2001b). In this case, the first agent (the player) chooses the strategy that maximizes its benefit under the assumption that the other agent (the opponent) will act so as to minimize this benefit. The value of the next state used in the update is also assumed to be the worst-case value. That is, from the perspective of the player and assuming deterministic action selection for the opponent:

$$\begin{cases} \mathbf{eval}_1\left(Q(x,\cdot)\right) = \max\limits_{h_1(x,\cdot)\in\Pi(U_1)} \min\limits_{u_2\in U_2} \sum\limits_{u_1\in U_1} h_1(x,u_1)Q(x,u_1,u_2) \\ \mathbf{solve}_1\left(Q(x,\cdot)\right) = \arg \max\limits_{h_1(x,\cdot)\in\Pi(U_1)} \min\limits_{u_2\in U_2} \sum\limits_{u_1\in U_1} h_1(x,u_1)Q(x,u_1,u_2) \end{cases} \tag{2.38}$$

Note that, due to the fact that $\rho_1 = -\rho_2$, $Q_1 = -Q_2$ and an agent needs to store a single Q-table, denoted in (2.38) simply by Q. Modeling the opponent's Q-table is not necessary.

The minimax optimization problem can be solved by a linear program. The algorithm converges to the minimax values of the fully competitive game and the resulting policy attains at least the learned values regardless of the opponent's policy. The author experimented with the algorithm on a simulated soccer game between two agents on a 5×4 two-dimensional gridworld (see Section 5.2).

The minimax algorithm can be extended to more than two agents by assuming that all other agents will act against the learner, and minimizing over their joint action instead of u_2 in (2.38). This situation was formalized by Littman (2001a).

The same work integrated minimax-Q and team-Q (Section 2.5) into a single algorithm called "friend-or-foe" Q-learning and presented the convergence conditions for this algorithm in the specific classes of Markov games where it does converge (Littman, 2001a). The name of the algorithm reflects the fact that some of the agents are designated "friends" (and maximization is consequently applied for their action terms in the value computation) and others "foes" (with minimax consequently applied for their action terms). We do not give the actual intricate formulae for computing the value and the policy here.

Both team Q-learning and friend-or-foe Q-learning are instantiations of the generic multiagent Q-learning algorithm. E.g., team-Q uses plain maximization for its **eval** function, and the deterministic action choice achieving this maximum value as **solve**. Since the game is fully cooperative, the Q-tables of the agents are identical and modeling is not required.

The Nash Q-learning algorithm works in a more general class of Markov games (Hu and Wellman, 2003). As its name suggests, this algorithm computes a Nash equilibrium in every state of the Markov game and uses it for the temporal difference and policy updates:

$$\begin{cases} \mathbf{eval}_i\left(Q_1(x,\cdot),\ldots,Q_n(x,\cdot)\right) = V_i^{\mathbf{NE}[Q_1(x,\cdot),\ldots,Q_n(x,\cdot)]}(x) \\ \mathbf{solve}_i\left(Q_1(x,\cdot),\ldots,Q_n(x,\cdot)\right) = \mathbf{NE}_i\left[Q_1(x,\cdot),\ldots,Q_n(x,\cdot)\right] \end{cases} \tag{2.39}$$

The expression $\mathbf{NE}_i$ represents the strategy component of the Nash equilibrium corresponding to agent i. We have slightly abused the notation to keep the formula simple; the value $V_i(x)$ should in fact be conditioned by the joint policy whose component in state x is the Nash equilibrium strategy.

The estimated value V_i of a state for an agent i under a joint stochastic policy is the summation of the agent's Q-tables weighted by the probability of the corresponding joint actions, assigned by the joint policy in that state:

$$V_i^{(h_1,\ldots,h_n)}(x) = \sum_{u_1 \in U_1, \ldots, u_n \in U_n} h_1(x, u_1) \ldots h_n(x, u_n)\, Q_i(x, u_1, \ldots, u_n). \tag{2.40}$$

In Nash Q-learning, each agent needs to model the Q-tables of all the others. The algorithm was proven to converge given that the agents consistently use the same Nash equilibria to compute **eval** in all states of the game. So, Nash Q-learning suffers from the equilibrium selection problem, along with the other issues described in Section 2.3.3.

By replacing the Nash equilibrium in (2.39) with a correlated equilibrium, we obtain correlated equilibrium Q-learning (Greenwald and Hall, 2003). We do not give the modified formulae, as they are identical to (2.39) in all respects except the type of equilibrium. The advantage is that a correlated equilibrium is computable via linear programming (see Section 2.3.2). The authors experimented with four types of equilibria, with the correlations between agent policies expressed in the linear programming objective function: utilitarian, maximizing the sum of the agents' rewards; egalitarian, maximizing the minimum of the agents' rewards; republican, maximizing the maximum of the agents' rewards; and libertarian, independently maximizing the maximum of each agent's reward. The experiments were run on a small 3×3 gridworld and on a 2×2 grid soccer game.

The authors circumvented the need of modeling the Q-tables of other agents and the equilibrium selection problem by centralizing the learning process. This is, of course, not a feasible solution if this method is to be applied in a realistic multiagent system, so the equilibrium selection problem remains intact.

Similarly, by using the Stackelberg equilibrium (see Section 2.3.2) in the updates we obtain asymmetric multiagent RL (Könönen, 2003). This algorithm addresses situations where leader agents are able to enforce action selections onto follower agents. The author experimented with a simple grid world obtaining marginally better results in the asymmetric case with respect to the symmetric one. Though the agents do not use identical algorithms, it is necessary that they follow their complementary parts in the asymmetric learning process, so, in a sense, the assumption of self-play is still necessary.

One variant of multiagent Q-learning that does give up the self-play assumption is the so-called "extended optimal response" algorithm (Suematsu and Hayashi, 2002). The algorithm was designed for games with two agents. We describe it from the first agent's perspective. The goal of the algorithm is to reach a Nash equilibrium when the second agent is learning, but exploit it using a best response when it exhibits stationary behaviour; this is the extended optimal response. The agent uses the following heuristic **solve** function:

$$\mathbf{solve}_1\left(Q_1(x,\cdot), Q_2(x,\cdot)\right) = \arg \max_{h_1(x,\cdot) \in \Pi(U_1)} \left[V_1^{(h_1, \widehat{h}_2)}(x) - \varsigma\, \varrho\left(x, Q_2(x,\cdot), h_1\right) \right], \tag{2.41}$$

where ς is a tuning parameter, $\widehat{h}_2$ is an estimate of the second agent's policy, and ϱ a distance function that approximates the increase in return that the second agent could achieve by changing its policy given the policy h_1:

$$\varrho\left(x, Q_2(x,\cdot), h_1\right) = \max_{h_2(x,\cdot) \in \Pi(U_2)} V_2^{(h_1,h_2)}(x) - V_2^{(h_1,\widehat{h}_2)}(x).$$

The values V of joint policies are computed by (2.40). The actual temporal difference update used by the agent is the SARSA update (2.17), so that **eval** is simply the Q-value of the next taken joint action.

The first part of the maximization argument in (2.41) accounts for the "best-response" part of the agent behaviour, whereas the second part drives the system towards a Nash equilibrium by indirectly reducing the opponent's desire to deviate from its policy.

The extended optimal algorithm was empirically tested on simple, stateless games such as matching pennies and battle of the sexes. When the second agent was learning, the policies learned by the first agent kept on oscillating slightly without converging, most likely due to the heterogeneous nature of the criterion (2.41).

2.7 Adaptive techniques

Though the adaptive multiagent reinforcement learning thread is not very well represented, a few researchers have investigated some of the possibilities. The approaches fall under the heading of either parametric adaptation (Definition 1.8) or structural adaptation (Definition 1.9).

2.7.1 Parametric adaptation

The Win-or-Learn-Fast policy hill-climbing (WoLF-PHC) algorithm performs parametric adaptation of the learning process (Bowling and Veloso, 2002). The approach combines basic Q-learning with the policy gradient ascent idea outlined in Section 2.6.1 (Singh et al., 2000b).

More precisely, in the simple two-agent, two-action setting, the step size parameter in (2.37) is allowed to take one of two values ("adapt" in a coarse sense) on the basis of the analysis of the agent's behaviour:

$$\begin{cases} \alpha_{k+1} = \alpha_k + \delta_{1,k} \dfrac{\partial V_1(\alpha_k, \beta_k)}{\partial \alpha} \\ \beta_{k+1} = \beta_k + \delta_{2,k} \dfrac{\partial V_2(\alpha_k, \beta_k)}{\partial \beta} \end{cases} \tag{2.42}$$

where:

$$\begin{aligned} \delta_{1,k} &= \begin{cases} \delta \cdot \delta_{\text{win}} & \text{if } V_1(\alpha_k, \beta_k) > V_1(\alpha^*, \beta_k) \\ \delta \cdot \delta_{\text{lose}} & \text{otherwise} \end{cases} \\ \delta_{2,k} &= \begin{cases} \delta \cdot \delta_{\text{win}} & \text{if } V_2(\alpha_k, \beta_k) > V_2(\alpha_k, \beta^*) \\ \delta \cdot \delta_{\text{lose}} & \text{otherwise} \end{cases} \end{aligned} \tag{2.43}$$

Here, α^* and β^* are equilibrium strategies independently selected by the two agents. If $\delta_{\text{win}} < \delta_{\text{lose}}$ and under the infinitesimal step size assumption ($\lim_{\delta \to 0}$), the strategies of the agents were shown to converge to a Nash equilibrium, enhancing the result of Singh et al. (2000b), that only proved convergence of the average payoffs to Nash payoffs.

The intuition behind (2.42) is that the agent should change its policy cautiously when winning (as the opponent is likely to take strong measures to remedy the situation), and boldly when losing, to improve its chances of escaping the losing situation. This should encourage convergence, and the effect should extend beyond self-play, i.e., when just one of the agents uses (2.42), and the other one uses some other learning rule. The goals of

the researchers in designing the algorithm were the rationality and convergence properties described in Section 2.3.3.

A practical version of the algorithm was given by authors, supporting problems with multiple states, any number of agents, and discrete action spaces of any finite size. This practical version is given in Algorithm 2.6. It uses a heuristic average performance criterion to assess whether the agent is winning or losing (line 11).

Note that the algorithm does not depend directly on the knowledge of, or on the interaction with other agents.

The policy update in line 12 is given in a simplified form; the actual update used ensures the validity of the probability distribution.

Algorithm 2.6 WoLF-PHC for one agent

Input: learning rate α, discount factor γ, step sizes δ_{win}, δ_{lose}

1: $Q(x,u) \leftarrow 0$, $h(x,u) \leftarrow \frac{1}{|U|}, \quad \forall x \in X, u \in U$

2: $C(x) \leftarrow 0, \quad \forall x \in X$ ▷ state visits counter

3: $\bar{h}(x,u) \leftarrow \frac{1}{|U|}, \quad \forall x \in X, u \in U$ ▷ average policy

4: observe initial state x

5: **loop**

6: draw u according to $h(x,u)$

7: apply u with suitable exploration, observe r and x'

8: $Q(x,u) \leftarrow Q(x,u) + \alpha \left[r + \gamma \max_{u' \in U} Q(x',u') - Q(x,u) \right]$

9: $C(x) \leftarrow C(x) + 1$ ▷ increment visits counter

10: $\bar{h}(x,\tilde{u}) \leftarrow \bar{h}(x,\tilde{u}) + \frac{1}{C(x)} \left(h(x,\tilde{u}) - \bar{h}(x,\tilde{u}) \right), \quad \forall \tilde{u} \in U$ ▷ update average policy

11: $\delta \leftarrow \begin{cases} \delta_{\text{win}} & \text{if } \sum_{\tilde{u} \in U} h(x,\tilde{u}) Q(x,\tilde{u}) > \sum_{\tilde{u} \in U} \bar{h}(x,\tilde{u}) Q(x,\tilde{u}) \\ \delta_{\text{lose}} & \text{otherwise} \end{cases}$ ▷ adapt step size

12: $h(x,\tilde{u}) \leftarrow h(x,\tilde{u}) + \begin{cases} \delta & \text{if } \tilde{u} = \arg\max_{\tilde{u}' \in U} Q(x,\tilde{u}') \\ -\frac{\delta}{|U|-1} & \text{otherwise} \end{cases}, \forall \tilde{u} \in U$ ▷ policy

13: $x \leftarrow x'$

14: **end loop**

The algorithm was empirically tested on two simple, stateless games: matching pennies and rock-paper-scissors, and on two games with small state spaces: a 3×3 gridworld and the gridworld soccer game in Littman (2001a). Convergence was achieved in all cases, but only after extremely large numbers of learning iterations: around 1 million for matching pennies, on the order of $100,000$ for the 3×3 gridworld, and above 1 million for the gridworld soccer.

The averaging nature of the heuristic adaptation criterion gives the algorithm a certain amount of inertia, which may be at least part of the reason for the very slow convergence. Banerjee and Peng (2003) propose an instantaneous performance criterion instead of (2.43), expressed in terms of the first and second order difference of the agent strategy. Using this criterion, the adaptation rule for the first agent is:

$$\delta_{1,k} = \begin{cases} \delta \cdot \delta_{\text{win}} & \text{if } \mathrm{d}\alpha_k \cdot \mathrm{d}^2\alpha_k < 0 \\ \delta \cdot \delta_{\text{lose}} & \text{otherwise} \end{cases} \tag{2.44}$$

where $\mathrm{d}\alpha_k = \alpha_k - \alpha_{k-1}$ and $\mathrm{d}^2\alpha_k = \mathrm{d}\alpha_k - \mathrm{d}\alpha_{k-1}$.

The validity of the criterion was proven and a practical algorithm was demonstrated both in simple games and in a two-dimensional block pushing task with continuous state. The results were encouraging: the algorithm converged on the simple tasks significantly faster than WoLF-PHC, by factors of around 2 to 4 in the various problems. On the continuous-state task, the comparison was not so favourable in terms of convergence speed, but the quality of the learned policy was more than around 2 times better, in terms of average path length to the goal position.

2.7.2 Structural adaptation

Existing work on structural adaptation in multiagent RL focuses on the adaptation of the action dimensions of the state-action space (Fulda and Ventura, 2003; Kok et al., 2005b). This is closely related to the empirical study of the concept of *awareness* in cooperative multirobot teams. Awareness was defined by Touzet (2000) as "the perception of other robots locations and actions". In the MA-RL setting, "location" should be replaced by the more generic term "state".

Structural adaptation in MA-RL takes the awareness concept one step further, discriminating the awareness needs of the agents on the basis of state. We can relate the need of awareness to the degree of coupling between agents. If in some region $\tilde{X} \in X$ of the state space the reward function of agent i depends significantly only on its own action choice:

$$\exists \widehat{\rho}_i : X \times U_i \to \mathbb{R} \text{ such that } \overline{\rho}_i(x, u_1, \ldots, u_i, \ldots, u_n) \approx \widehat{\rho}_i(x, u_i), \\ \forall x \in \tilde{X},\, u_j \in U_j,\, j \in A,\, j \neq i \quad (2.45)$$

we can say that in that region agent i is "loosely coupled" with other agents. In this situation, it is likely that the agent can learn well and achieve good performance without being aware of the other agents.

On the other hand, if (2.45) is not satisfied over the region $\tilde{X}$, meaning that the reward function of agent i depends significantly on several components of the joint action, we can say that agent i is "tightly coupled" with the corresponding agents in that region, and we can conclude that it needs to be aware of these agents in order to perform well.

The important questions here are, of course, how to identify the regions of space where the agent is loosely coupled with the other agents, and, for those regions where it is not, to determine with which of the agents it is tightly coupled. The literature takes heuristic approaches to answering these questions.

Fulda and Ventura (2003) use a tree structure where every node contains Q-values. Initially, the Q-values are discriminated only on the basis of the agent's own action. There exists a node for each state, and each node contains a vector of Q-values discriminated by the agent's action. Take as example the node for some given state $\underline{x}$, containing the Q-table with one dimension $Q_i(\underline{x}, u_i)$. Such a node has as children a set of $n-1$ fringe nodes, each corresponding to one of the other agents' action variable. Inside a fringe node, a more detailed Q-table is stored, discriminating values by the action of the agent corresponding to that node: in our example, the fringe node for agent j, child of node for state $\underline{x}$, would contain a two-dimensional Q-table of the form $Q_i(\underline{x}, u_i, u_j)$.

After a given number of learning steps elapses, the fringe children of each node are examined, to determine which of them could lead to the greatest increase in returns if Q-values were discriminated on the action variable in that node. The node is then expanded along

that action, becoming a branch node with a regular node as child for each action. If, say, the fringe node for agent j were chosen for expansion, it would spawn a set of $|U_j|$ nodes, each corresponding to some fixed $\underline{u}_j \in U_j$ and containing a one-dimensional Q-table of the form $Q_i(\underline{x}, u_i, \underline{u}_j)$, where only u_i varies over U_i. Each such child receives its own set of fringe nodes, $n-2$ to be precise, since one of the action variables was expanded: one fringe node for each agent different from i and j. The process then continues until either a maximum expansion depth has been reached, or the increase in returns that could be gained by expanding nodes drops below a specified threshold.

The algorithm was tested on a stateless game with four players (an extended version of matching pennies), where it achieved a performance close to that of an algorithm that considered the complete joint action $\boldsymbol{u}$ from the start, while storing a less number of Q-values.

Kok et al. (2005b) introduced a more structured approach that uses of a model able to explicitly represent the awareness needs of the agents. This model is the coordination graph, described in detail in Section 4.7. In brief, a coordination graph has agents as nodes, and relationships between agents as arcs. An agent needs to be aware of all the agents it is directly connected with. A variable elimination algorithm can be used to choose a coordinated joint action given that the graph structure and the value functions of the agents are known. The authors present a method of learning the structure of the coordination graph, based on the statistical analysis of expected returns for hypothetical relationships.

The method was empirically evaluated on a simple example problem and in the predator-prey domain, where it is able to reach the performance of team Q-learning, after a longer time needed for discovering the value rules.

2.8 Issues in realistic RL

When MA-RL (and RL in general) are applied to real-life tasks or realistic simulations, a number of issues arise that, for the sake of brevity, haven't been discussed above. We briefly make some remarks on the most relevant of these issues in what follows.

Large and / or continuous state and action spaces

When the state and action spaces of the agents are large and / or have continuous dimensions, tabular representations of the value functions cannot be used. Function approximation techniques from supervised learning can help in these situations. Linear approximators such as tile coding, or nonlinear approximators such as neural networks, can be used (Sutton and Barto, 1998).

The problem is that many theoretical convergence guarantees no longer hold for most function approximators. One class of methods that does retain its convergence properties when function approximators are used is direct policy search by gradient descent. These methods, however, are typically slow and can get stuck in local optima (Baird, 1995; Baird and Moore, 1998).

Partial measurability

When the state of the environment is not completely measurable, the Markov assumption is violated. Depending on the severity of this violation, it may or may not be safely ignored by the learning agents. When it needs to be accounted for, the literature uses one of two

classes of solutions. The first is to maintain probabilistic distributions of beliefs on the state of the underlying Markov task, and update these beliefs with observations using a Bayesian framework. This incurs high computational costs. The second class of solutions endows the agent with the ability to remember sequences of states, in the hope that these sequences will retain sufficient information to satisfy the Markov property (Whitehead and Lin, 1995).

Prior knowledge

In realistic problems, *tabula rasa* model-free RL is not effective: exploration of a huge state-action space, delayed reinforcement, and uncertainty, slow down learning or make it impossible. While it is unlikely that accurate models of the task will be available, prior knowledge can still be used to help the agents learn, by incorporating *bias* into the learning solution. This can be done in many ways, among which:

- *Initialization.* The value function can be initialized so as to contain prior knowledge. This might not be a trivial task, especially when using generalizing function approximators.

- *Local reinforcement signals* can help alleviating the structural and temporal credit assignment problem. This is probably the most natural way of biasing reinforcement learning, though it is contradicting with the recommendations of Sutton and Barto (1998), which state that the reinforcement signal should only encode the goal of the agent, and not the path to the goal. E.g., Matarić (1996) used progress estimators, akin to error signals in process control, to tell the agent when it is on the right path and when not. The same work used heterogeneous reinforcement signals to encode information about multiple agent goals.

- *Teaching.* A human could temporarily take control of an agent and guide it towards the goal, afterwards relinquishing control and allowing the agent to fine-tune its behaviour. This can speed up learning significantly, as the longest phase of model-free RL is when the agents, having no knowledge whatsoever of their goal, roam about the state space searching for it.

- *Shaping.* Start learning on very simple tasks, then progressively increase the difficulty.

- *Problem decomposition* followed by layered learning (Stone and Veloso, 1998). Elementary behaviours can be learned separately and then integrated into a higher-level policy to solve the original task.

- *Reflexes* provided to the agent upon inception. These can be used as building blocks for a more complex, learned behaviour (e.g., Matarić, 1996).

2.9 Concluding remarks

The field of multiagent reinforcement learning has not yet reached maturity. The work on extending the firm and well-understood results from single-agent RL to the multiagent case is currently in progress. A clear sign of this is the controversy existent in the field regarding a suitable MA-RL goal (Section 2.3.3).

Another sign is the gap between the theoretical results and the practical applications of MA-RL.

Theoretically justified methods, strongly influenced by game theory, make very restrictive assumptions on the learning task in order to prove their results, assumptions that do not hold in most practical situations. E.g., some of the methods assume that the agents know each other's reward functions; this will rarely be the case. Other methods assume that all the agents use the same learning algorithm, which removes heterogeneous and open multiagent systems from the discussion.

Another problem, inherited from the single-agent case, is the requirement for an explicit tabular representation of the state-action space. The problem is aggravated in the multiagent context by the exponential explosion of this space with the number of agents. This implies that learning algorithms should be very selective with the information they use, or in other words, that they should be "local" as much as possible.

Moreover, empirical evaluations of the algorithms are typically performed on very simple problems, some stateless, some with very small state spaces. Hints on how algorithms would scale up to realistic problem sizes are rarely given (an exception is e.g., Banerjee and Peng, 2003).

On the other hand, practical applications mostly use direct extensions of the basic, single agent algorithms. These extensions are placed on top of an often complex and ad-hoc machinery, designed to simplify the otherwise very difficult learning problem. E.g., features are extracted from the environment state, and the value function is computed in terms of these features. Local and instantaneous reward functions are used to guide the agents through the state space (Matarić, 1996).

This machinery makes the analysis of the methods very difficult. One important reason for this is that, many times, the machinery takes into account the other agents. Thus, the agents learn about each other in an indirect and obscure way.

Many MA-RL algorithms are designed for stateless games. Even those that do work in multi-state problems are biased by the stateless, game-theoretic perspective. Solving for game-theoretic equilibria in a stage-wise fashion sits at the core of theoretically justified MA-RL methods. However, the meaning and the suitability of stage-wise game-theoretic equilibria are doubtful in the context of finite and infinite horizon tasks with delayed reward.

An issue with the RL framework in general is that the RL goal is focused on optimality, disregarding other desirable properties of the agents' behaviour, such as robustness to changes in the environment, or an approximately monotonic increase in the performance during the learning process. Many RL algorithms offer asymptotic convergence guarantees without saying anything at all about the transient performance of the agent.

A set of relevant research questions arises from these considerations, stemming a corresponding set of possible research directions:

1. What is a suitable generic MA-RL goal? How can it incorporate performance requirements other than optimality?

2. Are stage-wise game-theoretic methods suitable for the multiple-states, finite or infinite horizon, delayed reward tasks?

3. How to unify the theoretically justified and the practically applied MA-RL methods? This question implies two complementary research directions: the first attempting to

relax the assumptions of the theoretically justified methods, and the second attempting to prove the soundness of practical methods.

4. How can a RL agent determine the minimal information on the basis of which it can learn effectively?

A research direction related to question 4 arises in the context of partially measurable learning tasks. If the state of the world is observable in the control-engineering sense, without being directly measurable (see Section 1.3), control-engineering observers could be used to estimate the unmeasurable components of the state. The use of observers is, however, conditioned on the availability of (some knowledge on) a model of the environment.

Opportunities for adaptive learning

In the area of parametric adaptation, an immediate opportunity is represented by the basic parameters of multiagent (and also single agent) RL: learning and exploration rates. The learning performance is very sensitive to these parameters. Exploration is especially important for the quality of learning, though most of the current approaches use very simple exploration techniques. Good initial values and decay schedules for the learning and exploration rates are typically obtained through a painstaking trial and error process. An automated procedure for the online adaptation of these parameters would be of significant value.

Besides these basic parameters, many methods use other specific parameters, whose adaptation could lead to improvements in performance. E.g., the Win-or-Learn-Fast algorithm step sizes δ_{win} and δ_{lose} give a very coarse adaptation; a way of adaptively shifting the step size between these extremes could be investigated.

As shown in Section 2.7, structural adaptation focuses on the action dimensions of the state-action space of the learner. This is a fruitful research direction that should be pursued. It might also be, however, that components of the environment *state* are irrelevant to the learner in all but a small part of the state space. E.g., the state components referring to some other agent might be relevant only in those regions of the state space where the learner closely interacts with that agent. This is related to research question 4 above. Significant reductions in computational requirements and learning time could be obtained in these cases if the learner considers only the relevant parts of the state.

This is an aspect typically disregarded in the literature on MA-RL, where the state of the world is regarded as monolithic and mandatory for all agents to consider. However, techniques encountered in the action space adaptation, such as second order analysis of returns, can be applied to state space adaptation as well. We have already begun investigating this possibility (Buşoniu et al., 2005).

This idea can be used as a starting point to develop techniques that would allow the agent to discover on its own when simple learning techniques, closer to single-agent RL, suffice to solve the problem, and switch to multiagent learning only when necessary. This would be not just structural, but full-blown adaptation, and would be an important step towards bridging the gap mentioned in the beginning of this section.

Chapter 3

Multiagent learning: other methods

While reinforcement learning is the technique most often applied to learning in multiagent systems, there are a number of other approaches worth considering. We review some of them here, focusing on learning as a tool of direct improvement of the agent's behaviour in the world, as opposed to e.g., its knowledge about other agents. This chapter is not intended to be a comprehensive presentation of non-RL methods in multiagent systems, rather its goal is to introduce a few illustrative approaches.

We begin by presenting evolutionary approaches to multiagent learning. We then review heuristic learning techniques, and close the chapter with a set of brief concluding remarks.

3.1 Evolutionary techniques

Genetic algorithms and genetic programming are the most popular evolutionary computation techniques. A *genetic algorithm* is a population-based heuristic search method that uses mechanisms borrowed from biological evolution, such as mutation and recombination.

A genetic algorithm maintains a randomly initialized population of candidate solutions represented as strings of values (e.g., binary values). These strings are called chromosomes, and the values are called genes. A fitness function gives the quality of the solution represented by such a chromosome as a real number. At each iteration, a genetic algorithm performs the following operations:

1. All the chromosomes are evaluated by the fitness function.

2. A selection function is applied to the population, probabilistically choosing a set of chromosomes to reproduce. The selection is biased towards chromosomes with higher fitness, but does not completely remove probability weight from poor chromosomes to maintain diversity in the population (i.e., to avoid local optima).

3. The selected chromosomes reproduce via mutation (e.g., bit flips: a chromosome `0000` might mutate to `0010`) and / or crossover (e.g., two chromosomes `0000` and `1111` might recombine via crossover to give birth to two new chromosomes `0011` and `1100`). The new chromosomes are added to the population.

The algorithm stops after a given number of iterations or when the quality of the best solution found exceeds a given threshold.

Genetic programming is a class of genetic algorithms that evolves programs. These programs are parse trees composed of building blocks that can be either terminals or non-terminals. Terminals are constants, variables, or functions without arguments. Non-terminals are functions that require arguments. In conventional genetic programming, all non-terminals accept and return a single value of a fixed data type (e.g., float).

Evolutionary computation searches directly in the space of agent behaviours, so it is closely related to direct policy search in MA-RL. The difference is that here the problem is not formalized as RL.

An important thing to note is that learning is offline: for each evolved generation, the performance of each agent on the task is tested over a number of trials, yielding the fitness value of the agent. Then, the selection function is applied, a new population of agents is born, and the process continues. The performance evaluation step is somewhat similar to learning value estimates in Monte-Carlo RL.

Even if the best individual somehow survives for many generations, that individual is not in itself significant in the evolutionary view; genetic algorithms work with populations, and not individuals. Learning takes place not along the lifetime of the agent, but along many generations of (hopefully improving) agents.

Haynes et al. (1995) applied an enhanced variant of genetic programming called strongly typed genetic programming to the predator-prey domain (see Section 5.2). Strongly typed genetic programming allows variables of any type and the definition of generic functions that work on generic data types. The restrictions it imposes also incur a smaller search space than that of basic genetic programming.

In order to evolve a predator program, the authors used the following set of terminals: boolean, action ("north", "east", "south", "west", "here" – stand still), a terminal for "self", the current predator agent, and one for the prey agent. They used the following set of functions:

- the if-then-else construct.
- a distance comparison function.
- a function computing the resulting cell position after taking a given action in a given cell.
- a function computing the Manhattan distance between two cells given as argument (the Manhattan distance is the sum of the horizontal and vertical offsets between the cells).

Using these functions and terminals, a predator program was evolved by strongly typed genetic programming. The same program was used by all the four predators. The resulting best program instructed the predators how to converge on the prey in a more efficient manner than several previous predator algorithms from the literature.

Programs evolved with strongly typed genetic programming were more humanly understandable than those evolved with basic genetic programming. They were also able to generalize behaviour to different world sizes, whereas the genetic programming solutions were not. The higher quality of the strongly typed genetic programming solutions was probably due in the most part to the smaller search space size.

Haynes and Sen (1996) attempted to improve on this solution by *competitive coevolution* of predators and prey. In coevolution, evolutionary techniques are applied simultaneously to

several agent behaviours, and the evaluation of these behaviours is based on the performance they achieve in interaction. Nevertheless, learning is still offline as explained above.

In the setting of Haynes and Sen (1996), a predator and a prey program were developed by competitive coevolution, in the hope of an arms race that would render the quality of the predator behaviour progressively better as the prey became more skilled in evading it. The results were, however, surprising: the prey quickly developed a very simple unbeatable strategy: it chose a random direction and steadily moved in a straight line along that direction (the world was toroidal, so the prey was able to do that indefinitely). This result casts a shadow of doubt on the relevance of the prey behaviours typically used in the pursuit domain as test-cases for multiagent learning and cooperation algorithms.

Cooperative coevolution was investigated by 't Hoen and de Jong (2004) in the dispersion game. This can be thought of as a special case of strategic game (Definition 2.5), where the agents have the same set of actions, $U_1 = \cdots = U_n = U$, and they prefer to select different actions. Dispersion games are among others helpful in modeling task allocation problems, where, if the tasks have equal priorities, the goal is for the agents to distribute as evenly as possible across the set of tasks. In this context, the actions of the agents are task choices, so each action corresponds to a task.

In the version used by the authors, the dispersion game has the same number of actions as agents: $|A| = |U| = n$. Thus, the ideal situation is where no two agents select the same action. This can be represented by the following fully cooperative reward function:

$$\overline{\rho}(u_1, \ldots, u_n) = \begin{cases} 1 & \text{if } u_i \neq u_j, \quad \forall i, j \in A, i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

The authors evolved n populations of behaviours in coevolution, one for each agent. The behaviour of an agent was represented as a vector of preferences over actions, $\tilde{u}_i = [\tilde{u}_{i,1}, \ldots, \tilde{u}_{i,n}]^{\mathrm{T}}$, where $\tilde{u}_{i,o}$ is the preference of agent i towards selecting action u_o, $o = 1, \ldots, n$. Each such vector is a chromosome, an individual in the population. The agent selects an action using a softmax rule (2.16):

$$\sigma_i(u_o) = \frac{e^{\tilde{u}_{i,o}/\tau}}{\sum_{\tilde{o}=1}^{n} e^{\tilde{u}_{i,\tilde{o}}/\tau}}. \tag{3.2}$$

The fitness of the agents was not computed by using the reward function (3.1), but in the following way. Each chromosome from an agent population was paired with the fittest chromosomes from the other agent populations. The strategies represented by these chromosomes were then used by the agents to select actions. When several agents chose the same action – the same "task", the task was assigned to one of them randomly. The fitness of the chromosome was 1 if a task was successfully assigned to the agent using the corresponding strategy, and 0 otherwise. Several trials were performed and the fitness values were averaged to obtain a better estimate of the agents' performance.

The fitness of the MAS as a whole was computed by selecting the strategies with the best fitness from the agent populations, putting them together and executing them. The fitness of the MAS was then the percentage of tasks that were successfully assigned.

The authors also experimented with the idea of collective intelligence. Collective intelligence addresses the problem of designing utility functions for the agents so that they are both easily learnable, and when learned give good overall performance of the multiagent system.

E.g., the so-called "wonderful life utility" is computed by subtracting from the collective utility of the system with the agent in place, the collective utility with the agent removed.

In the coevolution setting, the utility is measured by the fitness of the system. Then, the wonderful life utility of an agent is the difference between the fitness of the MAS with the agent in place, and the fitness of the MAS with the agent removed.

The authors found that using the wonderful life utility led to enhanced performance of the agents. The process was also significantly less sensitive to the learning parameters such as the population size, or the number of trials ran for estimating the fitness value.

3.2 Heuristic approaches

The approaches we discuss in this section are oriented toward resource management and task allocation (see Section 5.4). They typically use a decision making algorithm that relies on a number of parameters. These parameters are learned by observation, many of the learning rules being similar in spirit to the temporal difference update (2.17), though the methods do not use a RL framework.

The Learning ALLIANCE (L-ALLIANCE) framework deals with task allocation in heterogeneous multiagent systems (Parker, 1997). The agent actions in L-ALLIANCE are structured on three levels. An agent (say, agent i) can perform a set of a mid-level functions U_i. When performing any function u_i of this set, the agent is working on a high-level task $t_i(u_i)$, $t_i : U_i \to T$ where T is the set of tasks required by the system's mission. Finally, at the lowest level the agents can take over functions $u_i \in U_i$ (begin working on task $t_i(u_i)$), or give them up (stop working on task $t_i(u_i)$ before it is completed). The efficiency of the agent in performing a function u_i is evaluated via a performance metric, such as completion time, applied to the task $t_i(u_i)$.

In executing the low-level actions, the agents are driven by, respectively, motivations of "impatience" for taking over functions and "acquiescence" for giving them up. The rate at which they become impatient or acquiescent depends upon three control parameters:

- $\delta_i^{\text{fast}}(u_i)$, the rate of impatience of agent i concerning function u_i, when no other agent is working on $t_i(u_i)$.
- $\delta_i^{\text{slow}}(u_i, j)$, the rate of impatience of agent i concerning function u_i, when agent j is working on $t_i(u_i)$.
- $\psi_i(u_i)$, the time agent i will maintain function u_i before acquiescing to another agent.

Each agent adjust these parameters by the heuristic learning mechanism, using a set M_i of monitors, one monitor for every function it is able to perform, $|M_i| = |U_i|$. The monitor m_i of function u_i observes all the agents performing task $t_i(u_i)$ and records their performance.

The system execution is divided in two phases: a learning phase (termed "active learning" by the authors), and a problem solving phase (though this phase is termed "adaptive learning" by the authors, in our adaptive learning framework the name is not justified; we give the reason below). In the learning phase, agents are maximally patient and minimally acquiescent (i.e., they do not take over attended tasks and do not give up tasks they began undertaking). The purpose of this phase is to allow the agents to familiarize with the performance of the team. In the problem solving phase, the agents become impatient and acquiescent according to their

control strategies, to achieve performance, but keep updating their monitors, so that their contents reflect the current performance of the agents.

We do not give the actual heuristic control strategies and updates (for those see Parker, 1997). Instead, we examine how L-ALLIANCE fits into our DMAS framework (Definition 1.2). The information contained in the monitors M_i is part of the agent's internal state s_i, and is updated on the basis of the agent's observations of other agents y_i via the dynamics p_i. The internal state s_i also contains the impatience and acquiescence levels of the agent, as well as the parameters used in the alteration of these levels. These parameters are themselves altered on the basis of the information contained in the monitors – so, state influences state, also via the dynamics p_i. The agent then decides to take over or give up tasks by the policy function h_i applied to the impatience and acquiescence components of the state.

We see then why in this view the term "adaptive learning" does not apply to the monitor and parameter updates occurring during the problem solving phase. In fact, these processes are still simply learning, though online, as opposed to the learning phase that can be considered in a sense off-line, because the performance of the system is not relevant then.

The relationship between L-ALLIANCE and DMAS is summarized in Table 3.1. Note that the control parameters $\delta_i^{\text{fast}}(u_i)$, $\delta_i^{\text{slow}}(u_i, j)$, and $\psi_i(u_i)$ are not a part of the internal state of the agent; they are only constant coefficients in the internal dynamics p_i.

L-ALLIANCE	**DMAS**
set of agents A	set of agents A
tasks T_i, functions U_i, low-level actions	action space U_i
set of monitors M_i	part of the internal state space S_i
impatience and acquiescence levels	part of the internal state s_i
monitors, impatience and acquiescence levels updating	internal dynamics p_i
low-level action execution rules	policy h_i

Table 3.1: Correspondences between the L-ALLIANCE and DMAS elements

Schaerf et al. (1995) approach the load balancing problem from a perspective closely related to RL. It is instructive to study their formal model of a load balancing multiagent system:

Definition 3.1 *A* multiagent multi-resource stochastic system *is a tuple* $\langle A, R, p, \delta, \zeta, L\rangle$ *where:*

- A *is the set of agents,* $|A| = n$ *being their number.*
- R *is the set of resources,* $|R| = m$ *being their number.*
- $p : A \times \mathbb{N} \to [0, 1]$ *is a probabilistic job submission function;* $p(i, k)$ *gives the probability with which the idle agent* i *submits a new job at time step* k*. An agent is idle at time step* k *if all the jobs it submitted have been completed prior to time step* k*.*
- $\delta : A \times \mathbb{N} \times \mathbb{R} \to [0, 1]$ *is a probabilistic job size function;* $\delta(i, k, d)$ *is the probability that the job submitted by agent* i *at time step* k *has size* d*.*
- $\zeta : R \times \mathbb{N} \times \mathbb{R} \to [0, 1]$ *is a probabilistic resource capacity function;* $\zeta(r, k, c)$ *is the probability that resource* r *has capacity* c *at time step* k*.*
- L *is the resource selection rule, instructing the agents which resource to choose when they submit a new job.*

The notation has been slightly altered to accommodate our notational conventions. The goal of the system is to minimize (i) the processing time per unitary job size, averaged over all jobs, and (ii) the standard deviation of this quantity. A small average job processing time yields efficiency, whereas a small standard deviation yields fairness, i.e., no jobs are processed significantly faster or slower than the others.

We discuss the selection rule L in more detail. The rule dictates how agents choose the resources to which they submit their jobs. Besides some fixed selection rules, the authors also study a learning selection rule they term "adaptive". This rule uses efficiency estimators e and counters for completed jobs b maintained by every agent for each resource r. The vectors e and b are initialized to 0.

In the spirit of RL, the information used for learning is local. It comes in form of experiences $(r, k_{\text{start}}, k_{\text{stop}}, d)$ for each completed job. An experience contains, in order, the resource where the job was submitted, the start and completion times, and the size of the job. At the receival of each such experience, the agent (say agent i) updates its efficiency estimators and job counters by:

$$e_i(r) \leftarrow e_i(r) + \alpha \left(\frac{k_{\text{stop}} - k_{\text{start}}}{d} - e_i(r) \right) \tag{3.3}$$

$$b_i(r) \leftarrow b_i(r) + 1, \tag{3.4}$$

where:

$$\alpha = \beta + \frac{1 - \beta}{b_i(r)}, \tag{3.5}$$

with β a positive constant.

The probability of an agent for selecting a resource r is given by the roulette wheel:

$$h_i(r) = \frac{\tilde{h}_i(r)}{\sum\limits_{\tilde{r} \in R} \tilde{h}_i(\tilde{r})}, \tag{3.6}$$

where:

$$\tilde{h}_i(r) = \begin{cases} e_i(r)^{-\nu} & \text{if } b_i(r) > 0 \\ E\{e_i\}^{-\nu} & \text{otherwise} \end{cases} \tag{3.7}$$

with ν a positive constant.

The relation (3.3) can be seen as a temporal difference update (2.17): the old estimate of $e_i(r)$ is moved a fraction α of the distance towards the target $\frac{k_{\text{stop}} - k_{\text{start}}}{d}$.

We investigate how this learning process and the formal model given by Definition 3.1 fits in the DMAS framework. The set of resources R is not represented explicitly in the DMAS, but it is part of the environment, and the state of the resources is part of the environment state x. The capacity function ζ is part of the environment dynamics f.

The experience tuples $(r, k_{\text{start}}, k_{\text{stop}}, d)$ of the agent are its observations y_i. The efficiency estimators e_i and the counters b_i are components of the internal state s_i, and the update formulae (3.3 – 3.5) are part of the agent dynamics p_i.

The job submission and size functions $p(i, \cdot)$ and $\delta(i, \cdot)$ can be interpreted as part of the agent i's policy h_i; resource choices and sizes of submitted jobs are then part of the stochastic

agent actions u. The dependence of $p(i,\cdot)$ and $\delta(i,\cdot)$ on time is captured in DMAS by the dependence of the agent's policy on its internal state s_i. Together with (3.6 – 3.7), the job submission and size functions completely specify the policy of the agent.

These correspondences are summarized in Table 3.2.

L-ALLIANCE	**DMAS**
set of resources R	environment state space X
resource choices, job sizes for submitted jobs	action space U_i
experience tuples $(r, k_{\text{start}}, k_{\text{stop}}, d)$	agent observations y_i
resource capacity functions ζ	part of the environment dynamics f
efficiency estimators e_i, counters b_i	agent internal state s_i
update rules (3.3 – 3.5)	agent dynamics p_i
$p(i,\cdot)$, $\delta(i,\cdot)$, resource selection strategy h_i	agent policy h_i

Table 3.2: Correspondences between the load balancing model and the DMAS elements

Now, is α in (3.3) an adaptive parameter or not? This question illustrates the difficulty of distinguishing between adaptive learning and "static", non-adaptive learning in some situations. Since (3.5) only trivially corrects β via the internal state to obtain the learning rate α, we choose not to consider this parametric adaptation. This distinction is, of course, slightly arbitrary, and might be more difficult to make out for more complex learning algorithms.

There are a number of approaches in the literature that do not regard multiagent learning as the highly interactive and proactive process considered in MA-RL and the methods described above. E.g., Plaza and Ontañón (2003) present a multiagent approach to case-based reasoning that is probably best described as distributed learning. The case base is divided among the agents, either in disjoint or partially overlapping subsets. When receiving a new problem to solve, an agent first attempts to solve the problem by itself, and assesses the quality of the solution (using e.g., a confidence measure). If the solution is deemed good, it is considered the final solution. If, on the other hand, the agent finds that the solution is not reliable, it proceeds to ask other agents about their opinion on the problem. This querying process continues until a termination condition is satisfied (e.g., a majority is realized).

This situation could arise in real-life if, for instance, each a group of organizations desire to maintain the privacy of their case bases, but nevertheless wish to collaborate in finding solutions.

Note that here the MAS is seen as a tool to facilitate a machine learning technique by decentralizing the learning process. This view is opposite to what has been encountered so far: in the other methods presented here and in Chapter 2, learning is a tool used by the MAS to improve its decision making process.

3.3 Concluding remarks

Some of the approaches presented in this chapter take a different view on learning than that introduced in Chapter 1, where learning is a process that improves the decision making abilities of an agent along its lifetime.

Evolutionary approaches use many generations of short-lived individuals to perform learning. For such approaches to work, the interaction between the agents and the environment

needs to be simulated very efficiently. This is because the fitness evaluation requires a large number of trials to be executed. Executing them in the real world seems to be out of the question. Also, it is unclear how the agents could adapt in an online fashion to changes in the environment, after a behaviour has been learned by evolutionary computation and loaded into the agents.

One of the approaches in Section 3.2 looked at the MAS as a tool for distributed learning. Our research does not focus on this perspective, because it appears to have little relevance to multiagent control.

The heuristic, and sometimes complex nature of the learning methods reviewed in Section 3.2 makes them difficult to analyze, and to judge whether adaptation of parameters or structure could be useful.

Chapter 4

Coordination

4.1 Introduction

The problem of coordination arises in multiagent systems due to the distributed nature of the control exercised by the agents. Coordination is defined by Vlassis (2003) as the process by which the individual decisions of the agents result in good overall decisions for the group. The problem is more stringent in cooperative multiagent systems, but also appears when the agents are self-interested.

An example of coordination in a cooperative setting is a team of agents controlling traffic lights in a busy city. If they do not properly synchronize their decisions of switching the colour of the traffic signals, traffic jams will probably result and the likelihood of accidents will increase. Another example is a network of controllers in a plant: two locally good control decisions might be harmful to the plant, leading it into a dangerous regime where both the installations and the people are at risk. When the agents are self-interested, they might still need to coordinate in certain situations: a police car and an ambulance driving to different emergencies still have to coordinate their entrance if they meet at an intersection, or they will crash and neither of the two emergencies will be serviced.

Formally, coordination can be defined in the context of a dynamic multiagent system (Definition 1.2) as follows.

Definition 4.1 Coordination *in a* DMAS $\langle A, X, f, x_0 \rangle$ *is the problem of consistent selection by the agents, at each time step* k*, of a joint action* $\boldsymbol{u}_k = [u_{1,k}, \ldots, u_{n,k}]^{\mathrm{T}} \in \boldsymbol{U}$ *that does not jeopardize the performance of any agent* $i \in A$*.*

The definition is necessarily vague; any coordination technique is good as long as the agents' opportunity to reach their respective goals is not eliminated by incorrect action selections. The definition does not imply that the goals of all the agents are achievable; it says only that they will remain achievable if they were so before the joint action selection. It also does not say how easy it will be for the agents to reach their goals after the action selection; some coordination techniques will be better than others.

An important thing to note is the tight interconnection between learning and coordination. Effective learning requires coordination between agent actions, though up to some point agents can learn useful things from miscoordination. Conversely, coordination can be assisted in many ways by learning.

The first aspect was already mentioned in Chapters 2 and 3, sometimes under disguise names such as "consistent joint action selection" or "equilibrium selection". This chapter

treats it in detail, reviewing frameworks and techniques used in achieving coordination. In order to investigate how learning can help coordination, for each of the discussed techniques, we differentiate between designed and learned coordination, and focus the exposition on the latter.

The chapter is structured in the following way. We classify coordination techniques along several dimensions, after which we review the coordination frameworks encountered in the literature. Then, we discuss some of the possible alternatives for coordination in a multiagent system, including all of those mentioned above: learning of coordination, social conventions, roles, and coordination graphs. We close with some concluding remarks and research opportunities.

4.2 Taxonomy

Degree of information sharing

Coordination techniques differ perhaps most significantly along the dimension of information sharing: how much information is shared among the agents, and how this information comes to be shared.

At one end of the spectrum lie coordination methods that do not require the agents to explicitly share any information. Agents using such methods must rely on learning, in one way or another, to develop coordination skills (e.g., Boutilier, 1996; Kok et al., 2005a). At the other end, agents exchange rich information via communication (e.g., Tambe, 1997). In between sit techniques relying on prior domain knowledge.

The distinction between these categories is not clear-cut. For instance, roles can rely exclusively on prior knowledge, or they might require communication to some extent in order to be properly assigned (Spaan et al., 2002).

Offline design vs. learned coordination

Coordination methods may be designed offline and hardwired into the agents, or learned by the agents from interactions, during their lifetime. The first approach is typically simpler; the second is more flexible and is beneficial in open or complex multiagent systems (Walker and Wooldridge, 1995).

At the very former end of the spectrum lie methods that view coordination as an offline phase of planning. In this view, the coordination phase is followed by an execution phase when the formed plans are actually pursued. It is unlikely that this type of offline coordination can handle the activity of the multiagent systems we focus on. This is because, first, it is unlikely that an accurate enough model of the problem is available beforehand, and second, even if the plans were initially satisfactory, their suitability is likely to degrade as the environment evolves. Therefore, we do not discuss such approaches further.

Along these taxonomy dimensions, we are interested in:

1. all degrees of information sharing;

2. both offline designed and learned coordination, especially the latter.

4.3 Coordination frameworks

The Markov game (Definition 2.6) and its stateless version, the strategic game (Definition 2.5) are popular formal models for the coordination problem in the non-communicative setting. The coordination problem in this setting is interpreted as the consistent selection by the agents of their part in a joint action. This joint action is somehow deemed good by the agents, and is many times a game-theoretic equilibrium such as the Nash equilibrium. Hence, the coordination problem in a Markov game is many times interpreted as the *equilibrium selection problem*, introduced in Section 2.6. Note, however, that game-theoretic equilibria are currently not accepted as the indisputable goal of the multiagent learning problem (Section 2.3.3).

Since we already discussed the strategic and Markov games, we proceed to a different formal model used in the non-communicative context by Walker and Wooldridge (1995) to describe the emergence of social conventions. This model is interesting because it is among the few that explicitly acknowledges the existence of an internal agent state – albeit this state is a simple memory.

The model consists of the following elements:

- The set of agents A.
- A common set of "strategies" U. The term "strategy" does not have the same meaning here as that described in Chapter 2: each of the strategies represents a possible convention the agents may agree upon. These can in fact be entire policies or constraints on policies, e.g., dictating that robotic agents should always yield the right-of-way to the robot coming from the right (see Section 4.5). However, for modeling the learning process it suffices if we identify them with primitive actions.
- A set of possible interactions $I = U \times U$. An interaction occurs when two agents meet and compare their strategies. This is in fact a special structure of the observation functions $\overline{\omega}_i$, pairing agents at each time step and allowing them to observe each other's action. Hence, the common observation space of the agents has the structure $Y = A \times U$, and an observation $y_i = [j, u_j]^{\mathrm{T}}$ identifies the agent j with whom agent i interacted together with the strategy it was observed to use.
- A "memory" s_i for each agent i. This memory consists of a finite sequence of past interactions to which agent i participated. Hence, the memory space is $S_i = Y^m$, where m is the length of the memory.
- A "strategy update function" $\overline{h}_i : S \to U$ that, given the agent's memory, chooses the strategy the agent will follow: $u_i = \overline{h}_i(s_i)$.

The goal is for the agents to eventually settle on the same strategy.

The update is in fact a policy, or a "meta-policy" since it switches among strategies. The update of the internal memory, though not explicitly present in the model, forms the internal agent dynamics p_i. Similarly, the memory of the agent is its internal state. The notations have been intentionally altered to bring forth these similarities. The similarities are summarized in Table 4.1. Note that the agents are homogeneous.

What we see, then, in this model, is a primitive form of learning where the learning process is a simple transfer of observations to the internal state.

Social conventions model	DMAS
set of strategies (conventions) U	action space $U_i = U, \forall i \in A$
interactions I	observation space $Y_i = A \times U = Y, \forall i \in A$
memory space S_i, of length m	internal state space $S_i = Y^m$
memory update function p_i	internal dynamics p_i
strategy update function $\overline{h}_i$	policy $\overline{h}_i$

Table 4.1: Correspondences between the social conventions emergence framework and the DMAS elements

From the coordination frameworks using communication, we illustrate the Shell for TEAMwork (STEAM) model (Tambe, 1997). The STEAM model relies, typically for its class, on vast domain knowledge shared among the agents. This knowledge is specified in operational form inside each agent, as an operator hierarchy. The operator is the building block of the agents' behaviour. A portion of an example operator hierarchy for a helicopter attack domain is presented in Figure 4.1 (taken from Tambe, 1997).

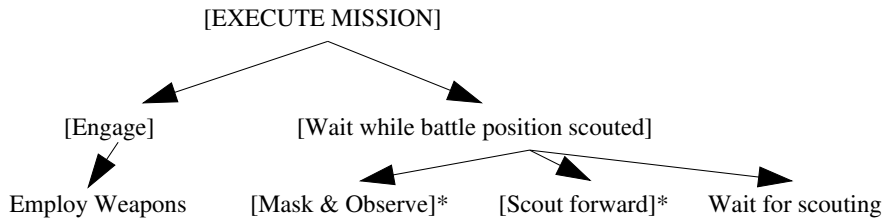


Figure 4.1: An example STEAM operator hierarchy.

Operators are of two types: individual operators, that require a single agent to be executed, and team operators, requiring the whole team or a subteam for execution. The team operators are enclosed in square brackets in Figure 4.1, and those involving subteams are marked by an asterisk.

The core notion of STEAM is the *team joint intention*. A joint intention to do an action is a mental state of a team of agents where all members of the team are jointly committed to do that action, and they mutually believe that they are committed to do it. Informally, the joint commitment ensures that the action is either taken to the end or terminated jointly by all the involved agents. In order to execute a team operator in STEAM, the corresponding team or subteam instantiates a team joint intention to execute it.

A STEAM agent also has a private state that includes information on the application of its individual operators, and also a team state including beliefs on the team and subteams it participates in (such as the team members, the team leader, available communication channels). The goal of the STEAM agent team is to achieve the top-level team operator (in Figure 4.1, [EXECUTE MISSION]).

Whenever a team operator is found to be unachievable by the team, a special team operator called [repair] is invoked, that attempts to reassign agents within the team so that the failed operator becomes achievable again.

The role of communication in STEAM is to ensure the correct application of the team operators, by communicating beliefs. This can lead, however, to a large number of messages being exchanged. In order to limit the communication overhead, STEAM implements a selective

communication mechanism. More precisely, for every message it is about to send, the agent heuristically balances the cost of communication with the estimated cost of miscoordination, taking into consideration the utility of achieving the goal of the team operator and the possibility that the team already knows the information contained in the message. The agent then sends the message only if the miscoordination cost exceeds the communication cost.

It is not easy to precisely fit this highly complex and expressive framework into the more general DMAS model. An approximate correspondence is given in Table 4.2.

STEAM	**Communicative DMAS**
individual operator	domain-level policy h_i^{e}
team operator	domain-level and message sending policies h_i^{e}, h_i^{snd}
private and team state	agent internal state s_i
selective communication	message sending policy h_i^{snd}

Table 4.2: Correspondences between the STEAM and the communicative DMAS elements

The Communicative Multiagent Team Decision Problem (COM-MTDP) is a framework introduced with the goal of analyzing the optimality / complexity tradeoff in multiagent coordination (Pynadath and Tambe, 2002). We have already introduced and analyzed this framework as an example in Section 1.4.3.

We illustrated a few representative approaches to modeling coordination in multiagent systems. We mention here a few other interesting ideas, without going into details. Stone and Veloso (1999) introduced the *periodical team synchronization* domains as time-critical environments in which agents act autonomously with low communication, but in which they can periodically synchronize in a free-communication setting. In these periods of free communication, the agents establish so-called "locker room agreements", consisting of role formations together with environmental triggers for switching between formations. The agents then move into the low-communication periods, where they apply the previously set agreements, and are also allowed a limited amount of communication. This is a combination of off-line and on-line coordination. The authors applied this approach to robotic soccer, where the free-communication period was the mid-game break. A more relevant example is perhaps a robotic search and rescue team, that would have the opportunity of establishing locker-room agreements before starting a mission, and possibly in periods of less intensive activity during the mission.

Another interesting view is that coordination can be mediated by specialized entities existing in the environment, called *coordination artifacts* (Ricci et al., 2005). Such an artifact consists of a usage interface operable by agents via domain-level actions, a set of operating instructions describing the interface, and a specification of the coordination behaviour implemented by the artifact. One can think of a semaphore system in an intersection, with green light request buttons for pedestrians and bikers, in order to form an image about such an artifact. This type of coordination falls into the online category, and offers a compromise along the information sharing dimension, in the following way. The agents no longer need to share extensive domain knowledge, which is integrated into the coordination artifact; instead they only need to know how to read the operating instructions.

4.4 Learning coordination

The most popular model for learning coordination is the Markov game (Definition 2.6). As noted in Section 4.3, the coordination problem in this context is many times interpreted as the *equilibrium selection problem*.

Learning of coordination is typically studied on homogeneous, non-communicative multiagent systems endowed with full measurability. Many times, further assumptions such as measurable actions, or known reward functions, are necessary for the learning algorithms.

These are all characteristics of the game-theoretic approaches to multiagent learning. For a multiagent learning approach to classify as learning coordination, we further require that the agents reason explicitly on coordination. This means that the agents acknowledge the fact that the action choices of other agents are not necessarily part of the desired joint action; and that, reasoning explicitly in terms of the other agent's actions and using learning mechanisms, the agents strive to minimize the negative impact of this issue on their performance.

This includes learning of coordination mechanisms such as roles or social conventions. We do not discuss them in this section, but treat them separately in more detail in the following sections.

4.4.1 Learning about other agents

An obvious way to learn coordination is to learn how other agents behave, predict their actions using this knowledge, and choose an appropriate response to these actions. The knowledge on another agent's behaviour is the *model* of that agent. This approach is often called "opponent modeling" in the literature, because it was mostly used in the game-theoretic context to exploit other agents once their model is known. It is, however, useful in cooperative settings as well, when the agents only have access to limited information about each other, and to limited communication. We therefore refer to learning about other agents by the broader term *agent modeling*. Typically, modeled agents are assumed to be reactive, and thus reactive agent models are maintained and updated by the learning agent.

Note that, in order to exploit the learned models, the agent has to know the effect of other agent's actions on its performance. In a stochastic game, that means that the agent (say agent i) has to know, or learn a model of, its reward function ρ_i.

Fictitious play is one of the simplest agent modeling algorithms. It was designed for repeated stochastic games. In fictitious play, agent i models each other agent $j \in A, j \neq i$ by a counter function $N_j : U_j \to \mathbb{N}$. The value $N_j(u_j)$ counts how many times agent j has been observed taking action u_j. At every game repetition, agent i estimates j's mixed strategy by :

$$\widehat{\sigma}_j(u_j) = \frac{N_j(u_j)}{\sum\limits_{\tilde{u}_j \in U_j} N_j(\tilde{u}_j)}, \quad u_j \in U_j. \tag{4.1}$$

It then plays a best response σ_i^* to the estimated reduced strategy profile $\widehat{\boldsymbol{\sigma}_{-i}}$ (this reduced strategy profile consists of the estimated strategies of all the agents except i – see Section 2.3.2). Convergence results for fictitious play exist only in several restrictive settings, among which fully competitive games are notable. An interesting property of fictitious play is that Nash equilibria are absorbing: that is, if at any point the agents play a Nash equilibrium, they will continue playing that equilibrium for all subsequent repetitions of the game.

The most important shortcomings of fictitious play are the restriction to repeated games, and the requirement that the actions of the agents are measurable. Boutilier (1996) attempted

to soften these restrictions. The author argued that coordination problem in a multiagent Markov decision process (MMDP) (Definition 2.7) can be decomposed into coordination problems at each state, and solved locally in terms of expected returns in those states. Each visit to a state of the MMDP is one play of an embedded repeated game corresponding to that state. An important advantage of this approach is that it allows for adding coordination overhead only to those states where coordination problems arise – i.e., where the optimal joint action is not uniquely determined.

The method used to learn coordination in this setting was a Bayesian extension of fictitious play. Agent i models agent j by a prior distribution over j's strategies. It uses this prior to predict the actions of agent j, and updates it via Bayes' rule using its observations. The Dirichlet distribution was used as prior in the work. The Dirichlet distributions in the MMDP states can be conveniently expressed with a "counter" function similar to that used in fictitious play: $N_j : X \times U_j \to \mathbb{R}$. The value $N_j(x, u_j)$ "counts" how many times agent j executed action u_j in state x. The meaning of the quotes will become evident immediately.

If actions are measurable, then N_j is easily updated by adding a unitary increment to the counter of the observed action. For the case of non-measurable actions, the agents use Bayes' rule to infer a distribution over other agents' actions. After each time step k, $N_j(x_k, u_{j,k})$ is increased with a fractional increment equal to the probability that agent j performed action $u_{j,k}$. This is why N_j is not exactly a counter function, though the meaning of its values is similar to that of the counters in fictitious play.

To infer the posterior probability that agent j executed $u_{j,k}$ at time step k, agent i uses:

$$P\left(u_{j,k} \,|\, x_k, u_{i,k}, x_{k+1}\right) = \frac{P\left(x_{k+1} \,|\, x_k, u_{j,k}, u_{i,k}\right) P\left(u_{j,k} \,|\, x_k\right)}{P\left(x_{k+1} \,|\, x_k, u_{i,k}\right)}, \tag{4.2}$$

where agent i observed a transition from x_k to x_{k+1}, after it executed $u_{i,k}$. The quantity $P\left(u_{j,k} \,|\, x_k\right)$ belongs to the estimated policy of agent j in x_k, and is represented by the Dirichlet prior, computed similarly with (4.1):

$$P\left(u_{j,k} \,|\, x_k\right) = \widehat{h}_j(x_k, u_{j,k}) = \frac{N_j(x_k, u_{j,k})}{\sum\limits_{\tilde{u}_j \in U_j} N_j(x_k, \tilde{u}_j)}. \tag{4.3}$$

The other two probabilities can be computed by integration from the transition function f: $P\left(x_{k+1} \,|\, x_k, \boldsymbol{u}_k\right) = f(x_k, \boldsymbol{u}_k, x_{k+1})$. Hence, (4.2) assumes that a model of the environment is known. Though the author does not mention this, a model of the world could be learned in a similar fashion with the agent model.

Hu and Wellman (2001) studied recursive agent models in the domain of an auction market. For the purposes of modeling, the policy of a non-learning agent was assumed to be deterministic and affine linear in its state:

$$\overline{h}_j(s_j) = A_j s_j + B_j, \tag{4.4}$$

with A_j and B_j constant matrices of appropriate dimensions. This linear relation expresses the relation between the quantities of goods s_j held by the agent, and its bidding prices u_j.

Note that (4.4) implies that the internal state of the modeled agent needs to be observed, or estimated in its turn.

The modeling level of an agent was defined recursively as follows:

- A 0-level learning agent models other agents by looking at the history data of those agents' actions (similarly to fictitious play).
- A 1-level learning agent models the policies of other agents, assuming these policies are fixed (e.g., of the form (4.4)).
- An n-level learning agent models other agents as being $(n-1)$-level learning agents.

Perhaps the most important conclusion of this work is that incorrect assumptions on other agents are harmful: the 0-level algorithm (making the fewest assumptions on other agents) performs better than higher-level algorithms if the modeling level of the other agents is under- or overestimated.

4.4.2 The value of coordination

Most coordination techniques ask the question: "How can agents coordinate?" and then proceed to seek for an answer, typically an optimal, coordinated joint policy of the agents. However, if coordination is not a certain fact – which is especially the case for learning of coordination, where several (perhaps many) rounds of miscoordination are required until agents learn how to coordinate – another important issue arises. It might be that the possible miscoordination is so harmful that an alternate, suboptimal but safer path is preferable. So, a new question is "When should agents strive to coordinate?" or, equivalently, "What is the value of coordination?"

Boutilier (1999) introduced an extension of value iteration that, together with the state of the underlying problem, explicitly considers the state of the coordination mechanism. The actual nature of the coordination mechanism is in principle unimportant. Its state variable can be intuitively thought of as having two values: "coordinated" and "uncoordinated". The discrimination along this state dimension enables the usage of the dynamic programming engine to reason on the benefits of attempting to coordinate.

Specifically, if a certain state of the underlying Markov game requires coordination, and the value of the coordination mechanism state at that underlying state is "uncoordinated", the agent may choose not to pursue the path to the uncoordinated state due to the costs of miscoordination.

Chalkiadakis and Boutilier (2003) placed the value of coordination in a fully Bayesian framework. The state of the agent is expanded to a belief state containing beliefs on the model of the environment and the models of other agents. The tradeoff between exploration (improving the model) and exploitation (using it) is resolved via Bayesian inference. The expected return given the current state is weighed against the possible increase in knowledge brought by taking an exploratory action. The latter component is termed "expected value of information". If the value of this component is high, the agent will be biased towards learning about the world and the other agents, at the risk of losing immediate reward, so that in the future it will be more capable to coordinate. If the value of the information is low, the agent will prefer safer choices.

While providing methods of computing the value of coordination, these approaches do so at high computational costs. This problem is accentuated for the second method, where exact inference becomes intractable even for small problems, while approximate methods become impractical for not significantly larger problems.

4.5 Social conventions

A social convention is a recipe that places constraints on the behaviour of the agents. As such, social conventions have two functions: (i) they strike a balance between the individual freedom of the agents on the one hand, and the goal of the multiagent system on the other; and (ii) they simplify the agents' decision making process (Walker and Wooldridge, 1995). From a coordination perspective, we are more interested in the latter function. When the agents are faced with choosing from a set of otherwise equally good joint actions (a *tie*), they can use social conventions to focus their choice on fewer of these joint actions. Ideally, after doing that they are left with a single joint action (i.e., no the tie has been eliminated), and coordination arises.

The typical example of a social convention in everyday life is the righthand right-of-way rule applied by drivers when arriving at an unmarked intersection. Note that this convention is not tie-free: if all the streets meeting at the intersection are occupied, a tie arises, which must be somehow broken by a mechanism other than the righthand right-of-way!

Social conventions should be rational, in order to appeal to rational agents. That is, given that the social convention is common knowledge, the agents are better off by abiding it rather than not.

As any coordination technique, social conventions can be designed offline or be learned in an online fashion from the interactions between agents.

Designed social conventions

If the action sets of the agents are discrete and common knowledge, a very simple social convention called *lexicographic ordering* can be applied (Boutilier, 1996). This convention requires that the set of agents is ordered, the action sets of the agents are ordered, and these orderings are common knowledge. By using the orderings, the agents can uniquely sort the joint actions first by agent and then by the action of each agent. Coordination is then achieved if each agent selects the first joint action in its sorted list, and executes its part of this joint action.

Learned social conventions

Typically, when talking about "conventions emergence", researchers use simple learning rules to update the agents' choice of social conventions. So, the term "emergence" is in this context just another name for learning. The work of Walker and Wooldridge (1995) falls along this line. The model used by these researchers to describe conventions emergence has already been introduced under Section 4.3. The model assumes that the set of possible conventions is common knowledge, and that at each time step agents are paired two by two and allowed to interact. During this interaction, the two agents have the chance to observe and memorize each other's current convention of choice. The memory of an agent is of finite length, and is used to trigger changes in the convention of choice of that agent, via the policy $\overline{h}_i$.

The researchers investigated a set of policies based on majority rules, in some instances also allowing the agents to communicate their memory contents. It turned out that there is a tradeoff between the speed of convergence to a social convention, and the number of convention changes incurred in the process. The number of switches is a measure of the cost of the learning process, since every switch between conventions is likely to incur a cost for the agent in real life.

Matarić (1997) casted the social conventions learning problem in the reinforcement learning framework, by interpreting social rules as actions and providing reinforcement for these actions. The power of RL was then available for learning the social conventions.

The author used a physical domain where robotic agents had to forage for "food" (pucks) over a limited area, and return the pucks to home base. The agents had to learn (along other social behaviours) a social convention for yielding in one-to-one motion conflicts. Reward was given to the agents not only for progress towards their (individual) goal, but also for:

– repeating behaviour observed in other agents – "observational reinforcement". This is a form of imitation;

– reinforcement obtained by other agents – "vicarious reinforcement". This is a heuristic solution to the structural credit assignment problem, rewarding an agent for the overall progress of the multiagent system.

Using a weighted combination of the rewards obtained from these three sources (individual, observational and vicarious), the agents successfully learned the yielding social rule. Due to the small number of RL states (interpreted from environmental cues in a designed fashion) and actions, the obtained results are not very insightful. However, the framing of learning social conventions in the RL methodology is powerful.

4.6 Roles

A role is a constraint imposed on the action space of an agent. Roles were formalized within the DMAS framework by Definition 1.11. In practice, roles reduce the number of actions that agents have to consider when taking a decision. Roles are, therefore, very similar to social conventions. The difference is that the constraints imposed by roles are applied to the action space prior to computing the set of actions eligible for execution in the current state (say, the set of equilibria of the stochastic game), whereas the constraints imposed by the social conventions are applied to this set, after it was computed. Roles have then a greater potential of simplifying the decision-making process of the agents than social conventions. This is because they reduce the size of the input to the time-consuming, actual decision making process of the agent. This process is illustrated in Figure 4.2.

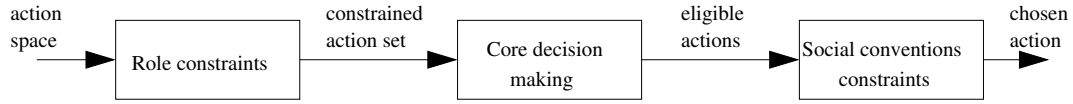


Figure 4.2: Decision making with roles and social conventions.

Designed roles

The literature dealing with offline designed roles typically associates the following elements to a multiagent system endowed with roles:

- a set of roles U^{r}, containing $m = |U^{\mathrm{r}}|$ roles.
- a priority ordering (preference ordering) over roles, $\{u_1^{\mathrm{r}}, \ldots, u_m^{\mathrm{r}}\}$. This encodes the importance of the roles in the task to be performed.

- a utility (potential) function $\rho^{\mathrm{r}} : X \times A \times U^{\mathrm{r}} \to \mathbb{R}$. The value $\rho^{\mathrm{r}}(x, i, u_j^{\mathrm{r}})$ measures how fit agent i is to fulfil role u_j^{r} in the environment state x.
- a set of formations (strategies). Every formation is associated with a different priority ordering.
- a set of rules for switching among formations.

When the literature uses several names for an element, alternatives were given in parentheses (Spaan et al., 2002; Stone and Veloso, 1999).

The rules for switching among formations are typically simple. E.g., in robotic soccer, one formation can be used when the team has possession of the ball and another when the ball is with the opposing team (Spaan et al., 2002). A larger set of environmental cues may be used to determine formations switching (Stone and Veloso, 1999).

If we assume that the roles and the utility function are common knowledge, and that the action sets are discrete, roles assignment in each state can be performed by Algorithm 4.1 (Spaan et al., 2002).

Algorithm 4.1 Role assignment for agent i

Require: roles U^{r}, utility function ρ^{r}
Input: current state x
1: $\tilde{A} \leftarrow \emptyset$
2: **for** each role u_j^{r} in the current formation $\{u_1^{\mathrm{r}}, \ldots, u_m^{\mathrm{r}}\}$ **do**
3: assign role u_j^{r} to agent $a^* = \arg\max_{\tilde{i} \in A \setminus \tilde{A}} \rho^{\mathrm{r}}(x, \tilde{i}, u_j^{\mathrm{r}})$
4: $\tilde{A} \leftarrow \tilde{A} \cup a^*$
5: **end for**

If the agents do not know each other's utilities in performing the roles, they can exchange them by communication before line 3.

Learned roles

When the roles assignment is learned, the literature does not consider priority orderings over roles. This is because in the presence of learning, priority orderings become superfluous: provided that the learning is successful, the priority orderings will be implicit in the learned role choices. This also rules out formations and the rules for switching formations.

We are left with two elements: the set of roles, and the utility function. This yields two types of role learning:

(i) learning both the roles and the utility function.

(ii) learning only the utility function.

Along the first line, there exists research showing that, under certain conditions, teams of learning agents have a tendency to evolve towards behavioural diversity (Balch, 2000). That seems to imply both that specialization in multiagent teams, i.e., roles, is beneficial in some cases, and that it can be effectively learned by the agents.

Along the second line, Prasad et al. (1998) applied utility learning to a multiagent cooperative problem solving domain. The agents learn which roles to assume in different situations,

using the Utility, Probability and Cost (UPC) framework, described below. The activity of the system is divided into a learning phase and a problem solving phase. During the first phase, agents learn appropriate situation-based role assignments, which are then applied in actual problem solving.

We will regard the problem from the perspective of one agent perspective and omit the agent index. States are mapped into a smaller number of situations, X. The agent can fulfill any of a set of roles U^{r}. It maintains tables of utilities, probabilities, costs and potentials for each role in each situation. The utility $U(x, u^{\mathrm{r}})$ is the agent's estimate of the final state's expected value given that it selects role u^{r} in situation x. The probability $P(x, u^{\mathrm{r}})$ represents the uncertainty of the agent that the final state will be reached from x given that it chooses role u^{r}. The cost $C(x, u^{\mathrm{r}})$ represents the expected computational cost of reaching the final state from x after choosing u^{r}. The potential of a role $\pi(x, u^{\mathrm{r}})$ estimates the usefulness of a role in discovering pertinent global information and constraints. This measure is strongly related to the value of coordination: if $\pi(x, u^{\mathrm{r}})$ is high, it biases the agent towards discovering new information on the problem, risking its short-term performance in the process.

An objective function g is used to evaluate a role choice given its utility, probability, cost and potential values. This function corresponds to the role utility function ρ^{r} introduced above. During learning, this function is used to choose roles via a roulette wheel selection mechanism. The probability of choosing role u^{r} in situation x is:

$$h(x, u^{\mathrm{r}}) = \frac{g\left(U(x, u^{\mathrm{r}}), P(x, u^{\mathrm{r}}), C(x, u^{\mathrm{r}}), \pi(x, u^{\mathrm{r}})\right)}{\sum_{u^{\mathrm{r}\prime} \in U^{\mathrm{r}}} g\left(U(x, u^{\mathrm{r}\prime}), P(x, u^{\mathrm{r}\prime}), C(x, u^{\mathrm{r}\prime}), \pi(x, u^{\mathrm{r}\prime})\right)}. \tag{4.5}$$

An agent learns U, P, and π estimates in a Monte Carlo fashion (cost is disregarded in the objective function in this particular work) (Prasad et al., 1998). After each problem solving instance in the learning phase, the agent updates the estimates for all situation-role pairs (x, u^{r}) encountered en route to the final (solution) state F by:

$$U(x, u^{\mathrm{r}}) \leftarrow U(x, u^{\mathrm{r}}) + \alpha(U_F - U(x, u^{\mathrm{r}})) \tag{4.6}$$

$$P(x, u^{\mathrm{r}}) \leftarrow P(x, u^{\mathrm{r}}) + \alpha(O_F - P(x, u^{\mathrm{r}})) \tag{4.7}$$

$$\pi(x, u^{\mathrm{r}}) \leftarrow \pi(x, u^{\mathrm{r}}) + \alpha(\pi_F - \pi(x, u^{\mathrm{r}})), \tag{4.8}$$

where U_F is the utility of the final state; O_F is 1 if the final state was a success and 0 if it was a failure; and π_F is 1 if conflicts between agents followed by information exchange were encountered on the path to the solution, and 0 if not. The constant α is a learning rate.

In the problem solving phase, the agent uses greedy role choice on the learned values:

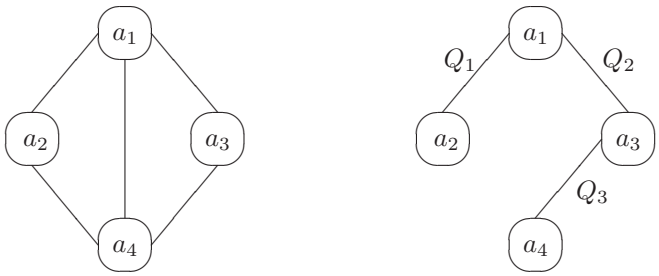
$$h(x) = \arg\max_{u^{\mathrm{r}} \in U^{\mathrm{r}}} g\left(U(x, u^{\mathrm{r}}), P(x, u^{\mathrm{r}}), C(x, u^{\mathrm{r}}), \pi(x, u^{\mathrm{r}})\right). \tag{4.9}$$

This approach resembles reinforcement learning in the following ways. The learning selection rule (4.5) can be thought of as an exploratory policy related to softmax, and (4.9) is, in fact, a greedy policy in the objective function. Information about the joint utility of roles selection is encoded in the targets used by the updates. However, the value estimates take into account only the role choice (action) of one agent. With greedy action selection, the method probably suffers from threats to convergence similar to those that affect basic Q-learning in multiagent contexts, especially if couplings between agents are significant, which is typically the case in task allocation.

4.7 Coordination graphs

In almost all realistic situations, an agent does not have to consider the entire joint action and coordinate with all other agents, in all the environment states. Rather, given the current environment state, only a small set of the other agents influences the outcomes of the agent's actions. Moreover, each agent may be interested in only certain components of the environment state vector.

Coordination graphs are a model used to represent situations where these assumptions hold (e.g., Guestrin et al., 2002). They are typically placed on top of MMDPs as the underlying task model. A coordination graph has agents in the nodes, and the arcs express the coordination relationships between agents: if two agents are connected by an arc, they need to directly coordinate their actions. An example is given in Figure 4.3(a). Without knowing the environment state, agent a_1 knows that it needs to coordinate with a_2, a_3 and a_4, agent a_2 needs to coordinate with a_1 and a_4, and so on.



(a) Prior to conditioning on the state. (b) After conditioning on the state.

Figure 4.3: An example coordination graph.

After conditioning on the current environment state x, the coordination graph might reduce to something like Figure 4.3(b) (henceforth the example is borrowed from Kok et al. (2005a)). The functions Q_i are the local expected returns (Q-functions) and specify the coordination dependencies between agents: e.g., Q_1 is the local Q-function involving only agents a_1 and a_2.

These local expected returns are components of the global Q-function of the MMDP in state x. Since in an MMDP the reward of the system is the sum of the agents' rewards, the global Q-function is the sum of the local Q-functions:

$$Q(x, \boldsymbol{u}) = Q_1(u_1, u_2) + Q_2(u_1, u_3) + Q_3(u_3, u_4), \tag{4.10}$$

where, of course, $\boldsymbol{u} = [u_1, u_2, u_3, u_4]^{\mathrm{T}}$. We omit the state variable in the local Q-functions since they are already conditioned on the current state x.

An optimal action choice $\boldsymbol{u}^*$ that maximizes Q can be found by variable elimination. This proceeds as follows: at each step, an agent is chosen for elimination. This agent collects the local Q-functions he is involved with by communication with its neighbours, and then computes an optimal strategy that is conditioned on the action choices of the rest of these agents. Since the overall strategy no longer depends on the chosen agent, it can be eliminated from the graph. The process continues iteratively until only one agent remains, the action of which can be trivially chosen to maximize the return. In reverse order of elimination, agents choose their actions using their conditional strategies, given the actions chosen by their predecessors.

The optimal joint action found in this way does not depend on the order in which agents are eliminated. The efficiency of the algorithm, however, does.

In the example of Figure 4.3(b), after being chosen for elimination, agent a_1 in Figure 4.3(b) collects the local Q-functions Q_1 and Q_2 in which he is involved, and maximizes over their sum:

$$\max_{\boldsymbol{u}} Q(x, \boldsymbol{u}) = \max_{u_2, u_3, u_4} \left[Q_3(u_3, u_4) + \max_{u_1} \left[Q_1(u_1, u_2) + Q_2(u_2, u_3) \right] \right]. \tag{4.11}$$

The conditional strategy of agent a_1 corresponds to the inner maximization term:

$$Q_4(u_2, u_3) = \max_{u_1} \left[Q_1(u_1, u_2) + Q_2(u_2, u_3) \right]. \tag{4.12}$$

The agent computes and communicates the new Q-function $Q_4(u_2, u_3)$, essentially committing to execute the action that maximizes this function once a_2 and a_3 have chosen their actions. The relation (4.11) can now be written as:

$$\max_{\boldsymbol{u}} Q(x, \boldsymbol{u}) = \max_{u_2, u_3, u_4} \left[Q_3(u_3, u_4) + Q_4(u_2, u_3) \right]. \tag{4.13}$$

Since only Q_4 depends on u_2, agent a_2 computes $Q_5(u_3) = \max_{u_2} Q_4(u_2, u_3)$. The relation (4.13) can be written as:

$$\max_{\boldsymbol{u}} Q(x, \boldsymbol{u}) = \max_{u_3, u_4} \left[Q_3(u_3, u_4) + Q_5(u_3) \right]. \tag{4.14}$$

By elimination of agent a_3 via $Q_6(u_4) = \max_{u_3} \left[Q_3(u_3, u_4) + Q_5(u_3) \right]$, (4.14) reduces to:

$$\max_{\boldsymbol{u}} Q(x, \boldsymbol{u}) = \max_{u_4} Q_6(u_4), \tag{4.15}$$

that is a straightforward maximization.

After a_4 chooses $u_4^* = \arg\max_{u_4} Q_6(u_4)$, it communicates this choice to the other agents. Agent a_3, binding u_4 to u_4^* in (4.14), computes u_3^* via a simple maximization, and communicates it. This backward pass continues up to agent a_1, at which moment the optimal joint action $\boldsymbol{u}^*$ is determined.

The selective dependence on state and actions of an agent's returns, central to the coordination graphs approach, can be conveniently represented by so-called "value rules" (Kok et al., 2005a). These rules condition the returns of the agent on the environment state and the actions of the other agents. For instance, a rule describing that when agent a_1 is very close to a wall (say, lying along the vertical coordinate 10) and moves north, it will bump, would look something like:

$$\text{if } Y_1 > 9.9 \text{ and } u_1 = \texttt{move-north} \text{ then } q_1 = -10, \tag{4.16}$$

where the vertical coordinate Y_1 is a component of the environment state x. The return of the agent can then be obtained by summing the returns of the value rules whose condition parts are satisfied.

The variable elimination algorithm described above is used to ensure a consistent selection of the optimal joint action given the state of the game. So, its function is similar to that of social conventions. This means that, in a manner similar to that presented in Figure 4.2, the

problem can be further simplified by combining coordination graphs with roles (Kok et al., 2005a). Roles can be assigned to the agents by Algorithm 4.1.

If the agents are assigned roles, then a further conditioning of the coordination graph on the role restrictions follows after that on the environment state in Figure 4.3(b). Alternatively, we can say that a role term is added to the condition part of all the value rules of the agent. The process is represented schematically in Figure 4.4.

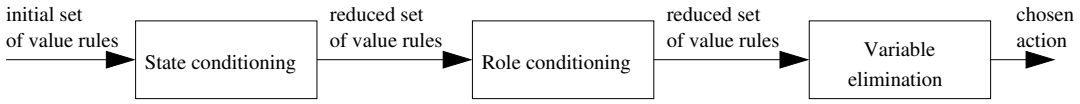


Figure 4.4: Decision making with roles and coordination graphs.

Designed coordination graphs

If the coordination relationships between the agents are known a priori, the coordination graph can be designed offline and wired into the agents. This is the approach taken by Guestrin et al. (2002) and by Kok et al. (2005a).

The variable elimination algorithm presented above requires communication between the agents. However, since the resulting joint action does not depend on the order in which agents are eliminated, with additional assumptions variable elimination can be run in parallel at all agents (Kok et al., 2005a). These assumptions are the following:

- There exists an lexicographic ordering of actions, and this ordering is common knowledge among all agents. This ordering is used to break ties in a consistent manner during the backward pass of variable elimination.

- The value rules of an agent are common knowledge among all agents connected to that agent in the coordination graph.

- An agent observes the components of the state that appear in the condition part of the value rules of all the agents it is connected to in the coordination graph.

Learned coordination graphs

Kok et al. (2005b) present a method for learning coordination relationships when they are not available a priori. If no prior knowledge is available, the initial coordination graph will have no arcs, and the agents will be independent learners. If some of the coordination relationships can be guessed, as it will most likely be the case in many situations, then the coordination graph can be initialized to capture them, and be afterwards extended by learning.

The agents create hypothetical arcs in the coordination graph, and store statistics of expected returns for these hypothetical arcs. Periodically, the agents examine the stored statistics, analyzing whether the difference between the obtained returns with the hypothetical arc in place, and without it. If the difference is statistically significant, then there is benefit in considering the corresponding coordination relationship, so it is inserted into the coordination graph.

4.8 Concluding remarks

We have reviewed in this chapter some representative frameworks and the most relevant techniques for coordinating the activity of a multiagent system. Many of the approaches presented here look at coordination as a distinct process, separate from learning or added on top of it. Indeed, coordination issues can arise both in the presence and absence of learning. We believe though that the two activities are tightly interrelated, being both manifestations of the effort of rational agents towards their goals. We therefore believe that looking at them from a unified perspective can be very helpful.

The framework of coordination graphs is an important step in this direction. Coordination graphs are a tool for coordination, but can also be used as a representation of the multiagent learning task (Kok and Vlassis, 2004). We hope that our DMAS framework is a complementary step. Viewing both learning and coordination as part of an agent's dynamics should facilitate the interpretation and analysis of these processes and of their interconnection.

Some important research opportunities stem from the agent modeling area. Most of the approaches have a common shortcoming: they consider the modeled agent to be a static, reactive system. At most, recursive modeling looks considers agents that in their turn model other agents. The static agent assumption is violated in most settings. An immediate and common example where it fails is when a modeled agent is also a learning agent. Furthermore, most agent modeling approaches use black-box models, with the notable exception of recursive modeling where a simple linear policy is assumed (Hu and Wellman, 2001) (see Section 4.4.1). This ignores cooperative situations where (perhaps rough) models might be known by the agents.

The solution to the first problem is using a model that is able to represent agent dynamics. The solution to the second problem is using grey-box modeling, where the model is assumed partly known. At least two possibilities come to mind:

- If the structure of the model is known (grey-box modeling), filtering approaches could be used.
- If nothing is known about the modeled agent (black-box modeling), dynamics might still be captured by a neural network.

Further opportunities for learning

Coordination mechanisms always add an overhead to decision making (in terms of processing, communication, or both). This overhead is more significant for techniques that share little information between agents, such as learning of coordination (see e.g., Chalkiadakis and Boutilier, 2003). Therefore, in many situations it is desirable that the agents ask not only (i) "How to coordinate?" and (ii) "What is the value of coordination?", but also (iii) "Where to coordinate?". Specifically, the agents should be very selective with the situations where they apply coordination mechanisms.

One flexible way of doing this is the online discovery of coordination needs. This falls under the term of structural adaptation, and we have seen an example in Section 4.7 (Kok et al., 2005b). Another work in this direction is (Buşoniu et al., 2005). These are just first steps, and the existence and form of general criteria answering question (iii) should be investigated further.

Chapter 5

Multiagent application domains

5.1 Introduction

We review in this chapter the specific application domains used by the multiagent system (MAS) literature as test-beds for learning and coordination approaches. We select the most promising domains that could be used to validate our research into multiagent learning and coordination. We identify the specific challenges posed by each of these domains, and investigate how well it fits in the large-scale traffic incident scenario of the Interactive Collaborative Information Systems (ICIS) project.

The task that needs to be performed by the MAS influences its position along the taxonomy dimensions introduced in Section 1.3. It influences, of course, the environment-related dimensions: whether (and to what degree) communication is available, whether (and to what degree) the environment is observable, and whether the agents have local or global perspectives. Moreover, the objective of the task translates into the goals of the agents, and therefore influences the degree of cooperation between the agents. The task might also imply design considerations that restrict choices along the rest of the dimensions (agent heterogeneity, reactivity vs. deliberation) but at this point we do not consider these aspects.

Since we are interested in cooperative agent teams, we will focus on cooperative tasks. We bias the review towards partially observable tasks, but that allow in a first approximation the assumption of complete observability. We do not impose restrictions along the other dimensions, but mention any such restrictions implied by the reviewed domains.

5.2 Robotic teams

Robotic teams (also identified as "multi-robot systems" by the literature) are probably the most widely used MAS application domain. Each agent is in this case a robot, and the task is instantiated on a spatial domain, most often having two dimensions.

The multi-robot domain can be real or simulated. The most interesting advantage of the simulated version is that the difficulty of the task is under the control of the designer. Other advantages are the lower cost and greater speed of the experiments. The simplest abstraction of a multi-robot domain is the two-dimensional, rectangular gridworld. It is especially popular in the multiagent reinforcement learning literature (e.g., Hu and Wellman, 2003; Bowling and Veloso, 2002). Many varieties and extensions exist, such as:

– toroidal world, where advancing beyond the edge returns the agent on the opposite side (e.g., Kok et al., 2005b);

– continuous coordinates (e.g., Sen et al., 1994);

– circular (e.g., Touzet, 2000) or arbitrarily shaped world (the latter especially useful in multiagent localization and mapping).

In what follows, we list the most relevant multi-robot tasks studied in the literature. We use the terms "agent" and "robot" interchangeably. Since the literature typically uses two-dimensional tasks, we describe them in two-dimensional terminology. Extensions to three dimensions are however possible for all these tasks.

Navigation

The goal of the navigation task is for each agent to find its way from some starting position to some final, goal position, while avoiding harmful interference with other agents (e.g., Hu and Wellman, 2003; Bowling and Veloso, 2002). The goal position may be fixed or may change dynamically. The importance of the navigation task is obvious, and this task appears, either implicitly or explicitly, as a basic building block of all other multi-robot tasks.

Area Sweeping

This task involves navigating through the multi-robot environment for either observation or exploitation purposes. Several kinds of related area sweeping tasks exist:

- *Retrieval:* retrieving objects from the environment (e.g., Matarić, 1996).
- *Coverage*: navigating over as much surface from the environment as possible, for instance for the purpose of harvesting resources or cleaning (e.g., Balch and Arkin, 1994).
- *Exploration*: observing as much surface of the environment as possible. The difference from coverage is that the agents need not actually reach every point of the environment, but only bring it in sensor range. Note that retrieval implicitly involves an initial (if the objects are grouped) or permanent (if the objects are scattered) exploration component.

Both coverage and exploration have continuous variants, where the agents need to minimize the time elapsed between two consecutive passes over (observations of) every point in the environment.

Multi-target observation

The multi-target observation task is actually an extension of the exploration task, where the goal of the agents is to maintain within sensor range a group of targets that are themselves in motion (e.g., Touzet, 2000).

Object transportation

The object transportation task, as its name implies, requires the relocation of a set of objects into given final positions and configuration. Many times the method of moving an object is pushing it. This task is especially useful for studying coordination, since the mass or other

properties of some of the objects may exceed the transportation capabilities of one agent, thus requiring several agents to coordinate in order to bring about the objective (e.g., Matarić, 1996).

Robotic soccer

Robotic soccer is a very popular test-bed for multiagent learning and coordination approaches, both in its simulated and real variants (Stone and Veloso, 2000). The reason is that it combines all the tasks presented above, possibly adding other things as well (such as limited communication, leading to the necessity of selectivity in exchanging messages). It involves for instance object retrieval and transportation (intercepting the ball and leading it into the goal), multi-target observation (keeping the adversary team under observation), and an advanced version of coverage task – the strategic dynamic placement of the players on the field.

Lately, robotic soccer competitions are complemented by a rescue competition, where the objective is for the team of robots to salvage victims from simulated disaster areas.

Attack and pursuit

Attack and pursuit tasks stem from research in the military field, and involve the destruction or capture of static and / or moving enemy targets by the robotic teams. This type of task involves a special case of foraging, where the objects are attacked upon discovery. Such offensive tasks may also conceivably include a phase of stealth scouting, where multi-target observation is the main component.

One example of such a task is the helicopter attack domain (Tambe, 1997). Another is the pursuit domain, where several "predator" agents in a gridworld have to capture a moving "prey" agent by converging on it (e.g., Kok et al., 2005b). This latter domain is a very popular type of gridworld task. The team of predators contains either two or four predators; and typically a single prey agent exists.

The challenges posed by robotic team domains can include (and in real domains, typically *will* include):

- partial observability (due to the limited number of sensors and their limited range);
- uncertain observations and uncertain effect of robot actions (due to noisy sensors and imperfect actuators);
- limited communication;
- time constraints on the decision-making process;
- limited resources available to a robot's control program.

In a disaster scenario such as the ICIS large scale traffic incident, the main application of multi-robot systems is to robotic search and rescue teams. Robotic airborne scouts may be the first to observe the scene of the disaster; rescue robots may be the first to advance in the dangerous area where the incident occurred. In fact, rescue robots might be the only relief that can reach the incident area for a very long time; for instance, the Mont-Blanc tunnel fire raised the temperature in the tunnel so much that manned rescue teams were unable to get inside for several hours.

The following skills might be useful for the robotic search, rescue and assistance teams:

- Navigation skills will presumably be needed in the incident area due to the agglomeration of vehicles and debris.
- Exploration, foraging and transportation skills can be used to retrieve victims and return them to a safe area for medical assistance.
- Multi-target observation by airborne scouts can be used to monitor and optimize the activity of the manned and robotic rescue teams.

Though soccer and attack teams are not immediately applicable to the rescue task, they bear strong resemblance and pose similar challenges with this task. Therefore, results of research in these fields, with modifications, are applicable to robotic search and rescue teams.

5.3 Distributed control

Distributed control is an approach that separates the control task into lower-level subtasks to be realized by controllers that are distinct, have a significant degree of autonomy, and are interconnected in a network. The plant to be controlled is typically large and complex. The control structure may be completely distributed, if all the controllers interact directly with the plant, or it can include hierarchy. In the latter case, controllers at upper levels in the hierarchy look at a coarser grain of the control task, and communicate coarse decisions to lower-level controllers that carry them out.

In the multiagent context, the controllers are the agents, and the plant is the environment (see Chapter 1). In this view, all the multiagent applications described in this chapter are, in effect, distributed control systems. For instance, in the case of object transportation by robotic teams, the plant is the agents' world containing the objects to transport, and the control goal of the agents is moving the objects in the desired final positions and configuration.

Distributed control typically places the agents at fixed locations in the topology of the system. A representative and relevant example of distributed control is the traffic control system. The process is in this case the flow of traffic on highways, ring roads, and streets, and the control goal is, roughly speaking, to maintain the traffic flow as fast, smooth, and safe as possible. The observations of the agents include e.g., traffic density, average vehicle speed, time of day, and the control outputs include e.g., traffic signals and dynamic route information panels. Agents may be placed e.g., at intersections or allocated to highway sectors.

Other examples of distributed control applications are power networks, communication networks, and flexible manufacturing systems.

The challenges posed by distributed control include:

– hard real-time constraints on the control decisions;

– the need for efficient and reliable coordination mechanisms, so that coherent joint control decisions are taken;

– the need of obtaining (at least rough) process models in order to achieve the desired control performance.

We find traffic control to be of use in the ICIS scenario in two critical aspects:

- An efficient traffic control system can be used to *prevent* many of the possible traffic incidents by avoiding dangerous situations such as traffic jams or high vehicle velocities in bad weather.

- If the incident could not be avoided, traffic control can *reroute* the flow of traffic out of and around the incident area, such that the size of the incident is kept under control, and the access of relief teams is not impeded.

5.4 Logistics

Logistics is the process of controlling the flow and usage of resources within a system so that the activities of the system are properly supplied with the needed resources. So, when applied to logistics, agents manage resources.

We differentiate two closely related types of logistics in the multiagent context, by the nature of the controlled resources.

- Actual *resource management*: the resources are passive tools used by the agents in their work. Examples are load balancing (Schaerf et al., 1995), scheduling (Crites and Barto, 1998), and routing (Boyan and Littman, 1993).
- *Workflow management*: the resources are the agents themselves. They decompose the task of the MAS into smaller subtasks and synchronize their work on the subtasks (Parker, 1997). When the decomposition into subtasks is already given, workflow management reduces to a simpler problem called *task allocation*.

Typically, schedules for resource allocation are computed and enforced by a central authority. MAS offer a robust alternative, by distributing the resource management process among the agents. Moreover, all the approaches cited above use learning, so that the scheduling agents do not require complete information on the problem.

The challenges posed by the logistics domain to multiagent systems include:

– highly-dimensional search spaces with constraints;

– the need for an efficient multiagent coordination mechanism, so that partial (agent) solutions form a coherent combined (MAS) solution;

– incomplete information available to the agents in order to develop a solution.

MAS could control partially or completely the flow of resources (such as ambulances and fire-trucks) around and within the incident area in the ICIS scenario. They could also use workflow management to sequence and synchronize their own activity.

5.5 Automated information systems

An information system collects, processes, transmits and disseminates data. An agent-based automated information system has agents controlling some or all of these stages. We can therefore view an automated information system as a special case of distributed control system, where the controlled process is the flow of data among the information system components.

An essential characteristic of the information system is the extensive use of communication.

The applications of automated information systems include among others purely informational systems such as database access and electronic commerce. We are however more interested in physical applications of information systems, such as smart sensor networks (Lewis, 2004).

The sensing equipment in a smart sensor network not only collects the data, but also performs some or all of the following functions:

– initial stages of data processing;

– decision making, e.g., in order to activate alarms;

– self-diagnosis;

– communication and coordination with other nodes.

These functions work towards a common goal of the sensor network (Lewis, 2004). Thus, the sensor network can be seen as a common-goal, cooperative, and communicative multiagent system.

The challenges posed by information systems include (Singh and Huhns, 1996):

– coping with the open and unpredictable nature of the environment (adding and removing components is part of the normal functioning of the system);

– handling an unusually high degree of agent heterogeneity.

Smart sensor networks could be used in the ICIS traffic incident scenario to monitor the traffic infrastructure, notably sensitive areas such as tunnels. The smart sensor network, possibly working in tandem with the traffic control system, could perform the following functions:

- Collecting and supplying data to other systems, among which the traffic control system.
- Being the first line of reaction against the incident, it would alarm relief services such as medical and fire assistance, and take immediate countermeasures such as activating smoke ventilation shafts.
- Collaborating with the traffic control system to quickly direct unaffected vehicles out of the incident area.

5.6 Concluding remarks

We have reviewed in this chapter relevant applications of multiagent learning and coordination in the context of the ICIS traffic incident scenario: multirobot teams, distributed control, logistics, and automated information systems. Other applications might also be useful. E.g., auctions could be used in resource management, or even the application to games could provide valuable insight. These two examples are, however, less relevant for the coordinated, cooperative teams we focus on.

In a unified view, all the applications above could be integrated in a single distributed system, managing and monitoring the disaster relief effort. Traffic control, using data supplied by the smart sensor network, would monitor and regulate traffic. If an incident occurs, these two components would collaborate in rerouting traffic out of and around the area, while simultaneously alerting relief services. Robotic and human search and rescue teams would be dispatched to the area, where the former would function autonomously using robotic team skills. Their tasks and serviced areas would be assigned by the automated resource and workflow management system.

Keeping realistic, however, we target one, or perhaps two integrated components of this system. Other relevant applications, not foreseen at this point, are not excluded either.

As to an immediate research avenue, we have not yet encountered in the literature extensions of the gridworld to three dimensions. This extension would be useful in simulating robotic agents such as autonomous underwater vehicles, unmanned air vehicles, and even ground robots on uneven surfaces.

Chapter 6

Conclusions

We have reviewed in this work frameworks and methods for learning and coordination in multiagent systems. In this chapter, we state our main conclusions on these topics, and identify a set of useful research questions and directions.

Reinforcement learning is the prevalent learning technique in MAS. However, the field of multiagent RL has not yet reached maturity. Its most important problems are the lack of a widely accepted problem statement, and the gap between theory and practical applications. Theoretical methods are strongly influenced by game theoretical, stateless solutions, the meaning of which is unclear in the multiple-state, delayed reward MA-RL setting. Theoretical methods also make strong, restrictive assumptions that are difficult to justify in practice. On the other hand, many practical applications are heuristic in nature, and thus difficult to analyze. This latter statement applies to the other reviewed multiagent learning techniques, as well.

Coordination is an essential part of learning; coordination can also be learned by the agents. Most of the reviewed coordination techniques look at coordination as a stage-wise activity, similarly to MA-RL methods. One consequence of this is that most agent modeling methods consider the agents stateless, reactive entities.

It is our opinion that in order to understand multiagent learning, and its relationship with coordination, it is essential that the dynamic nature of the agents and their environment is taken into account. This involves considering rich state spaces, and state evolution through time, both for the environment and for the agents. We believe that this dynamic nature, and the effects of placing multiple dynamic agents together in a dynamic environment, are not necessarily a daunting difficulty to overcome, as seems to be the prevailing view in the literature. Looking at the problem from this perspective could provide insight into new solutions, where agents make use of the fact that they are part of a dynamic multiagent system, instead of trying to circumvent it.

This issue can be stated succinctly as the following research question:

- How can a dynamic agent best make use of the fact that it is part of a dynamic multiagent system?

We have already suggested some research directions deriving from this question. One of them is using dynamic, black- or grey-box agent models, such as neural networks and Kalman or particle filters (Section 4.8). Similarly, observers could be used to estimate the state of a partially measurable, but observable environment (Section 2.9).

Another research avenue in this context is using control- and system-theoretic methods for analyzing the properties of learning and coordination techniques, e.g., stability and robustness to changes in the environment. Relevant approaches could be found in the area of (adaptive) distributed control of large-scale systems.

One way of using the MAS context, explored in the MAS learning literature, is teaching and imitation. The investigation of the suitability of stage-wise learning and coordination methods in the DMAS is also related to this research question.

Another research question arises from the controversy in the literature with respect to the MA-RL goal. In addition to the aspects mentioned in the literature and reviewed in Section 2.3.3, we believe that the encountered learning goals are biased towards optimality, without considering other aspects of the agent behaviour. Moreover, results are many times given in terms of asymptotic or average performance, without concern for the agent's transient or short-term performance.

Desirable properties of the agent's behaviour, besides optimality, include:

– robustness against disturbances and changes in the environment.

– approximate monotonicity of agent performance. We consider approximate monotonicity because of the exploration issue: the agent may accept temporary losses in performance in order to gain knowledge on the problem. We stress, however, that these temporary losses should be bounded. Most of the literature is not at all concerned with this aspect.

– satisfying given lower bounds on the performance.

In brief:

- What is a suitable multiagent learning goal? How can it include performance requirements other than optimality?

We mentioned in Section 2.9 that an important difficulty in MA-RL is the explosion of the state space with the number of agents. We also observed that the agents must not always take into consideration the entire state of the world, nor the actions of all the other agents. We concluded that learning should be focused on interesting parts of the state space – i.e., it should be "local" as much as possible. In addition to reducing computational requirements, this selective focus should also speed up learning, due to the smaller size of the search space.

Similarly, coordination should be restricted to the regions of the state space where it is necessary Section 4.8. These are actually two aspects of the same question:

- Do there exist general conditions to determine whether a learning agent can perform well while focusing on a restricted region of the state (and action) space? If they do exist, then what are they?

These conditions, if they exist, are likely to give different answers at different moments in time, as the agent learns and the world evolves. This implies that the agent must adapt its focus over time (Buşoniu et al., 2005).

Focusing on restricted regions of the state space would imply in many cases that the agent ignores a part of this sensory input. However, such conditions, when they can be analyzed over the entire task span, could also be used the other way around: to determine whether the agent can perform well given its limited sensory input; or, to determine what is the minimal set of sensors with which the agent has to be endowed in order to perform well at its task.

A related research question is:

- How can an agent make use most effectively of incomplete and / or uncertain information?

Many times, focusing on restricted regions of the state space will lead to incompleteness and uncertainty in the information considered by the agent. However, incompleteness and uncertainty might be natively present in the system, due to e.g., limited and noisy sensors.

We are especially interested in this issue in the context of modeling: an agent wishes to construct and use models of the environment or of other agents. The literature typically takes one of two approaches: either the uncertainty is ignored and the model output is considered the true value of the estimated variable, or a Bayesian reasoning framework is used, and the model outputs a distribution over all the possible variable values. Ignoring uncertainty might be a dangerous thing to do. On the other hand, Bayesian reasoning is highly computationally intensive. Intermediate methods could be investigated, e.g., models that in addition to estimates also output confidence levels for those estimates.

The methods could be used not only in the online construction of models, but also in conveying limited and imperfect prior knowledge to the agents, via imperfect models.

Finally, we need a framework of determining whether adaptation of "static" learning is necessary, given the problem:

- Do general conditions exist that, given the learning problem, could be used to determine whether adaptation of the learning processes of the agents is necessary? If they exist, what are they?

Given that adaptive learning can always be restated as static learning, (see Section 1.2), the answers to this question could be used to determine, given the problem, whether learning is necessary or a static agent behaviour will suffice.

6.1 Research agenda

We conclude this work by enumerating the research questions identified above, in the order of relevance:

1. Do general conditions exist that, given the learning problem, could be used to determine whether adaptation of the learning processes of the agents is necessary? If they exist, what are they?

2. What is a suitable multiagent learning goal? How can it include performance requirements other than optimality?

3. How can a dynamic agent best make use of the fact that it is part of a dynamic multiagent system?

4. Do there exist general conditions to determine whether a learning agent can perform well while focusing on a restricted region of the state (and action) space? If they do exist, then what are they?

5. How can an agent make use most effectively of incomplete and / or uncertain information?

Bibliography

Başar, T. (1985). An equilibrium theory for multiperson decision making with multiple probabilistic models. *IEEE Transactions on Automatic Control*, 30(2):118–132.

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)*, pages 30–37, Tahoe City, California, USA.

Baird, L. and Moore, A. (1998). Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing 11 (Neural Information Processing Systems 1998, NIPS-98)*, pages 968–974, Denver, Colorado, USA.

Balch, T. R. (2000). Hierarchic social entropy: An information theoretic measure of robot group diversity. *Autonomous Robots*, 8(3):209–238.

Balch, T. R. and Arkin, R. C. (1994). Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):27–52.

Banerjee, B. and Peng, J. (2003). Adaptive policy gradient in multiagent learning. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, pages 686–692, Melbourne, Australia.

Bonarini, A. and Trianni, V. (2001). Learning fuzzy classifier systems for multi-agent coordination. *Information Sciences*, 136:215–239.

Booker, L. B. (1988). Classifier systems that learn internal world models. *Machine Learning*, 3:161–192.

Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge (TARK-96)*, pages 195–210, De Zeeuwse Stromen, The Netherlands.

Boutilier, C. (1999). Sequential optimality and coordination in multiagent systems. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 478–485, Stockholm, Sweden.

Bowling, M. (2004). Convergence and no-regret in multiagent learning. *Advances in Neural Information Processing Systems*, 17:209–216.

Bowling, M. and Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250.

Boyan, J. A. and Littman, M. L. (1993). Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems 6 (Neural Information Processing Systems 1993, NIPS-93)*, pages 671–678, Denver, Colorado, USA.

Buşoniu, L., De Schutter, B., and Babuška, R. (2005). Multiagent reinforcement learning with adaptive state focus. Accepted at the 17th Belgian-Dutch Conference on Artificial Intelligence (BNAIC-05).

Chalkiadakis, G. and Boutilier, C. (2003). Coordination in multiagent reinforcement learning: A Bayesian approach. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, pages 709–716, Melbourne, Australia.

Clouse, J. (1995). Learning from an automated training agent. In *Working Notes of the Workshop on Agents that Learn from Other Agents, Twelfth International Conference on Machine Learning (ICML-95)*, Tahoe City, California, USA.

Conitzer, V. and Sandholm, T. (2003). AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, pages 83–90, Washington, DC, USA.

Crites, R. H. and Barto, A. G. (1998). Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33(2–3):235–262.

De Jong, E. (1997). An accumulative exploration method for reinforcement learning. In *Notes of the Workshop on Multiagent Learning, The 14th National Conference on Artificial Intelligence (AAAI-97)*, Providence, Rhode Island.

Dorigo, M. and Bersini, H. (1994). A comparison of Q-learning and classifier systems. In *From Animals to Animats 3. Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior (SAB-94)*, pages 248–255, Brighton, United Kingdom.

Fulda, N. and Ventura, D. (2003). Dynamic joint action perception for Q-learning agents. In *Proceedings of the 2003 International Conference on Machine Learning and Applications (ICMLA-03)*, pages 73–79, Los Angeles, California, USA.

Greenwald, A. and Hall, K. (2003). Correlated-Q learning. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, pages 242–249, Washington, DC, USA.

Guestrin, C., Lagoudakis, M. G., and Parr, R. (2002). Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML-02)*, pages 227–234, Sydney, Australia.

Haynes, T. and Sen, S. (1996). Evolving behavioral strategies in predators and prey. In Weiß, G. and Sen, S., editors, *Adaptation and Learning in Multi-Agent Systems*, pages 113–126. Springer.

Haynes, T., Wainwright, R., Sen, S., and Schoenefeld, D. (1995). Strongly typed genetic programming in evolving cooperation strategies. In *Proceedings of the 6th International Conference on Genetic Algorithms (ICGA-95)*, pages 271–278, Pittsburgh, Philadelphia, USA.

Hu, J. and Wellman, M. P. (2001). Learning about other agents in a dynamic multiagent system. *Journal of Cognitive Systems Research*, 1:67–79.

Hu, J. and Wellman, M. P. (2003). Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069.

Jung, H., Nair, R., Tambe, M., and Marsella, S. (2002). Computational models for multiagent coordination analysis: Extending distributed POMDP models. In *Proceedings of the 2nd International Workshop on Formal Approaches to Agent-Based Systems (FAABS-02)*, pages 103–114, Greenbelt, Maryland, USA.

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

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

Kok, J. R., 't Hoen, P. J., Bakker, B., and Vlassis, N. (2005b). Utile coordination: Learning interdependencies among cooperative agents. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG-05)*, pages 29–36, Colchester, United Kingdom.

Kok, J. R. and Vlassis, N. (2004). Sparse cooperative Q-learning. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML-04)*, pages 481–488, Banff, Canada.

Könönen, V. (2003). Asymmetric multiagent reinforcement learning. In *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology (IAT-03)*, pages 336–342, Halifax, Canada.

Lauer, M. and Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-00)*, pages 535–542, Stanford University, Stanford, California, USA.

Lewis, F. L. (2004). *Smart Environments: Technologies, Protocols, and Applications*, chapter Wireless Sensor Networks. John Wiley, New York.

Littman, M. L. (2001a). Friend-or-foe Q-learning in general-sum games. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML-01)*, pages 322–328, Williams College, Williamstown, Massachusets, USA.

Littman, M. L. (2001b). Value-function reinforcement learning in Markov games. *Journal of Cognitive Systems Research*, 2:55–66.

Littman, M. L. and Stone, P. (2001). Implicit negotiation in repeated games. In *Pre-proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, pages 96–105, Seattle, Washington, USA.

Matarić, M. J. (1996). Learning in multi-robot systems. In Weiß, G. and Sen, S., editors, *Adaptation and Learning in Multi–Agent Systems*, pages 152–163. Springer Verlag.

Matarić, M. J. (1997). Learning social behavior. *Robotics and Autonomous Systems*, 20:191–204.

Moore, A. W. and Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130.

Parker, L. E. (1997). L-ALLIANCE: Task-oriented multi-robot learning in behavior-based systems. *Advanced Robotics*, 4(11):305–322. Special Issue on Selected Papers from the 1996 International Conference on Intelligent Robots and Systems.

Peng, J. and Williams, R. J. (1996). Incremental multi-step Q-learning. *Machine Learning*, 22(1-3):283–290.

Plaza, E. and Ontañón, S. (2003). Cooperative multiagent learning. In Alonso, E., Kudenko, D., and Kazakov, D., editors, *Adaptive Agents and Multi-Agent Systems*, pages 1–17. Springer.

Powers, R. and Shoham, Y. (2004). New criteria and a new algorithm for learning in multi-agent systems. In *Advances in Neural Information Processing Systems 18 (Neural Information Processing Systems 2004, NIPS-04)*, pages 1089–1096, Vancouver, Canada.

Prasad, M. V. N., Lesser, V. R., and Lander, S. E. (1998). Learning organizational oles for negotiated search in a multiagent system. *International Journal of Human-Computer Studies*, 48(1):51–67.

Price, B. and Boutilier, C. (2003). Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, 19:569–629.

Pynadath, D. V. and Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423.

Ricci, A., Viroli, M., and Omicini, A. (2005). Environment-based coordination through coordination artifacts. In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *Environments for MultiAgent Systems*, pages 190–214. Springer-Verlag. Revised, Selected and Invited Papers from the 1st Workshop on Environments for MultiAgent Systems (E4MAS-04), New York, USA.

Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 2nd edition.

Schaerf, A., Shoham, Y., and Tennenholtz, M. (1995). Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research*, 2:475–500.

Sen, S., Sekaran, M., and Hale, J. (1994). Learning to coordinate without sharing information. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 426–431, Seattle, Washington, USA.

Sen, S. and Weiss, G. (1999). *Learning in Multiagent Systems*, chapter 6, pages 259–298. The MIT Press, Cambridge, MA, USA.

Shoham, Y., Powers, R., and Grenager, T. (2003). Multi-agent reinforcement learning: A critical survey. Technical report, Computer Science Department, Stanford University, California, USA.

Singh, M. P. and Huhns, M. N. (1996). Challenges for machine learning in cooperative information systems. In Weiß, G., editor, *Distributed Artificial Intelligence Meets Machine Learning, Learning in Multi-Agent Environments*, pages 11–24. Springer. Selected papers from the ECAI'96 Workshop LDAIS, Budapest, Hungary, and the ICMAS'96 Workshop LIOME, Kyoto, Japan.

Singh, S., Jaakkola, T., Littman, M. L., and Szepesvári, C. (2000a). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308.

Singh, S., Kearns, M., and Mansour, Y. (2000b). Nash convergence of gradient dynamics in general-sum games. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 541–548, San Francisco, California, US.

Spaan, M. T. J., Vlassis, N., and Groen, F. C. A. (2002). High level coordination of agents based on multiagent Markov decision processes with roles. In *Workshop on Cooperative Robotics, The 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-02)*, pages 66–73, Lausanne, Switzerland.

Stone, P. and Veloso, M. (1998). A layered approach to learning client behaviors in the RoboCup soccer server. *Applied Artificial Intelligence*, 12:165–188.

Stone, P. and Veloso, M. (1999). Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273.

Stone, P. and Veloso, M. (2000). Multiagent systems: A survey from the machine learning perspective. *Autonomous Robots*, 8(3):345–383.

Suematsu, N. and Hayashi, A. (2002). A multiagent reinforcement learning algorithm using extended optimal response. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-02)*, pages 370–377, Bologna, Italy.

Sutton, R. S. (1991). Integrated modeling and control based on reinforcement learning and dynamic programming. In *Advances in Neural Information Processing Systems 3 (The 1991 Neural Information Processing Systems Conference, NIPS-91)*, pages 471–478, Denver, Colorado, USA.

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

't Hoen, P. J. and de Jong, E. D. (2004). Evolutionary multi-agent systems. In *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*, pages 872–881, Birmingham, United Kingdom.

Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning (ICML-93)*, pages 330–337, Amherst, Massachusets, USA.

Thrun, S. (1992). The role of exploration in learning control. In White, D. and Sofge, D., editors, *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Van Nostrand Reinhold.

Touzet, C. F. (2000). Robot awareness in cooperative mobile robot learning. *Autonomous Robots*, 8(1):87–97.

Vlassis, N. (2003). A concise introduction to multiagent systems and distributed AI. Technical report, University of Amsterdam, The Netherlands.

Walker, A. and Wooldridge, M. (1995). Understanding the emergence of conventions in multi-agent systems. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 384–390, San Francisco, California, USA.

Watkins, C. J. C. H. and Dayan, P. (1992). Technical note: Q-learning. *Machine Learning*, 8:279–292.

Whitehead, S. D. and Lin, L.-J. (1995). Reinforcement learning of non-Markov decision processes. *Artificial Intelligence*, 73(1–2):271–306.