Chapter 9

DEC-MDP/POMDP

9.1. Introduction

MDPs and POMDPs are both mathematical models that have been successfully used to formalize sequential decision-theoretic problems under uncertainty. They allow an agent to decide how to act in a stochastic environment in order to maximize a given performance measure. The agent's decision is based on a complete (MDPs) or partial (POMDPs) observability of the environment. It has been proved that these models are efficient tools for solving problems of mono-agent control and they have been successfully applied to many domains such as mobile robots [BER 01], spoken dialog managers [ROY 00] or inventory management [PUT 94]. This encourages us to consider the extension of these models to cooperative multiagent systems.

EXAMPLE 9.1. Let us go back to the car maintenance example introduced before (see Chapter 1, section 1.1). We now consider several garage employees, who do not always coordinate, who are responsible for repairing and maintaining cars. We can imagine a futuristic scenario where the garage is equipped with a set of robots, each one dedicated to a specific task: one for brakes, one for tires, another one for oil, etc. Thus, several agents may decide to act simultaneously on the same car (or on the same part of the car) while possible having different and partial knowledge about the underlying state of the car. One key question then is: how can the agents cooperate in order to optimize the outcome of their joint action (which results from the simultaneous execution of the agents' individual actions)? The current chapter presents extensions of the MDP formalism that allow for modeling such a decision problem.

Chapter written by Aurélie BEYNIER, François CHARPILLET, Daniel SZER and Abdel-Illah MOUADDIB.

The models that are developed in this chapter rely on different types of hypotheses that can be classified within: i) each agent has a complete knowledge of the system state; ii) each agent has a partial knowledge of the system state; iii) the agents can communicate; iv) the agents cannot (or can partially) communicate; etc.

These hypotheses have led to several formalisms. Among them, we review the most well-known ones: MMDP, Dec-MDP, Dec-POMDP, MTDP, Dec-MDP-Com, COM-MTDP, ND-POMDP, TI-Dec-MDP, OC-Dec-MDP. This chapter also deals with the complexity of computing optimal solutions for the multiagent decision problems described with these formalisms. Indeed, decentralized decision problems suffer from a high complexity which makes the computation of an optimal solution very difficult. We show that the structure of the problem domain, such as the locality of interactions, can sometimes be exploited to overcome this complexity barrier. We also present approximate algorithms that have been developed to deal with more complex problems. It has been proved that some of them converge to local optima and reach special kinds of equilibrium solutions.

9.2. Preliminaries

The observability of a system describes the information available to an agent (or a group of agents) for its decision-making. As explained in Chapter 7, a system is said to be *partially observable* if an agent or a group of agents does not have access to all the information required to make perfect decisions. The problem is then how agents can act optimally in a cooperative manner despite the lack of information. On the other hand, if the agents have full access to all information relevant to optimal decision-making, the system is said to be *fully observable*.

It has been shown that the degree of observability has a big impact on the complexity of decision-making. Therefore, solving a POMDP is much more difficult than solving an MDP. This remains true in the case of multiple decision-makers. In this section, we define several notions related to the observability of a multiagent system that will later be used in this chapter.

In multiagent systems we distinguish between *joint observability* and *local observability* (or individual observability) of the system state (environment + agents). By joint observability we mean the combination of the individual observations of all agents. If joint observability allows the agents to deduce the state of the system, this state is said to be *jointly fully observable*. Otherwise, some features of the system state are not observed by the agents and the system state is *jointly partially observable*.

The system state is said to be *locally (fully) observable* if each agent alone observes all the features of the system state. In such a case, the system state is also jointly fully observable. Similarly, if at least one agent fully observes the system state, this

state is also jointly fully observable. When dealing with blind agents that have no observability of the system, the system is said to be *non-observable* (locally or jointly). In mono-agent settings, local observability of the system state distinguishes MDPs from POMDPs (see Chapter 7). In multiagent settings, more classes of observability exist. Figure 9.1 presents the relationships between the different kinds of observability about the system state in multiagent systems.

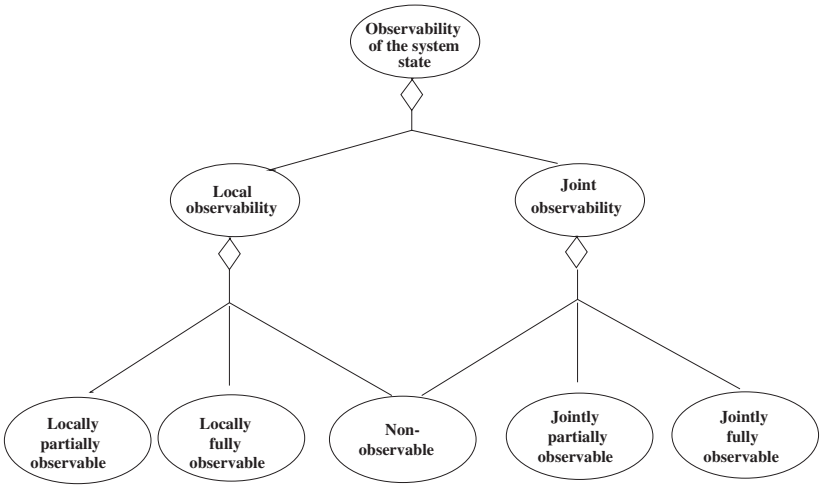

Figure 9.1. *Observability of the system state*

9.3. Multiagent Markov decision processes

In order to extend MDPs to multiagent settings, Boutilier [BOU 96, BOU 99a, BOU 99b] has introduced Multiagent Markov Decision Processes (MMDPs). MMDPs allow for representing sequential decision-making problems in cooperative multiagent settings. This formalism is very much related to models dealing with stochastic games [SHA 53], as presented in Chapter 8. However, MMDPs (and the approaches that are presented in the present chapter) only model cooperative systems. The reward function is thus a common function the agents must maximize, whereas most stochastic games consider separate reward functions and thus different, possibly conflicting, goals for each agent.

Since MMDPs derive from standard Markov decision processes, an MMDP is defined by a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a transition function $\mathcal{T}$ and a reward function $\mathcal{R}$. In order to model the multiagent decision problem, each element a of $\mathcal{A}$ is a "joint action" which is defined as a set of individual actions, one for each agent. Since several agents are considered, a component n, which denotes the number of agents, is added to the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.

DEFINITION 9.1 (multiagent Markov decision processes). *An* MMDP *is defined as a tuple* $\langle n, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ *where:*

– n *is the number of agents* $\mathcal{A}g_i$ *of the system,* $i \in \{1, \ldots, n\}$*;*

– $\mathcal{S}$ *is the set of states* s *of the system;*

– $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ *denotes the set of joint actions, where* $\mathcal{A}_i$ *is the set of individual actions of agent* $\mathcal{A}g_i$*;*

– $\mathcal{T}$ *is the transition function; it gives the probability* $\mathcal{T}(s, a, s')$ *that the system moves to state* s' *when the agents execute the joint action* $a \in \mathcal{A}$ *from state* s*;*

– $\mathcal{R}$ *is the reward function;* $\mathcal{R}(s, a, s')$ *is the reward obtained by the system when the system state changes from* s *to* s' *under the influence of joint action* a*.*

An MMDP can then be viewed as a large MDP. The set of agents is effectively represented as a single agent whose aim is to optimize a policy for an MDP where each action is in fact a joint action (see Chapter 1).

Solving an MMDP consists of finding a joint policy $\pi = \langle \pi_1, \ldots, \pi_n \rangle$, where π_i is the individual policy of agent $\mathcal{A}g_i$. This is a function $\pi_i : \mathcal{S} \rightarrow \mathcal{A}_i$ which maps each state of the system to an action a_i of the agent $\mathcal{A}g_i$. Since an MMDP can be viewed as a large MDP, techniques for optimally solving MDPs, like policy iteration or value iteration, can be used to solve MMDPs (see Chapter 1).

Note that the MMDP model does not consider individual observations. In order to execute its policy, each agent must therefore know the state of system. In many multiagent systems, this property does not hold. Two approaches can then be considered: i) a central entity knows the system state and makes decisions for all the agents; ii) the agents communicate their observations. The last approach can be used if and only if the system state is jointly fully observable and communication is free and instantaneous. Thus, after each action, each agent communicates to the other agents its own observation. After communication, each agent can therefore deduce the system state and is able to decide which action it must execute. Unfortunately, these assumptions seldom hold in multiagent systems. Most of the time, communication is not free and the agents cannot jointly observe the system state.

9.4. Decentralized control and local observability

Decentralized Partially Observable Markov Decision Processes (DEC-POMDPs) and Decentralized Markov Decision Processes (DEC-MDPs) extend POMDPs and MDPs to multiagent decentralized control. These models allow agents to act in a fully decentralized manner while maximizing the performance of the system as a whole. Unlike MMDPs, DEC-POMDPs and DEC-MDPs do not assume complete individual observability of the system state.

In the following section we introduce the DEC-MDP and DEC-POMDP models. Algorithms for solving problems formalized as DEC-POMDPs or DEC-MDPs are described in section 9.6.

9.4.1. *Decentralized Markov decision processes*

In a general decentralized control problem, agents may not always be able to observe the true state of the environment, nor the current configuration of the other agents. Often, agents must therefore base their decisions on a partial view of the system. This is why DEC-MDPs and DEC-POMDPs define a set of observations Ω, which consists of the individual observation sets Ω_i of each agent $\mathcal{A}g_i$. The observation function $\mathcal{O}$ then gives the probability for agent $\mathcal{A}g_i$ to observe o_i when the system is in state s, the agents execute joint action a and the system moves to the state s'.

DEFINITION 9.2 (decentralized partially observable Markov decision processes). *A* DEC-POMDP *for n agents is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \Omega, \mathcal{O}, \mathcal{R} \rangle$ where:*

– $\mathcal{S}$ is a finite set of system states;

– $\mathcal{A} = \langle \mathcal{A}_1, \ldots, \mathcal{A}_n \rangle$ is a set of joint actions; $\mathcal{A}_i$ is the set of actions a_i that can be executed by agent $\mathcal{A}g_i$;

– $\mathcal{P} = \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ is a transition function; $\mathcal{P}(s, a, s')$ is the probability of the outcome state s' when the agents execute the joint action a from s;

– $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n$ is a finite set of observations, where Ω_i is $\mathcal{A}g_i$'s set of observations;

– $\mathcal{O} = \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \Omega \rightarrow [0,1]$ is the observation function; $\mathcal{O}(s, a, s', o = \langle o_1, \ldots, o_n \rangle)$ is the probability that each agent $\mathcal{A}g_i$ observes o_i when the agents execute the joint action a from state s and the system moves to state s';[1]

– $\mathcal{R}$ is a reward function; $\mathcal{R}(s, a, s')$ is the reward the system obtains when the agents execute joint action a from state s and the system moves to state s'.

Similarly to POMDPs, the definition of a DEC-POMDP problem includes a problem horizon T, which specifies the number of decision steps that are taken into account. As in single-agent problems, finite-horizon or infinite-horizon problems can be considered. Note that a single-agent DEC-POMDP is a POMDP.

If the state of the system is jointly fully observable, the DEC-POMDP is a DEC-MDP.

1. This definition allows for modeling individual observations which are correlated to each other.

DEFINITION 9.3 (decentralized Markov decision processes). *A* DEC-MDP *is a special kind of* DEC-POMDP *where the system state is jointly observable. This property can be formalized as follows:*

$$\textit{If } \mathcal{O}\big(s, a, s', o = \langle o_1, \ldots, o_n \rangle\big) > 0, \textit{ then } \Pr\big(s' \mid \big\langle o_1, \ldots, o_n \big\rangle\big) = 1.$$

Note that this property does not imply that each agent separately can observe the full system state. Thus, a DEC-MDP has the same components as a DEC-POMDP.

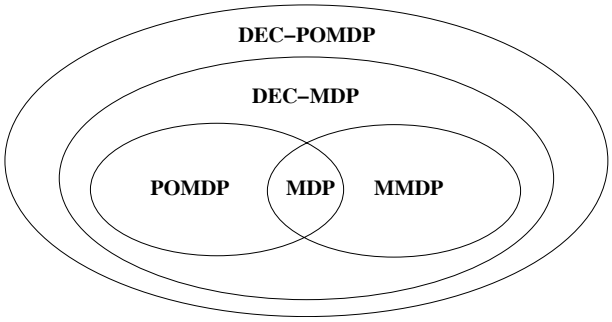

Figure 9.2. *Relationships between the different types of Markovian processes*

For each possible transition, the reward function $\mathcal{R}$ of a DEC-POMDP (or a DEC-MDP) gives the reward which is obtained by the system. The goal is to maximize the accumulated rewards. Recent works have focused on developing offline planning algorithms to solve problems formalized as DEC-POMDPs and DEC-MDPs. They consist of computing a set of individual policies, one per agent, defining the agents' behaviors. Each individual policy maps the agent's information (its state, its observations or its belief state) to an action.

DEFINITION 9.4 (individual policy for a DEC-POMDP (or a DEC-MDP)). *An individual policy π_i for an agent $\mathcal{A}g_i$ in a* DEC-POMDP *(or a* DEC-MDP*) is a mapping from the agents' information to an action $a_i \in \mathcal{A}_i$.*

Usually, an individual policy for a DEC-POMDP (or a DEC-MDP) is defined under the "perfect recall assumption" which assumes that each agent always has access to all its received information [SEU 08]. An individual policy π_i for an agent $\mathcal{A}g_i$ in a DEC-POMDP (or a DEC-MDP) is then defined as a mapping from the agent's local history of observations $\bar{o_i} = o_i^1, \ldots, o_i^t$ to an action $a_i \in \mathcal{A}_i$. Since there provably exists an optimal deterministic policy to every DEC-POMDP, the agents' local history of observations is a sufficient information to make an optimal decision. When the optimal policy is probabilistic, each agent's has to consider the history of its observations and the history of its actions.

However, an individual policy π_i for an agent $\mathcal{A}g_i$ in a DEC-POMDP (or a DEC-MDP) can also be defined as a mapping from the agents' belief states to an action $a_i \in \mathcal{A}_i$, like it has been done in the POMDP setting (See Chapter 7, section 7.1.4).

DEFINITION 9.5 (joint policy for a DEC-POMDP (or a DEC-MDP)). *A joint policy π in an n-agent* DEC-POMDP *(or a* DEC-MDP*) is a set of individual policies $\langle \pi_1, \ldots, \pi_n \rangle$, where π_i is the individual policy of the agent $\mathcal{A}g_i$.*

Optimally solving a DEC-POMDP consists of finding a joint policy which maximizes the expected total reward of the system. When considering a finite-horizon DEC-POMDP with initial state s_0, this consists of maximizing the value $V^{\pi}(s_0)$ defined over the finite horizon T such as

$$V^{\pi}\big(s_0\big) = E\left[\sum_{t=0}^{T-1} R\big(s_t, a_t, s_{t+1}\big) \mid s_0, \pi\right].$$

When considering infinite-horizon DEC-POMDPs, a discount factor $\gamma \in [0, 1)$ is used to weight the expected reward:

$$V^{\pi}\big(s_0\big) = E\left[\sum_{t=0}^{T-1} \gamma^t R\big(s_t, a_t, s_{t+1}\big) \mid s_0, \pi\right].$$

9.4.2. *Multiagent team decision problems*

Multiagent Team Decision Problems (MTDP), introduced by Pynadath and Tambe [PYN 02], are very similar to DEC-POMDPs. Like DEC-POMDPs, MTDPs allow for formalizing problems of decentralized control in stochastic and cooperative domains. An MTDP has almost the same components as a DEC-POMDP. In order to easily formalize the beliefs each agent has about the system state at step t, a set of belief states is added to the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R} \rangle$. Thus, MTDPs distinguish between observations and beliefs about the state of the system.

DEFINITION 9.6 (multiagent team decision problem). *An* MTDP *is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, B, \Omega, \mathcal{O}, \mathcal{R} \rangle$ where:*

– $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R} \rangle$ is a DEC-POMDP*;*

– $B = \langle B_1, \ldots, B_n \rangle$ is the set of belief states of the agents where B_i is the set of belief states b_i^t of $\mathcal{A}g_i$; each agent $\mathcal{A}g_i$ builds its own belief state b_i^t at time t from the observations it has made until t.

Solving an MTDP consists of finding an optimal joint policy $\pi = \langle \pi_1, \ldots, \pi_n \rangle$, where each individual policy π_i is a mapping from a belief state b_i^t to an action a_i.

Unlike Seuken and Zilberstein's definition of an individual policy for a DEC-POMDP [SEU 08], the MTDP definition of an individual policy does not assume perfect recall.

DEFINITION 9.7 (individual policy for an MTDP). *An individual policy π_i for an agent $\mathcal{A}g_i$ in an* MTDP *is a mapping from the agents' belief states $b_i^t \in B_i$ to an action $a_i \in \mathcal{A}_i$.*

DEFINITION 9.8 (joint policy for an MTDP). *A joint policy π in an n-agent* MTDP *is a set of individual policies $\langle \pi_1, \ldots, \pi_n \rangle$, where π_i is the individual policy of the agent $\mathcal{A}g_i$.*

Seuken and Zilberstein [SEU 08] have proven that DEC-POMDPs and MTDPs are equivalent if each agent has access to all of its received information (each agent recalls all the observations and the messages it has previously received). Indeed, we have shown in Chapter 7, section 7.1.3, that the belief state can be calculated from the entire history of observations and actions. This assertion remains true in the multiagent case, and the state of information captured by a DEC-POMDP and an MTDP is thus equivalent.

Seuken and Zilberstein have also demonstrated that the policies are equivalent if each agent has access to all the information it has accumulated: a policy with a value k for a given MTDP has the same value k for the corresponding DEC-POMDP. This DEC-POMDP is obtained by extracting the history of observations from each belief state. Thus, an optimal policy for an MTDP is also an optimal policy for the corresponding DEC-POMDP, and an optimal policy for a DEC-POMDP is also an optimal policy for the corresponding MTDP.

Table 9.1 describes the relationships between MTDPs, DEC-POMDPs, DEC-MDPs, POMDPs and MDPs.

<table>
<tr><th>Control</th><th colspan="2">Centralized</th><th colspan="2">Decentralized</th></tr>
<tr><td>Joint observability of the system state</td><td>Yes</td><td>No</td><td>Yes</td><td>No</td></tr>
<tr><td rowspan="2">Formalism</td><td>MDP
MMDP</td><td rowspan="2">POMDP</td><td>DEC-MDP</td><td>DEC-POMDP</td></tr>
<tr><td colspan="2">MTDP</td></tr>
</table>

Table 9.1. *Relationships between Markovian models and joint observability*

9.4.3. *Complexity*

Solving a DEC-POMDP or a DEC-MDP optimally is a very hard problem because the agents do not individually observe the state of the system – and because their respective beliefs about this state are not synchronized. While the performance of the system relies on the underlying state s and the joint action a that is been executed, the synchronized knowledge about this state and the potential actions of all agents is in general not obtainable in a decentralized setting.

Although optimally solving a POMDP is PSPACE-complete for the finite-horizon case (see Chapter 7, section 7.3) [PAP 87], optimally solving a DEC-POMDP is much more difficult. Bernstein *et al.* [BER 02] proved that solving a finite-horizon DEC-POMDP with 2 agents or more is NEXP-complete. The same complexity holds for DEC-MDPs.

These complexity results are closely related to the observability each agent has about the state of the system. If each agent individually fully observes the state of the system, the multiagent decision problem can be formalized as an MDP whose complexity is P-complete. If the agents have no observations about the system (blind agents), the multiagent decision problem can be formalized as a non-observable MDP (NOMDP) which is known to be NP-complete. Table 9.2 sums up how the observability influences the complexity of the problem.

Observability of the system state	Locally observable	Jointly fully observable	Jointly partially observable	Non-observable
Complexity	P-complete	NEXP-complete	NEXP-complete	NP-complete
Formalism	MMDP, DEC-MDP	DEC-MDP	DEC-POMDP	
	MTDP			

Table 9.2. *Observability and time complexity*

9.5. Sub-classes of DEC-POMDPs

It has been shown that optimally solving a DEC-POMDP is a hard problem. This is why several attempts have been made to identify problem properties that can help reducing the complexity of computing an optimal policy. We present several of such subclasses in the following sections.

9.5.1. *Transition and observation independence*

In some problem settings, the actions of each agent have only limited effects on the other agents. This observation led Goldman and Zilberstein [GOL 04] to introduce the

notions of transition independence and observation independence. In order to define transition independence and observation independence, we first introduce factored DEC-MDPs (which relate to the Factored MDPs presented in Chapter 4).

DEFINITION 9.9 (factored DEC-MDPs). *An n-agent factored* DEC-MDP *is a* DEC-MDP *where the state of the system $\mathcal{S}$ can be factored into $n+1$ components $\mathcal{S} = \mathcal{S}_0 \times \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$. $\mathcal{S}_0$ is the features of the system state that may be observed by the agents and that are not modified by their actions of the agents themselves. $\mathcal{S}_i$ ($i \in [1, n]$) denotes those features of the system state that are observed and affected only by agent $\mathcal{A}g_i$.*

In a factored DEC-MDP, for some agent $\mathcal{A}g_i$, we refer to $s_i \in \mathcal{S}_i$ as its *sub-state* and $\hat{s}_i \in \mathcal{S}_i \times \mathcal{S}_0$ as its *local state*. When the agent fully observes its local state, we obtain $o_i = \hat{s}_i \; \forall i$.

DEFINITION 9.10 (transition independence). *A factored* DEC-MDP *with independent transitions is such that the probability an agent $\mathcal{A}g_i$ moves from a sub-state s_i to a sub-state s_i' only depends on the action a_i the agent has executed. Formally, a factored* DEC-MDP *is transition independent if the set of states $\mathcal{S}$ can be decomposed into $n+1$ components $\mathcal{S}_0, \ldots, \mathcal{S}_n$ and the transition function can be decomposed as a product of probabilities such as $\mathcal{P} = \prod_{i=0}^{n} \mathcal{P}_i$, where*

$$\mathcal{P}_i\big(s_i' \mid (s_0 \ldots s_n), (a_1 \ldots a_n), (s_0' \ldots s_{i-1}', s_{i+1}' \ldots s_n)\big) = \begin{cases} \mathcal{P}_0\big(s_0' \mid s_0\big) & \text{if } i = 0, \\ \mathcal{P}_i\big(s_i' \mid \hat{s}_i, a_i, s_0'\big) & \text{if } 1 \leq i \leq n. \end{cases}$$

Given a two-agent factored DEC-MDP, transition independence can be formalized as follows:

$$\begin{aligned} &\forall s_0, s_0' \in \mathcal{S}_0,\ \forall s_1, s_1' \in \mathcal{S}_1,\ \forall s_2, s_2' \in \mathcal{S}_2,\ \forall a_1 \in A_1,\ \forall a_2 \in A_2, \\ &\Pr\big(s_0' \mid (s_0, s_1, s_2), a_1, a_2, (s_1', s_2')\big) = \Pr\big(s_0' \mid s_0\big) = \mathcal{P}_0, \\ &\Pr\big(s_1' \mid (s_0, s_1, s_2), a_1, a_2, (s_0', s_2')\big) = \Pr\big(s_1' \mid \hat{s}_1, a_1, s_0'\big) = \mathcal{P}_1, \\ &\Pr\big(s_2' \mid (s_0, s_1, s_2), a_2, a_1, (s_0', s_1')\big) = \Pr\big(s_2' \mid \hat{s}_2, a_2, s_0'\big) = \mathcal{P}_2. \end{aligned}$$

DEFINITION 9.11 (observation independence). *A factored* DEC-MDP *with independent observation is such that the set of states $\mathcal{S}$ can be decomposed into $n+1$ components $\mathcal{S}_0, \ldots, \mathcal{S}_n$ and there exists, for each agent $\mathcal{A}g_i$, an observation function $\mathcal{O}_i$ such that*

$$\begin{aligned} &\mathcal{O}_i\big(\hat{s}_i, a_i, \hat{s}_i', o_i\big) = \Pr\big(o_i \mid \hat{s}_i, a_i, \hat{s}_i'\big) \\ &= \Pr\big(o_i \mid \langle s_0, \ldots, s_n\rangle, \langle a_1, \ldots, a_n\rangle, \langle s_0', \ldots, s_n'\rangle, \langle o_1, \ldots, o_{i-1}, o_{i+1}, \ldots, o_n\rangle\big) \end{aligned}$$

and

$$\mathcal{O}\big(\langle s_0,\ldots,s_n\rangle,\langle a_1,\ldots,a_n\rangle,\langle s'_0,\ldots,s'_n\rangle,\langle o_1,\ldots,o_n\rangle\big)=\prod_{i=1}^{n}\mathcal{O}_i\big(\hat{s}_i,a_i,\hat{s}'_i,o_i\big).$$

Properties such as transition or observation independence allow for identifying classes of problems that are easier to solve than general DEC-MDPs. Goldman and Zilberstein [GOL 04] have proven that a DEC-MDP with independent transitions and observations is NP-complete. In fact, a local policy for an agent $\mathcal{A}g_i$ is a mapping from its last observation o_i to an action a_i. Its policy is thus of size polynomial in $|\mathcal{S}_i|T$, where $|\mathcal{S}_i|$ is the number of states of agent $\mathcal{A}g_i$. Computing the value of such a policy takes polynomial time. There are $|A_i|^{|\mathcal{S}_i|T}$ possible policies to evaluate, so computing an optimal policy for a DEC-MDP with independent transitions and observations is NP-complete.

Based on this observation, an optimal algorithm, the Coverage Set Algorithm (CSA), has been developed to solve DEC-MDPs with independent observations and transitions [BEC 03] (see section 9.6.1.3).

9.5.2. *Goal oriented DEC-POMDPs*

Goal oriented DEC-MDPs [GOL 04] formalize problems where the agents try to reach particularly identified goal states. For instance, in the car-maintenance example, the garage-agents may try to reach a state where there is no more cars to repair.

DEFINITION 9.12. *Goal oriented* DEC-POMDP*s A* DEC-POMDP *is goal-oriented (*GO-DEC-POMDP*) if the following conditions hold:*

1) there exists a special subset $\mathcal{G}$ of $\mathcal{S}$ of global goal states. At least one of the global goal states $g\in\mathcal{G}$ is reachable by some joint policy;

2) the process ends at time T (the finite horizon of the problem);

3) all actions a_i of agent $\mathcal{A}g_i$ incurs a cost $C(a_i)<0$;

4) the global reward is $R(s,\langle a_1,\ldots,a_n\rangle,s')=\sum_{i=1}^{n}C(a_i)$;

5) if at time T the system is in a state $s\in\mathcal{G}$, there is an additional reward that is awarded to the system for reaching a global goal state.

It has been proved that optimally solving a GO-DEC-MDP is NEXP-complete and thus not easier than solving DEC-MDPs in general. However, particular classes of GO-DEC-MDPs indeed have a lower complexity than NEXP.

DEFINITION 9.13. *A* GO-DEC-POMDP *has uniform cost when the cost of all actions is the same.*

Goldman and Zilberstein [GOL 04] have proven that optimally solving a GO-DEC-MDP with independent transitions and observations, with a single goal state and uniform cost, is P-complete. In this case, each agent separately calculates a policy that minimizes the cost to the global goal state. Since costs are uniform, computing an optimal policy therefore consists of finding the shortest path to the goal state (this can be done using dynamic programming).

In this section, we have described properties of the multiagent decision problems that allow for identifying sub-classes of DEC-POMDPs and DEC-MDPs with a lower complexity. These properties can then be exploited to develop more efficient algorithms. Such algorithms are presented in section 9.6.2.7. Figure 9.3 sums up the influence of transition independence, observation independence and goal-oriented properties on the complexity.

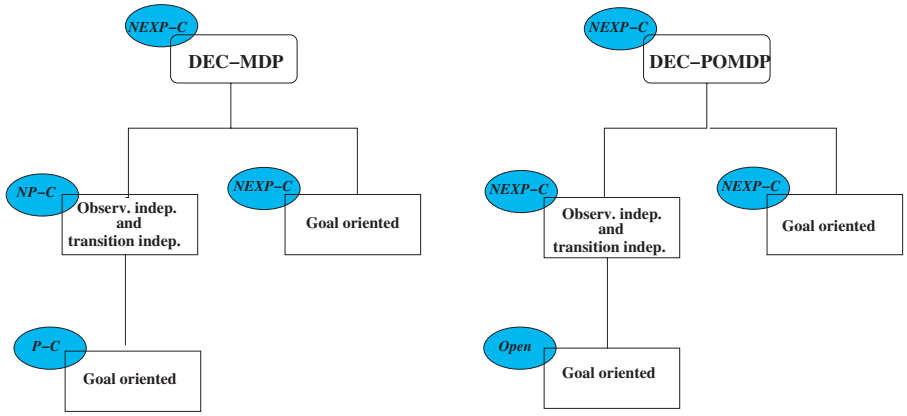

Figure 9.3. *Complexity analysis of classes of* DEC-MDP*s and* DEC-POMDP*s*

9.5.3. *DEC-MDPs with constraints*

DEC-POMDPs and DEC-MDPs represent powerful models to formalize general multiagent decision problems in stochastic environments, but they are sometimes too general and it may be difficult to formalize some particular structure within this general framework. One such example are constraints between actions. Time or resource dependencies between the actions often arise in practice, but they cannot directly be integrated into the standard DEC-POMDP formalism. Identifying such dependencies, however, can reduce the problem complexity and increase the applicability of DEC-POMDPs to real-world problems. In this section we introduce some classes of DEC-MDPs that allow for explicitly formalizing time and resource constraints between the actions of the agents.

9.5.3.1. *Event-driven DEC-MDPs*

Event-Driven Decentralized Markov Decision Processes (ED-DEC-MDP) have been defined by Becker *et al.* [BEC 04a]. They introduce time dependencies between the agents such as "enable" or "facilitate" dependencies. It is thus possible to express the fact that an agent $\mathcal{A}g_i$ cannot start the execution of an action a_i before another agent $\mathcal{A}g_j$ has finished the execution of an action a_j. Quality dependencies such as "a_i must be executed successfully before a_j starts to produce a higher quality for a_j" can also be represented.

These dependencies are first expressed using TAEMS [DEC 93] which is a hierarchical task modeling language. The transition function of the ED-DEC-MDP is then defined in order to formalize these constraints.

ED-DEC-MDPs assume that the system state can be factored, and that each agent fully observes its local state. Moreover, ED-DEC-MDPs are reward independent. Thus, the reward function $\mathcal{R}$ can be decomposed as a sum of local reward function such that

$$\mathcal{R}\big(\langle s_1, \ldots, s_n\rangle, \langle a_1, \ldots, a_n\rangle, \langle s'_1, \ldots, s'_n\rangle\big) = \sum_{i=1}^{n} R_i\big(s_i, a_i, s'_i\big).$$

Optimally solving an ED-DEC-MDP is NP-complete [BEC 04a]. In fact, an ED-DEC-MDP has a complexity exponential in the number of states $|\mathcal{S}|$ and doubly exponential in the number of dependencies.

9.5.3.2. *Opportunity-cost DEC-MDPs*

In order to extend the applicability of DEC-POMDP and DEC-MDP models to real-world applications, Beynier and Mouaddib [BEY 05, BEY 06] have introduced the OC-DEC-MDP model. OC-DEC-MDPs are able to represent multiagent decentralized decision problems with time and precedence constraints. They can deal with more complex time and action representations than standard DEC-POMDPs and DEC-MDPs since they allow for different possible durations for each task, and they take into account constraints on task execution. Unlike DEC-POMDPs and DEC-MDPs, Beynier and Mouaddib's approach do not make the assumption of synchronization between the agents: at each time step, only a subset of the agents have to make a decision while the rest of the agents keep in executing their current tasks.

The following section defines the concept of "mission" and "task" that are used in the OC-DEC-MDP model.

9.5.3.3. *Problem statement*

Based on the study of real-world multiagent decision problems such as decision making in colonies of robots, Beynier *et al.* have defined a mission $\mathcal{X}$ as a set of agents that must complete a set of tasks.

DEFINITION 9.14 (mission). *A **mission** $\mathcal{X}$ is defined as a pair $\langle \mathcal{A}g, \mathcal{T} \rangle$ where:*

– $\mathcal{A}g = \{\mathcal{A}g_1, \ldots, \mathcal{A}g_n\}$ is a set of n agents;

– $\mathcal{T} = \{t_1, \ldots, t_p\}$ is a set of tasks the agents have to execute.

Time and resources are considered as discrete. Thus, each task is assigned a probabilistic set of durations, a probabilistic set of resource consumptions, a set of predecessors and a time window. Each task of the problem must be executed respecting time, precedence and resource constraints.

DEFINITION 9.15. *Each **task** $t_i \in \mathcal{T}$ is defined by:*

*– an **agent** that must execute the task;*

*– **different possible durations** associated with **probabilities**; $P_i^t(\delta_i)$ is the probability for the execution of task t_i to last δ_i time units;*

*– **different possible resource consumptions** associated with **probabilities**. $P_i^r(\Delta_r^i)$ is the probability for the execution of task t_i to consume Δ_r^i resources; each agent has an initial amount of resources available for executing its tasks, and this amount of resources must be sufficient to successfully execute the task;*

*– a **time window** $TC_i = [EST_i, LET_i]$ during which the task t_i must be executed; EST_i stands for the* Earliest Start Time *of the task and LET_i is its* Latest End Time*; the time window describes time constraints on the execution of the task;*

*– a **set of predecessor tasks** $Pred_i$: the tasks that must be finished before t_i can start*

$$\forall t_i \in \mathcal{T}, \quad t_i \notin root \Longleftrightarrow \exists t_j \in \mathcal{T} : t_j \in \mathrm{Pred}\left(t_i\right)$$

where $root$ are the first tasks to be executed (tasks without predecessors);

*– a **reward** $\mathcal{R}_i$ obtained by the agent when the task t_i has been executed respecting time, precedence and resource constraints.*

Each time an agent finishes executing a task with respect to constraints, it obtains a reward which depends on the executed task. The essence of the problem is to find a joint policy that maximizes the cumulative reward of the agents. Since the execution time and the resource consumption of the tasks are uncertain, the optimal policy is the one that maximizes the expected reward of the system.

9.5.3.4. *The OC-DEC-MDP model*

In order to plan the execution of a mission $\mathcal{X}$ and to solve large problems, Beynier and Mouaddib suggest splitting the initial multiagent decision problem into a set of MDPs that are easier to solve. Each MDP stands for a single-agent decision problem.

Thus, individual states and actions are considered and the exponential growth of the joint action set and the state set is avoided. However, precedence constraints between the tasks lead to some dependencies between the agents, and the local MDPs are not independent. Beynier and Mouaddib thus introduce the OC-DEC-MDP framework, which performs such a decomposition and proposes a richer model of time and action formalizing time, precedence and resource constraints.

DEFINITION 9.16. *An n-agent **OC-DEC-MDP** is a set of n MDPs, one for each agent. The MDP associated with an agent $\mathcal{A}g_i$ is defined as a tuple $\langle \mathcal{S}_i, \mathcal{A}_i, \mathcal{P}_i, \mathcal{R}_i \rangle$ where:*

– *$\mathcal{S}_i$ is the finite set of states of the agent $\mathcal{A}g_i$,*

– *$\mathcal{A}_i$ is the finite set of actions of the agent $\mathcal{A}g_i$,*

– *$\mathcal{P}_i$ is the transition function of the agent $\mathcal{A}g_i$,*

– *$\mathcal{R}_i$ is the reward function of the agent $\mathcal{A}g_i$.*

In order to define the components of the local MDPs, constraints propagation algorithms that use the formal description of the mission are employed.

Each component of the local MDPs is defined to formalize the constraints on the task execution. There exists three kinds of states:

– *success states*: the states resulting from the successful execution of a task (respecting time, precedence and resource constraints);

– *partial failure states*: the states resulting from not respecting precedence constraints (it starts the execution of a task too early, i.e. before the predecessors finish their execution); when an agent starts to execute a task, it immediately observes whether the predecessors have finished their executions or not (this information is assumed to be an observable feature of the environment);[2]

– *permanent failure states*: the states resulting from violating time or resource constraints are violated (the agent finishes to execute the task after the deadline or it lacks resources).

Each agent $\mathcal{A}g_i$ observes its state s_i but does not have access to the other agents' states, nor to the system state.

Since problems formalized by OC-DEC-MDPs are transition dependent, each individual transition function depends on the other agents' transition functions. Beynier and Mouaddib use Bayesian Networks and time propagation algorithms to decompose the joint transition function into individual transition functions.

2. Since the agent could retry to execute the task later and succeed, this failure is not permanent and is called "partial failure".

It has been proved that optimally solving an OC-DEC-MDP requires an exponential amount of computation time. Note that time, resource and precedence constraints limit the numbers of states and actions. However, these constraints do not reduce the worst-case complexity of OC-DEC-MDPs.

9.5.4. *Communication and DEC-POMDPs*

Previous sections have discussed the influence of the (partial) observability of the environment on the complexity of multiagent decision problems. Studies of multiagent systems show that communication is an obvious way to increase each agent's knowledge about the system: if an agent communicates its observations, it increases the other agents' knowledge about the system state.[3] The agents can therefore better coordinate and may achieve a higher performance as a team.

In section 9.4.1, we show that the individual observability of the system state can be obtained if the system state is jointly fully observable and communication between the agents is free and instantaneous. Unfortunately, things are usually not that easy: communicating is costly (it takes time and it consumes resources such as energy or bandwidth) and the system state is not jointly fully observable. Even if an agent is able to communicate, it must therefore trade-off the cost of communicating against the utility of communicated information [BEC 05].

Communication can be expressed directly within the ordinary DEC-POMDP framework; it is sufficient to define actions that represent the transmission of a message, observations that represent the reception of a message, and to adapt the transition and observation functions accordingly. However, it may sometimes be helpful to introduce explicit communication into the DEC-POMDP framework. Common approaches that address this issue can be divided into two categories. The first category [XUA 01, GOL 04, PYN 02] assumes that the agents alternate the execution of a domain action[4] and the execution of a communication action (see Figure 9.4). During each communication period, the agents choose whether to communicate or not. If an agent decides not to communicate, it waits until the next domain action phase where it could execute a domain action. These approaches do not consider that the agents could execute domain actions instead of communicating.

Nair *et al.* [NAI 04] describe another kind of approach that does not assume a separate communication phase. Since spending time and resources on communication

3. Research on communication distinguishes between direct and indirect communication. Direct communication consists of information sharing using direct message exchange. Indirect communication consists of acting upon the environment to modify features that could be observed and interpreted by the other agents as communicated information.

4. A domain action is an action that does not consist of communicating.

influences the future opportunities to execute domain actions, there exists an associated decision problem at each time step in order to determine whether sharing information or executing a domain action is more promising.

9.5.4.1. *DEC-POMDPs with communication*

Goldman and Zilberstein [GOL 03] have extended DEC-POMDPs to include particular communication actions. They assume separate communication and action phases. At each decision step, each agent first decides whether it communicates or not. If an agent chooses to communicate, it sends its message to the other agents. The agent then decides which domain action to perform and it executes this action. As shown in Figure 9.4, each decision step consists of two stages: the communication stage and the domain action stage.

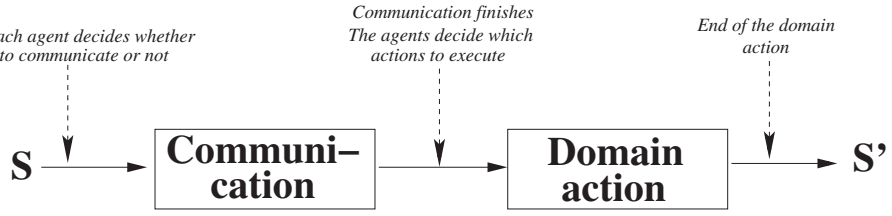

Figure 9.4. *Communication and action stages*

Each agent's policy π_i is then composed of two distinct policies:

– the communication policy π_i^{Σ} which maps histories of observations and histories of messages to a communication action,

– the action policy π_i^a which maps histories of observations and histories of messages to a domain action.

The DEC-POMDP-COM model proposed by Goldman and Zilberstein [GOL 03] adds two components to DEC-POMDPs: an alphabet of messages Σ and a cost function C_Σ associated with the transmission of a message.

DEFINITION 9.17 (decentralized partially observable Markov decision processes with communication). *A* DEC-POMDP-COM *is defined as a tuple* $\langle \mathcal{S}, \mathcal{A}, \Sigma, C_\Sigma, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R} \rangle$ *where:*

– $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R} \rangle$ *is a* DEC-POMDP*;*

– Σ *denotes the alphabet of messages;* $\sigma_i \in \Sigma$ *is a message that can be sent by agent* $\mathcal{A}g_i$*;*

– C_Σ *describes the cost of a message; this is a function:* $C_\Sigma : \Sigma \rightarrow \mathcal{R}$.

Note that an empty message with zero-cost can be considered.

Communication can also be introduced in DEC-MDPs [XUA 01] in a similar way. MTDPs have also been extended to model communication actions [PYN 02]. A COM-MTDP (*Communicative Multiagent Team Decision Problem*) is thus defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \Sigma, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{B}, \mathcal{R} \rangle$, where the reward function $\mathcal{R}$ is extended to take into account the cost of sending a message.

9.5.4.2. *Complexity results*

It has been proved [SEU 08] that DEC-POMDPs and DEC-POMDP-COMs are equivalent. In fact, a DEC-POMDP-COM can be mapped to an equivalent DEC-POMDP: communication actions are added to the set of actions $\mathcal{A}_i$ of each agent $\mathcal{A}g_i$. The state space, the transition function, the observation set, the observation function and the reward function are also adapted to consider explicit communication actions as domain actions. Seuken and Zilberstein have proven that a policy with a value k in the original DEC-POMDP-COM has the same value k in the corresponding DEC-POMDP. Furthermore, a DEC-POMDP can be mapped to an equivalent DEC-POMDP-COM with an empty set of communication messages and no cost function. It can be deduced that DEC-POMDPs and DEC-POMDP-COMs are equivalent.

It has also been demonstrated that COM-MTDPs and DEC-POMDP-COMs are equivalent if each agent has access to all of its received information (perfect recall assumption). When this property holds, all the aforementioned models (DEC-POMDP, MTDP, DEC-POMDP-COM and COM-MTDP) are equivalent [SEU 08]. Since optimally solving a DEC-POMDP with 2 agents or more is NEXP-complete, it follows that MTDPs, DEC-POMDP-COMs and COM-MTDPs are also NEXP-complete.

This complexity result assumes a general model of explicit communication where sending a message has a cost. Under the assumption of free and instantaneous communication, the problem complexity is reduced. If the state of the system is jointly fully observable, the problem can be reduced to an MMDP [BOU 99a] which is known to be P-complete [PAP 87]. If the system state is not jointly fully observable, the problem can be formalized as a large POMDP which is PSPACE-complete [PAP 87]. If the agents have no observability of the system (blind agents), the problem is reduced to an NOMDP which is NP-complete. Table 9.3 sums up how the observability and the communication model influence the problem complexity.

9.5.4.3. *Discussion*

The communication decision problem consists of deciding when to communicate, which information to communicate, and to which agents. Although some approaches, such as the DEC-POMDP-COM model, define an alphabet of messages which allows for considering different message content, most experiments dealing with these approaches assume that the agents always communicate their last observation.

Observability of the system state	Locally fully observable	Jointly fully observable	Jointly partially observable	Non-observable
Free communication	P-complete	P-complete	PSPACE-complete	NP-complete
General communication	P-complete	NEXP-complete	NEXP-complete	NP-complete

Table 9.3. *Observability and complexity results*

Moreover, communicative DEC-POMDPs have mainly been studied from the point of view of synchronizing communications [GOL 04, NAI 04, SHE 06]. This kind of explicit communication assumes that if an agent decides to communicate, then all agents must exchange their information so as to unify their world view. Synchronizing communications fit nicely with problems where initializing communication is very costly compared to the cost of the information transfer. Nonetheless, most of the time, such communication is inappropriate and may lead to undesirable extra cost. Moreover, it assumes that an agent cannot execute a domain action while other agents communicate.

Recently, Spaan *et al.* [SPA 08] have described an approach to deal with more realistic forms of communication. Instead of assuming that communication never fails and is instantaneous, they consider probabilities on the success of communications within Δt units of time. Even if this approach allows for representing different durations of communication actions, it considers synchronizing communications.

9.6. Algorithms for solving DEC-POMDPs

The following sections deal with solving problems formalized as DEC-POMDPs. We will essentially cover three classes of algorithms: optimal algorithms for finite-horizon DEC-POMDPs, approximate algorithms for finite-horizon DEC-POMDPs and approximate algorithms for infinite-horizon DEC-POMDPs. Approaches for solving general DEC-POMDPs will be described as well as approaches for solving specific sub-classes of DEC-POMDPs.

In this chapter, we are interested in decentralized control so we consider approaches where the policies are executed in a decentralized way. However, most of the following algorithms calculate an optimal or an approximate policy in a centralized way: a central entity calculates the individual policies of the agents which are then sent to the agents for execution.

9.6.1. *Optimal algorithms*

A policy for a finite-horizon POMDP can always be represented as a decision tree [KAE 98] denoted by q (see Chapter 7) where nodes are labeled with actions and arcs are labeled with observations. The action associated with the root of the tree is denoted by $\alpha(q)$ and the sub-tree related to the observation o is $q(o)$. $\lambda(q,i)$ is the ith leaf node of tree q.

Let q_i^T be a tree of horizon T. Solving a DEC-POMDP consists of finding a joint policy $\boldsymbol{q}^T = \langle q_1^T, q_2^T, \ldots, q_n^T \rangle$ of horizon T that maximizes the expected amount of reward over the horizon:

$$E\left(\sum_{t=0}^{T-1} \mathcal{R}\big(s_t, \big\langle a_1, a_2, \ldots, a_n \big\rangle_t, s_{t+1}\big) \right).$$

The value of the joint policy $\boldsymbol{q}^T$ for the initial state s can be calculated by dynamic programming using the following equation:

$$V\big(s, \boldsymbol{q}^T\big) = \sum_{s' \in \mathcal{S}} \underbrace{P\big(s' \mid s, \boldsymbol{q}^T\big)}_{\mathcal{P}\left(s, \boldsymbol{q}^T(s), s'\right)} \left[\sum_{\boldsymbol{o} \in \Omega} \underbrace{P\big(\boldsymbol{o} \mid s, \boldsymbol{q}^T, s'\big)}_{\mathcal{O}\left(s, \boldsymbol{q}^T(s), s', \boldsymbol{o}\right)} V_{\boldsymbol{q}^T}(s') \right],$$

where $\boldsymbol{o} = \langle o_1, \ldots, o_n \rangle$ denotes the joint observation and $\boldsymbol{q}^T(\boldsymbol{o})$ is the joint policy for horizon $(T-1)$ after observing $\boldsymbol{o}$.

One way to optimally solve a DEC-POMDP is to perform an exhaustive policy search (see Algorithm 9.1). The number of policies that have to be considered is here

$$\left(|\mathcal{A}|^{\frac{1-|\Omega|^T}{1-|\Omega|}} \right)^n$$

Two planning approaches have been recently proposed to make the exhaustive policy search more efficient. The next subsections describe these approaches.

9.6.1.1. *Dynamic programming for DEC-POMDPs*

Hansen *et al.* [HAN 04] introduced the first algorithm to generalize the dynamic programming approach for optimally solving general finite-horizon DEC-POMDPs. Their algorithm is based on the incremental exhaustive enumeration of possible policies and on the elimination of dominated strategies. This work extends the

Algorithm 9.1: Exhaustive generation

Input: A set of policies Q_i^t for horizon t
$k = |\Omega_i|$
for *all action* $a_i \in \mathcal{A}_i$ **do**
  for $p = 1$ ***to*** $|Q_i^t|^k$ **do**
    Let $\langle q_i^{o_1}, \ldots, q_i^{o_k} \rangle_p$ be the pth selection of k policies in Q_i^t, build an new policy q_i^{t+1} with:
    for $j = 1$ ***to*** k **do**
      $q_i^{t+1}(o_j) := q_i^{o_j}$
    $\alpha(q_i^{t+1}) := a_i$
    $Q_i^{t+1} \leftarrow Q_i^{t+1} \bigcup \{q_i^{t+1}\}$

Output: A set of policies Q_i^{t+1} for horizon $(t+1)$

concept of belief states (see Chapter 7) to the decentralized control in multiagent settings. A belief state describes the probabilistic knowledge available to an agent $\mathcal{A}g_i$ at execution step t. It can be seen as the summary of the estimate about the state of the system s_t and the future strategies of the other agents. Let Q_i^t be the set of possible policies of agent $\mathcal{A}g_i$ at time t, and let $\boldsymbol{Q}_{-i}^t = Q_1^t \times \cdots \times Q_{i-1}^t \times Q_{i+1}^t \times \cdots \times Q_n^t$ be the sets of possible future policies for the agents $\mathcal{A}g_j$, $j \neq i$, then a multiagent belief state for agent $\mathcal{A}g_i$ is a distribution over $\mathcal{S}$ and $\boldsymbol{Q}_{-i}^t$:

DEFINITION 9.18 (multiagent belief state). *A* multiagent belief state b_i *for an agent* $\mathcal{A}g_i$ *is a probability distribution over underlying states and future policies for all agents:* $b_i \in \Delta(\mathcal{S} \times \boldsymbol{Q}_{-i})$.

Contrary to single-agent belief states, which are of constant dimensionality, namely the dimensionality of the state space, multiagent belief states are of variable dimensionality: the belief over some other agent's policies effectively depends on the policy space of that agent. It should also be noted that the multiagent belief state reduces to a single-agent belief state in the case of a POMDP (see Chapter 7).

The multiagent belief state completely captures an agent's local partial view of the multiagent environment. It therefore enables the definition of a local value function, defined over the agent's local belief space. If we denote by V_i the value function for agent $\mathcal{A}g_i$, $\boldsymbol{q}_{-i}$ the joint policy for all other agents but agent $\mathcal{A}g_i$, $\langle \boldsymbol{q}_{-i}, q_i \rangle$ a complete joint policy, then the value of policy q_i in belief state $\boldsymbol{b}_i$ can be written as follows:

$$V_i(\boldsymbol{b}_i, q_i) = \sum_{s \in \mathcal{S}} \sum_{\boldsymbol{q}_{-i} \in \boldsymbol{Q}_{-i}} \boldsymbol{b}_i(s, \boldsymbol{q}_{-i}) V(s, \langle \boldsymbol{q}_{-i}, q_i \rangle).$$

It is important to note that a local policy cannot be evaluated by itself and without taking into account the dynamics of the other agents. This is true for competitive as well as for cooperative settings. In this sense, the decentralized control problem can be interpreted as a special case of a cooperative game. In game theory, the notion of best response (see section 8.2.2.4) captures the fact that a policy is the best choice, given that all other players have already made their choice.

DEFINITION 9.19 (best response policy). *We say that $B_i(\boldsymbol{b}_i)$ is the* best response policy *of the agent $\mathcal{A}g_i$ among the set of candidate policies Q_i and for belief state $\mathbf{b}_i$ if*

$$B_i\big(\boldsymbol{b}_i\big) = \arg\max_{q_i \in Q_i} V_i\big(\boldsymbol{b}_i, q_i\big).$$

The best response policy is the policy that optimally completes the behavior of the group, given that all agents $\mathcal{A}g_j$, $j \neq i$, already know which policy they should execute. Determining agent $\mathcal{A}g_j$'s optimal policy however requires that all agents – and thus agent $\mathcal{A}g_j$ – have made a decision about their optimal policy. This obviously leads to circular dependencies, and determining the optimal joint policy through n individual optimizations is usually not possible.

We will see that the dynamic programming approach proceeds the other way around: instead of determining the best response policies, it removes all those policies that can never constitute a best response in any case. We thus extend the concept of *useful vector* (introduced in Chapter 7) to multiagent settings and we introduce the notion of *useful policy*.

A policy is said to be useful if it constitutes a best response for at least one belief state. Otherwise, the policy is said to be dominated.

DEFINITION 9.20 (dominated policy). *A policy $q_i \in Q_i$ is said to be* dominated *if it is a suboptimal policy over the entire belief space, which means that for each belief state $\mathbf{b}_i$, there is at least one other policy $\tilde{q}_i$ that is at least as efficient:*

$$\big(\forall \boldsymbol{b}_i\big)\big(\exists \tilde{q}_i \in Q_i \backslash \{q_i\}\big) \quad \textit{with} \quad V_i\big(\boldsymbol{b}_i, \tilde{q}_i\big) \geq V_i\big(\boldsymbol{b}_i, q_i\big).$$

A policy that is not completely dominated is a useful *policy.*

Instead of computing the best response policies for each agent, Hansen *et al.* [HAN 04] proposed an approach that consists of identifying and pruning all dominated policies. The pruning step consists of solving the following linear program:

$$\begin{aligned}
\text{maximize} \quad & \epsilon \in \mathbb{R} \\
\text{subject to} \quad & V_i\big(\boldsymbol{b}_i, \tilde{q}_i\big) + \epsilon \leq V_i\big(\boldsymbol{b}_i, q_i\big) \qquad \big(\forall \tilde{q}_i \neq q_i\big) \\
\text{with} \quad & \sum_{s \in S} \sum_{\boldsymbol{q}_{-i} \in \boldsymbol{Q}_{-i}} \boldsymbol{b}_i\big(s, \boldsymbol{q}_{-i}\big) = 1 \\
& \boldsymbol{b}_i\big(s, \boldsymbol{q}_{-i}\big) \geq 0 \qquad\qquad\qquad (\forall s \in \mathcal{S})\big(\forall \boldsymbol{q}_{-i} \in \boldsymbol{Q}_{-i}\big).
\end{aligned}$$

If the result of the linear program is non-positive ($\epsilon \leq 0$), then the policy q_i is dominated and can be removed.

Algorithm 9.2: Dynamic programming for DEC-POMDPs

Input: A DEC-POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R} \rangle$ and a horizon T

for *all agent* $\mathcal{A}g_i$*:* **do**

  Initialize $Q_i^0 \leftarrow \emptyset$

for $t = 1$ ***to*** $t = T$ **do**

  for *all agent* $\mathcal{A}g_i$*:* **do**

    $Q_i^t \leftarrow \texttt{ExhaustiveGeneration}(Q_i^{t-1})$

  repeat

    Find an agent $\mathcal{A}g_i$ and a policy $q_i \in Q_i^t$ with

$$\forall \boldsymbol{b}_i,\ \exists \tilde{q}_i \in Q_i^t \backslash \{q_i\} \text{ such that } V_i\big(\boldsymbol{b}_i, \tilde{q}_i\big) \geq V_i\big(\boldsymbol{b}_i, q_i\big)$$

    $Q_i^t \leftarrow Q_i^t \backslash \{q_i\}$

  until *there is no more possible pruning*

Output: A set of useful policies for each agent

In the multiagent dynamic programming algorithm proposed by Hansen *et al.*, a generation step takes a set of policies of horizon t as input, and builds the exhaustive set of policies of horizon $t+1$ such that each policy of the new set contains a policy that appears in the original set. A pruning step then traverses the new policy set, examines whether each one of these policies is indeed useful and eliminates dominated policies.

This process is repeated for each horizon, starting from horizon 1. Algorithm 9.2 sums up the dynamic programming approach for DEC-POMDPs. Choosing an optimal joint policy for the initial state s_0 is then obtained by maximizing:

$$\boldsymbol{q}_{s_0}^* = \underset{\boldsymbol{q}^T \in Q_1^T \times \cdots \times Q_n^T}{\arg\max} V\big(s_0, \boldsymbol{q}^T\big).$$

9.6.1.2. *Heuristic search for DEC-POMDPs*

Szer *et al.* [SZE 05] proposed another approach for optimally solving finite-horizon DEC-POMDPs. This approach, called MAA* (multiagent A*), extends the heuristic search algorithm A* to multiagent settings.[5]

5. A variant of A*, called LAO*, has already been described in Chapter 6, section 6.2.3.3. It extends A* to solve mono-agent MDPs.

While dynamic programming algorithms build policies from the last decision step to the first decision step and prune dominated strategies at each step, MAA* builds policies from the first decision step to the last one and makes use of a heuristic function to eliminate dominated strategies. As proved by Szer *et al.* [SZE 05], MAA* is both complete and optimal for finite horizon DEC-POMDPs.

The algorithm incrementally develops the search tree of policies: the leaf nodes of the tree describe partial solutions and at each iteration the solution that seems to be the most promising is expanded for the next step. A leaf node of the policy tree stands for a joint policy up to horizon $t < T$. Expanding the joint policy $\boldsymbol{q}^t$ at horizon t consists of building all the children of $\boldsymbol{q}^t$, i.e. the joint policies $\boldsymbol{q}^{t+1}$ at horizon $t+1$ that consists of applying $\boldsymbol{q}^t$ up to horizon t and then executing one of the possible actions at horizon $t+1$. Figure 9.5 describes a piece of the search tree for the expansion process from horizon $t = 2$ to horizon $t + 1 = 3$.

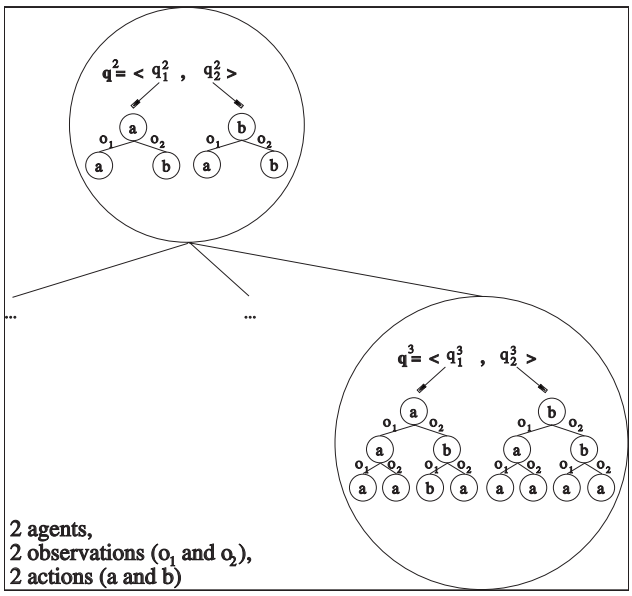

Figure 9.5. *A piece of the search tree. One of the children of the joint policy of horizon 2 is expanded to build a joint policy of horizon 3*

In order to decide which leaf node is the most promising, the algorithm calculates the exact value of the joint policy $\boldsymbol{q}^t$ that has been developed up to horizon t and a heuristic function H^{T-t} is then used to estimate the expected value of the policy to come. Δ^{T-t} is introduced as a completion of an arbitrary depth-t joint policy, which means a set of depth-(T-t) policy trees that can be attached at the leaf nodes of a joint policy $\boldsymbol{q}^t$ such that $\langle \boldsymbol{q}^t, \Delta^{T-t} \rangle$ constitutes a complete joint policy of depth T.

The difference between the values $V(s_0, \boldsymbol{q}^t)$ and $V(s_0, \langle \boldsymbol{q}^t, \Delta^{T-t} \rangle)$ defines the value of the completion Δ^{T-t}:

$$V\big(\Delta^{T-t} \mid s_0, \boldsymbol{q}^t\big) := V\big(s_0, \langle \boldsymbol{q}^t, \Delta^{T-t} \rangle\big) - V\big(s_0, \boldsymbol{q}^t\big).$$

Similarly, the value of a joint policy can be calculated as the sum of the value of a root and the value of the completion:

$$V\big(s_0, \langle \boldsymbol{q}^t, \Delta^{T-t} \rangle\big) = V\big(s_0, \boldsymbol{q}^t\big) + V\big(\Delta^{T-t} \mid s_0, \boldsymbol{q}^t\big).$$

Given a partially built policy $\boldsymbol{q}^t$, the upper bound on the value of the joint policy, based on $\boldsymbol{q}^t$ and defined up to horizon T, can be calculated using the value of the best possible completion:

$$\overline{V}^{*T}\big(s_0, \boldsymbol{q}^t\big) = V\big(s_0, \boldsymbol{q}^t\big) + \max_{\Delta^{T-t}} V\big(\Delta^{T-t} | s_0, \boldsymbol{q}^t\big).$$

An exact computation of this value requires an exhaustive search in the policy space and quickly becomes intractable. This is why MAA* makes use of the heuristic function H^{T-t} which estimates the value of the best completion.

DEFINITION 9.21 (multiagent heuristic function). *The heuristic value function that guides the search of the best completion must overestimate the actual expected reward for any completion of a joint policy $\boldsymbol{q}^t$:*

$$H^{T-t}\big(s_0, \boldsymbol{q}^t\big) \geq \max_{\Delta^{T-t}} V\big(\Delta^{T-t} \mid s_0, \boldsymbol{q}^t\big).$$

Such a heuristic function is said to be an admissible heuristic.

Given an admissible heuristic H, the value function of the algorithm MAA* can then be defined as follows:

$$\overline{V}^{T}\big(s_0, \boldsymbol{q}^t\big) = V\big(s_0, \boldsymbol{q}^t\big) + H^{T-t}\big(s_0, \boldsymbol{q}^t\big) \geq \overline{V}^{*T}\big(s_0, \boldsymbol{q}^t\big).$$

The core part of the search algorithm is the definition of an admissible heuristic function. As pointed out by [LIT 95] and later by [HAU 00] for the single-agent case, an upper bound for the value function of a POMDP can be obtained through the underlying completely observable MDP. The value function of a POMDP is defined over the belief states (see Chapter 7). The optimal value for a belief state of a POMDP can then be approximated as follows:

$$V^*_{\text{POMDP}}(\boldsymbol{b}) \leq \sum_{s \in \mathcal{S}} \boldsymbol{b}(s) V^*_{\text{MDP}}(s).$$

Although it is currently not known how to evaluate a value function over belief states in the multiagent case, a similar upper bound property can still be stated for decentralized POMDPs. Let $P(s \mid s_0, \boldsymbol{q})$ be the probability that the system is in state s after executing the joint policy $\boldsymbol{q}$ from s_0. A heuristic h that overestimates the optimal value of the DEC-POMDP can then be defined such that

$$h^t(s) \geq V^{*t}(s).$$

This allows for defining the following class of heuristic functions:

$$H^{T-t}(s_0, \boldsymbol{q}^t) := \sum_{s \in \mathcal{S}} P(s \mid s_0, \boldsymbol{q}^t) h^{T-t}(s). \tag{9.1}$$

Intuitively, any such heuristic is optimistic because it effectively simulates the situation where the real underlying state is revealed to the agents after execution of the joint policy $\boldsymbol{q}^t$. It has been proved [SZE 05] that any heuristic function H as defined in 9.1 is admissible if h is admissible.

The computation of a good heuristic function is in general much easier than the computation of the exact value. Choosing a heuristic function among admissible heuristics results from a trade-off between the quality of the estimate and the computation cost. Nonetheless, it is often difficult to decide *a priori* which heuristic would be the best one. Szer *et al.* described several admissible heuristics for DEC-POMDPs:

– *The* MDP *heuristic*: An easy way to calculate a value function heuristic is to use the solution of the underlying centralized MDP with the remaining finite horizon $T - t$. Solving an MDP can be done using dynamic programming or search technique and requires only polynomial time (see Chapter 1). Using the underlying MDP as an admissible heuristic for search in POMDPs has already been applied to the single-agent case as described in [WAS 96] and later in [GEF 98].

– *The* POMDP *heuristic*: A tighter yet more complex value function heuristic consists of using the solution to the underlying partially observable MDP. Solving POMDPs is PSPACE-complete [PAP 87] and usually involves linear programming. Although this heuristic is more difficult to calculate, it leads to better performance of MAA*.

– *The* DEC-POMDP *heuristic*: an important special case of a value function heuristic consists of the optimal value itself. One way to calculate this value efficiently is to apply MAA* again:

$$h^{T-t}(s) := V^{T-t}_{\text{DEC-POMDP}}(s) = MAA^{*T-t}(s). \tag{9.2}$$

This leads to a recursive approach, where a call to MAA^{*T} invokes several calls to MAA^{*T-1} and where each of them triggers new sub-searches.

9.6.1.3. *Optimal algorithms for sub-classes of DEC-POMDPs*

Based on the work of Goldman and Zilberstein about the influence of problem properties on DEC-POMDP's complexity, Becker *et al.* have developed an optimal algorithm for solving specific sub-classes of DEC-POMDPs. CSA (*coverage set algorithm*) allows for solving event-driven DEC-MDPs [BEC 04a] or DEC-MDPs with independent transitions and independent observations [BEC 03, BEC 04b].

Although CSA has been presented in the two-agent case, it can be used to solve larger problems involving more agents. CSA consists of three steps (which are detailed below): 1) create *augmented* MDPs, 2) find an *optimal coverage set* for the augmented MDPs, 3) find the optimal joint policy. If we consider two agents $\mathcal{A}g_i$ and $\mathcal{A}g_j$, an *augmented* MDP formalizes the decision problem in which agent $\mathcal{A}g_i$ must calculate its optimal policy given a fixed policy for agent $\mathcal{A}g_j$.

The first step of CSA thus consists of defining, for each policy of $\mathcal{A}g_j$, an augmented MDP. The second step of the algorithm calculates the optimal policy for each augmented MDP: the optimal policy of $\mathcal{A}g_i$ for each possible policy of $\mathcal{A}g_j$ is calculated. This set of policies defines the optimal *coverage set*. Computing this set aims at minimizing the space of possible policies while performing an exhaustive search: for each possible policy π_j of $\mathcal{A}g_j$, the best response π_i^* of $\mathcal{A}g_i$ is calculated and only the joint policy $\pi = \langle \pi_j, \pi_i^* \rangle$ is added to the coverage set. Joint policies such as $\pi = \langle \pi_j, \pi_i \neq \pi_i^* \rangle$ are ignored. Finally, the third and last step of the algorithm consists of searching for the best joint policy in the coverage set.

This algorithm has a complexity exponential in the number of states of the system. While solving ED-DEC-MDPs, it has been proved that the complexity is doubly exponential in the number of dependencies. The worst-case complexity of the CSA for solving ED-DEC-MDPs is equivalent to the complexity of the exhaustive search. This worst-case complexity explains why the scalability of CSA remains limited. Indeed, even if the complexity of the problems that can be solved by CSA is less important than the complexity of general DEC-POMDPs, solving large sub-classes of DEC-POMDPs remains a hard problem. Experimental results showed that CSA can only solve ED-DEC-MDPs involving few agents few dependencies (results are presented for 2 agents and 4 dependencies). While increasing the number of agents or the number of dependencies, the number of states of the ED-DEC-MDPs quickly becomes untractable.

9.6.2. *Approximate algorithms*

The approaches that have just been described all calculate an optimal solution to DEC-POMDPs or DEC-MDPs. Because of the high complexity of optimally solving DEC-POMDPs and DEC-MDPs, these approaches quickly become untractable. This is why approximate algorithms have recently been developed. In the following section,

we present some of these approximate solutions for both finite and infinite horizon problems.

9.6.2.1. *Heuristics and approximate dynamic programming*

As explained in the previous section, all exact algorithms suffer from high complexity, and using a heuristic value function to guide the policy search remains insufficient to obtain more scalable algorithms. Heuristic approximation methods have been proposed to reduce the complexity. Up to now, such heuristics mainly apply to dynamic programming approaches [SZE 06, SEU 08, SEU 07] and heuristic search [OLI 07b].

Szer *et al.* [SZE 06], for example, address two major drawbacks of the exact dynamic programming approach, namely the computationally expensive linear programming part necessary to identify dominated policies, and the difficulty to exclude those regions of the belief space that are never reachable. They introduce a point-based multiagent dynamic programming algorithm that allows us to reduce the space of belief states by determining those belief states that are likely to be visited in a real-world scenario. Moreover, Szer *et al.* propose to limit the set of possible joint policies by sampling from the set of possible prior policies. Only those policies that are most likely to be spread out far away from each other are retained. The Manhattan distance can be used as a simple metric between policy trees.

Szer *et al.* have thus been able to solve the multi-access channel problem up to horizon 8 whereas previous approaches were only able to solve this problem up to horizon 4. Other efficient heuristics [SEU 08, SEU 07] even enable to solve the same problems up to much larger horizons (horizon 100,000). Nonetheless, further experiments are necessary to estimate the effect of these heuristics on the solution quality.

In the context of heuristic search approaches, different kinds of heuristics can be used in order to speed up the search. They make use of approximations such as:

– performing incomplete searches (by limiting the branching factor for instance);

– estimating the value of a policy using simulation methods (Monte Carlo approach) instead of computing their exact value;

– computing the heuristic value function for the most likely belief states.

Recently, Dibangoye *et al.* [DIB 09a] have presented a version of the dynamic programming value iteration algorithm that provides an error bound on the quality of the returned solution. The Point-Based Value Iteration (PBVI) algorithm proposed by Dibangoye *et al.* is the first scalable algorithm for infinite-horizon DEC-POMDPs with provable error bound. Joint policies are represented as joint deterministic finite-state controllers (DJFSC). As it is the case for the mono-agent value iteration algorithm,

at each iteration step the PBVI algorithm first updates the value function and then updates the current DJFSC.

A modified version of the policy iteration algorithm has also been proposed: the Point-Based Policy Iteration algorithm (PBPI) [DIB 09b]. In order to speed up the algorithm and to improve its scalability, Dibangoye *et al.* make use of some adapted heuristic search methods: the point-based heuristic and the point-based heuristic with branch and bound.

Experimental results on PBPI and PBVI algorithms show some orders of magnitude of improvements in their performance compared to other existing algorithms dealing with dynamic programming and DEC-POMDPs. It has also been shown that the PBPI algorithm scales up to larger sizes of problems and is able to successfully solve many DEC-POMDP benchmarks.

9.6.2.2. *Bounded memory*

The major limitation of the general dynamic programming algorithm for DEC-POMDPs is the explosion in memory requirements: for each additional horizon, a super-exponential number of new value functions has to be considered [HAN 04]. Instead of fixing the problem horizon and searching for an optimal policy, Bernstein *et al.* [BER 05] therefore propose an alternative approach based on first bounding the size of the policy and then trying to optimize its parameters. In this case, policies are based on finite state controllers, which can be executed for an infinite amount of time. This algorithm thus constitutes one way of solving infinite-horizon DEC-POMDPs. Each agent's policy is represented as a stochastic finite state controller and a correlation device is used to allow the agents to coordinate their policies.

The policy iteration algorithm described in [BER 05] updates, at each iteration step, the parameters of a controller so as to improve the value function of the system. Convergence to a global optimum is not guaranteed but the backup applied to a local controller provably produces a joint policy with value at least as high as before the backup. The algorithm thus reaches a local optimum.

Experiments showed that increasing the number of nodes of each controller leads to higher quality but also increases the memory requirements for computing the solution. Since each iteration takes polynomial time and the algorithm uses a bounded amount of memory, this approach solves larger problems than exact dynamic programming algorithms. Nonetheless, this approach remains limited to problems involving a small number of agents since the time complexity of each iteration is exponential in the number of agents.

9.6.2.3. *Co-evolutive algorithms*

Some approximate algorithms for solving decentralized MDPs are based on a co-evolutive iterative approach. These algorithms consist of iteratively improving the

policy of a single agent by fixing the policies of the other agents. For an n-agent DEC-POMDP, a typical co-evolutive algorithm selects at each step an agent $\mathcal{A}g_k$ while the policies of the agents $\mathcal{A}g_i$, $i \neq k$, remain fixed. The co-evolutive algorithm then calculates an optimal policy for $\mathcal{A}g_k$ in response to these fixed policies. The process is repeated until none of the policies can be improved.

Chadès *et al.* [CHA 02, SCH 02] introduced the first co-evolutive algorithm for solving DEC-POMDPs. They first described two co-evolutive algorithms that calculate a Nash equilibrium for problems formalized as MMDPs.

Chadès *et al.* extended their approach to partially observable systems under decentralized control (i.e. problems formalized by DEC-POMDPs). In order to take into account partial observability of the system state, they introduced subjective-MDP. A subjective-MDP is an MDP where states are defined by the agent's perceptions. This model is similar to a POMDP where the decision is based on local perceptions and we do not consider probabilities on the underlying state neither histories of observations.

At each step of the co-evolutive algorithm, an agent $\mathcal{A}g_k$ is selected and the subjective-MDP of this agent is defined given the policies of the other agents. The subjective-MDP is then solved and a new policy is obtained for the agent $\mathcal{A}g_k$. Because a subjective-MDP is not Markovian, this policy is not optimal. In decentralized settings with partial observability, the co-evolutive algorithm proposed by Chadès *et al.* is thus not guaranteed to converge to a Nash equilibrium. Convergence to a Nash equilibrium is only obtained under full observability or centralized control (the problem is then formalized as an MMDP).

Nair *et al.* [NAI 03] have also proposed an algorithm based on co-evolutive principles. JESP (*Joint Equilibrium Based Search for Policies*) enables the solving of sequential multiagent decision problems formalized by MTDPs with independent observations.

At each iteration step, the policy of an agent is modified in order to improve the joint policy. The algorithm provably converges to a Nash equilibrium. This algorithm has the same worst-case complexity as the exhaustive search. It may, however, perform much better in practice.

Nair *et al.* proposed an improvement of JESP in terms of performance by using dynamic programming. DP-JESP (*Dynamic Programming Joint Equilibrium-based Search for Policies*) exploits Bellman's optimality principle to speed up policy computation (each policy is incrementally evaluated and built using Bellman's optimality principle). Thus, on the multiagent tiger problem, experiments show that JESP can solve the problem up to horizon 3 whereas DP-JESP solves the problem up to horizon 7. Nonetheless, since the complexity of the algorithm remains exponential,

larger problems are difficult to solve. Nair *et al.* have therefore been interested in exploiting the structure of the problem to propose another variant of their algorithm.

LID-JESP algorithm (*Locally Interacting Distributed Joint Equilibrium-based Search for Policies*) [NAI 05] combines the JESP algorithm with principles from the field of distributed constraint optimization. LID-JESP solves problems formalized by a Network-Distributed POMDP with independent transitions and independent observations. It exploits the locality of interactions between the agents to speed up the running time of JESP. An interaction graph describes the interaction between the agents. The agents are the vertices and edges describe how the actions of an agent influence the reward of the other agents. A neighborhood is thus defined for each agent $\mathcal{A}g_i$: it describes the set of agents $\mathcal{A}g_j$ whose utility is influenced by $\mathcal{A}g_i$'s actions. This neighborhood is then exploited in order to limit the set of agents to consider while revising the policy of $\mathcal{A}g_i$.

9.6.2.4. *Gradient descent for policy search*

In Chapter 5, it has been shown that optimization methods can be used to solve sequential decision problems using policy search. These methods can also be used in multiagent settings while searching to optimize a set of controllers instead of optimizing a single controller. Methods based on performing a gradient descent in policy space have been used for finding locally optimal solutions to multiagent sequential decision problems [PES 00, DUT 01, TAO 01]. More recently, the cross-entropy method, a method for combinatorial optimization, has been used to speed up policy search and to find approximate solutions of DEC-POMDPs [OLI 07a].

In cooperative multiagent settings, gradient descent methods can be executed in a centralized way by all the agents or in a decentralized way by each agent (each agent receives the same reward). The controller can be, for example, a finite state automaton [PES 00] or a function approximator (whose input is the current observation or a belief state) [DUT 01, TAO 01]. These learning approaches result in a local optimum. Nonetheless, unlike the JESP algorithm, this local optimum may not be a Nash equilibrium.

9.6.2.5. *Bayesian games*

Emery-Montemerlo *et al.* [EME 04] proposed a decentralized algorithm for DEC-POMDPs that approximates the problem with a series of Bayesian games.[6] These Bayesian games together approximate the initial multiagent decision problem.

The approach alternates planning and policy execution. Each planning step consists of solving a one-step Bayesian game for the next execution step. The

6. A Bayesian game is a strategic form game with incomplete information.

calculated policy is then executed and a new planning phase starts for the next execution step. Each Bayesian game is solved using a heuristic that allows for estimating the utility function. The performance of the approach is closely related to the kind of problems and the heuristics that are considered.

9.6.2.6. *Heuristics for communicating agents*

Goldman and Zilberstein [GOL 03] described a myopic greedy approach to solve the "meeting under uncertainty" problem formalized as a DEC-POMDP-COM. This problem involves two agents that have to meet at some location on a grid as soon as possible. The moves of the agents are uncertain and each agent cannot observe the location of the other agent. Nevertheless, the agents are able to communicate information about their position and to revise the meeting point consequently. Communication can therefore allow the agents to meet faster.

Since communicating is costly, a part of the decision problem consists of deciding when to communicate. If the communication cost is prohibitive, the optimal policy consists of never communicating. If communication is free, the optimal policy consists of communicating at each time step. In general, the optimal policy is between these two extremes. Goldman and Zilberstein described a myopic approach to approximate this optimal policy. At each time step, the agents optimize the choice of when to communicate assuming that they can communicate only once. This myopic approach allows the agents to exchange information only when communicating is particularly beneficial.

Other kinds of communication heuristics have been proposed by Xuan *et al.* [XUA 01] for the meeting under uncertainty problem. Although, these heuristics perform well on the meeting under uncertainty problem, they may be inappropriate for other kinds of problems. Thus, there is a need for approaches that are able to solve the general communication decision problem.

9.6.2.7. *Approximate solutions for OC-DEC-MDPs*

Based on the OC-DEC-MDP model described in section 9.5.3.2, Beynier *et al.* [BEY 05, BEY 06] developed some efficient algorithms for computing an approximation of the optimal solution even for large multiagent decision problems.

9.6.2.7.1. Opportunity cost and coordination

Given the problem decomposition which is part of the OC-DEC-MDP model, Beynier *et al.* proposed to simultaneously solve each local MDP and to deduce each agent's policy. Since they consider cooperative multiagent systems where the interactions between the agents arise from precedence and time constraints, maximizing a common reward requires the agents to coordinate – the local MDPs cannot be solved independently.

For the purpose of coordinating the agents, Beynier *et al.* introduced the notion of opportunity cost. The Opportunity Cost is borrowed from economics where it refers to hidden indirect costs associated with a decision. In their work, Beynier *et al.* use the opportunity cost to measure the indirect effect of an agent's decision on other agents. In order to obtain cooperative behaviors, the best action an agent $\mathcal{A}g_i$ can execute from a state s_i then results from a trade-off between:

– the expected utility Q of the agent, and

– the opportunity cost induced on the other agents.

Thus, the policy of an agent $\mathcal{A}g_i$ from a state s_i is calculated using the following equation:

$$\pi_i(s_i) = \arg\max_{a_i \in A_i} \big(Q_i(s_i, a_i) - OC(s_i, a_i)\big), \tag{9.3}$$

where A_i stands for the set of actions a_i the agent $\mathcal{A}g_i$ can execute from s_i, $Q_i(s_i, a_i)$ is the expected utility of the agent $\mathcal{A}g_i$ while executing a_i from s_i and $OC(s_i, a_i)$ is the opportunity cost induced on the other agents.

In fact, the opportunity cost stands for a difference in expected utility. It measures the loss in expected utility incurred by the other agents when they deviate from their best policy. Several opportunity cost evaluation algorithms have been proposed. Especially, Beynier *et al.* introduced the expected opportunity cost that accurately measures the influence of one agent on the other agents. Approximations of opportunity costs have also been presented.

9.6.2.7.2. Policy improvement using opportunity costs

In order to efficiently solve problems formalized by OC-DEC-MDPs, Beynier *et al.* described a revision algorithm that calculates an approximate solution respecting time, precedence and resource constraints on task execution. This algorithm starts with an initial policy set and tries to improve it using equation 9.3. Two versions of the algorithm have been developed. A centralized one allows a central entity to revise all agents' policies. It passes through the set of tasks and improves the agent's execution policy of each task. A decentralized version of the algorithm has also been proposed. Thus, each agent improves its own policy and individual policies are simultaneously considered. This second version requires offline communication of opportunity cost values.

Beynier *et al.*'s revision algorithm does not guarantee the optimality of the solution unless some properties hold such as: resources are unlimited, time constraints do not restrict execution tasks, the graph is a two-agent tree with one-way precedence constraints, etc. In general, an approximate solution is obtained. Despite other existing approaches whose complexity is exponential, Beynier *et al.* proved that

both versions of their algorithm have a time complexity polynomial in the number of states and actions. Moreover, precedence, time and resources constraints are exploited to limit the state and the action spaces. Thus, large problem sizes can be handled. Experiments demonstrated that missions involving more than ten agents and hundreds of tasks can be solved. This approach shows that exploiting the properties of the problem and computing an approximate solution is a promising approach to develop more scalable approaches. Such an approach has also been adopted by Nair *et al.* [NAI 05] to efficiently solve DEC-POMDPs in distributed networks (ND-POMDPs, Networked-Distributed POMDPs).

9.6.2.7.3. Iterative policy improvement

The revision algorithm described above consists of improving an initial policy. When it stops, each task has been revised once and a new local policy is available for each agent. In order to obtain better solutions, Beynier *et al.* proposed to re-execute the revision algorithm considering that the initial policy is the policy that has just been calculated [BEY 06].

At each iteration step, the agents improve their initial local policy at the same time. The outcome policies of iteration $N-1$ are the initial policies of iteration N. This process is repeated until no changes are made on the policies during one iteration. Two versions of the iterative algorithm – which are based on the versions (centralized and decentralized) of the revision algorithm – have been proposed.

The complexity of one iteration is the same as the complexity of the revision algorithms (polynomial in the number of states and actions). The time complexity of this iterative algorithm is thus mainly influenced by the number of iterations. It has been shown that convergence guarantees rely on the method that is used to calculate the opportunity cost. The algorithm converges if we use an estimation of the opportunity cost that accurately measures the influence of a decision on the other agents. In this case, experiments show that convergence is reached within a small number of iterations (most of the time less than 4 iteration steps).

Finally, experiments proved that the iterative algorithm can also be used to solve large problems (more than 10 agents and 100 tasks). Moreover, as explained in the next section, it has been shown that this approach is suitable to solve real-world multi-robot decision problems.

9.7. Applicative scenario: multirobot exploration

In order to prove the applicability of their approach to real-world multiagent decision problems, Beynier *et al.* [BEY 05, BEY 06] applied OC-DEC-MDPs to multirobot exploration scenarios. They considered variants of the Mars rover exploration scenarios (Chapter 14 introduces a similar application but it considers different difficulties from the ones considered by Beynier *et al.*).

A scenario, similar to Mars rover missions, was therefore implemented. It involves two robots that have to explore a set of 8 interesting places (Figure 9.6). The first robot (robot $\mathcal{A}g_1$) can take pictures and the second one (robot $\mathcal{A}g_2$) can collect and analyze ground samples. Robot $\mathcal{A}g_1$ must take picture of sites A, B, D, E, F, H and J, and robot $\mathcal{A}g_2$ must analyze sites C, D, F, H and I. Sites are ordered so as to minimize traveling resource consumptions. Furthermore, precedence constraints have to be taken into account. As taking samples of the ground may change the topology of the site, pictures of a site must be taken before the other robot starts to analyze it. Moreover, robot $\mathcal{A}g_1$ must have left a site before robot $\mathcal{A}g_2$ can start to analyze it. Thus, robot $\mathcal{A}g_1$ must have taken a picture of site D before robot $\mathcal{A}g_1$ enters this site. Time constraints have also to be considered: visiting earliest start times and latest end times are associated with each site.

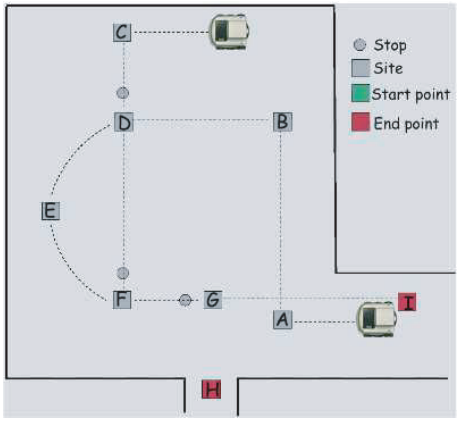

Figure 9.6. *Mission scenario*

The mission was represented using a mission graph. Then, the corresponding OC-DEC-MDP was automatically built and solved by the iterative algorithm. Finally, resulting policies were implemented on Koala robots. During task execution, robots only have to execute their policies which map each state to an action and thus limit the computational resources needed to make a decision. Coordination is performed without communication between the agents during the execution, and time and precedence constraints are respected. As shown in Figure 9.7 for the crossing point D, while deciding when to start its action, the first robot takes into account the fact that the other robot waits for him (thanks to the OC). The decision of the second robot is based on the probability that robot $\mathcal{A}g_1$ has left the site, the cost of a partial failure, and the robot's own expected value. Thus, robot 1 enters site D, completes its task (Picture 2) and leaves the site (Picture 3). As robot 1 does not know the other robot's actions, it may try to enter the site and fails because the other robot has not

Figure 9.7. *Execution of the mission by 2 Koala robots (crossing D)*

finished to take the picture. The second robot realizes that it fails when it tries to enter the site. If precedence constraints are not respected, the robot returns to its last position. If time constraints are respected, the robot enters the site (Picture 4). These experiments show that the policies calculated by the OC-DEC-MDP approach can be used by physical robots which are thus able to successfully and cooperatively complete their mission.

Experiments dealing with larger scenarios have been performed on simulators [BEY 05, BEY 06]. They proved that multirobot missions composed of about ten agents and hundreds of tasks can be planned and executed by the OC-DEC-MDP approach (problems up to 20 agents and 800 tasks were solved) with high performance. This approach is thus scalable enough to formalize and solve problems like the ones considered in Mars exploration researches (regarding Mars exploration, the mission to execute is sent to the robots once a day and may involve about ten robots that have to complete hundreds of tasks).

9.8. Conclusion and outlook

In this chapter, we introduced several extensions of MDPs and POMDPs to formalize decision problems in cooperative multiagent systems acting in stochastic environments. We first described the MMDP model to formalize decision problems where each agent observes the system state. We then turned to models that allow for formalizing problems where each agent has a partial observability of the system. We thus introduced DEC-POMDPs and DEC-MDPs, and we reviewed several variants of these models.

In the second part of this chapter, we described algorithms for solving the problems that were formalized in section 9.4. We described optimal algorithms that solve general classes of DEC-POMDPs. Given the high complexity of finding an optimal solution, these algorithms can only solve very small problems. Two kinds of approaches have been proposed to tackle this high complexity. The first set of approaches tries to identify specific properties of the problems (such as transition independence or observation independence) that reduce the complexity. Other approaches develop approximate algorithms that find an approximation of the optimal policy instead of searching for an exact optimal policy. Several approximate algorithms have been described in section 9.6.2.

In this chapter we also showed that some difficulties arise while formalizing real-world multiagent decision problems with DEC-POMDPs. Indeed, DEC-POMDPs suffer from limited representation of time and actions: it is assumed that all the actions have the same duration and they do not take into account constraints on action execution. This issue has been tackled by ED-DEC-MDP and OC-DEC-MDP models described in section 9.5. Nevertheless, these models allow for representing limited sets of constraints. In order to widely apply these models to real-world decision problems, it would thus be necessary to formalize richer sets of constraints.

Although recent works dealing with solving DEC-MDPs have demonstrated the efficiency of approximate algorithms, quality guarantees are lacking. Indeed, existing approximate approaches evaluate the quality of calculated policies in terms of equilibria. For instance, JESP algorithms returns a Nash equilibrium. Nonetheless, for a given problem there may exist several solutions that are Nash equilibria with different qualities. Nash equilibria being local optima, such solutions can lead to high or poor performance depending on the problems and of the equilibrium that is selected. Thus, characterizing the solution in terms of equilibria does not give real guarantees on the quality of the solutions and comparing the performance of different approaches is very difficult. At the moment, few approaches provide an error bound on the solution quality whose evaluation is an important issue. There is thus a need for developing approximate algorithms that provide an upper bound on the distance between the solutions that are calculated and the optimal solution (ϵ-optimal algorithms).

9.9. Bibliography

[BEC 03] BECKER R., ZILBERSTEIN S., LESSER V. and GOLDMAN C., "Transition-independent decentralized Markov decision processes", *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, ACM Press, New York, pp. 41–48, 2003.

[BEC 04a] BECKER R., LESSER V. and ZILBERSTEIN S., "Decentralized Markov decision processes with event-driven interactions", *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, New York, pp. 302–309, 2004.

[BEC 04b] BECKER R., ZILBERSTEIN S., LESSER V. and GOLDMAN C., "Solving transition independent decentralized Markov decision processes", *Journal of Artificial Intelligence Research*, vol. 22, pp. 423–455, 2004.

[BEC 05] BECKER R., LESSER V. and ZILBERSTEIN S., "Analyzing myopic approaches for multi-agent communication", *Proceedings of the International Conference on Intelligent Agent Technology (IAT)*, Compiegne, France, pp. 550–557, 2005.

[BER 01] BERNSTEIN D., ZILBERSTEIN S., WASHINGTON R. and BRESINA J., "Planetary rover control as a Markov decision process", *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Montreal, Canada, 2001.

[BER 02] BERNSTEIN D. S., GIVAN R., IMMERMAN N. and ZILBERSTEIN S., "The complexity of decentralized control of Markov decision processes", *Mathematics of Operations Research*, vol. 27, no. 4, pp. 819–840, 2002.

[BER 05] BERNSTEIN D., HANSEN E. A. and ZILBERSTEIN S., "Bounded policy iteration for decentralized POMDPs", *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, Edinburgh, Scotland, 2005.

[BEY 05] BEYNIER A. and MOUADDIB A. I., "A polynomial algorithm for decentralized Markov decision processes with temporal constraints", *Proceedings of the 4th International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS'05)*, The Netherlands, pp. 963–969, 2005.

[BEY 06] BEYNIER A. and MOUADDIB A. I., "An iterative algorithm for solving constrained decentralized Markov decision processes", *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, Boston, MA, pp. 1089–1094, 2006.

[BOU 96] BOUTILIER C., "Planning, learning and coordination in multiagent decision processes", *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge (TARK'96)*, Morgan Kaufmann, San Francisco, CA, pp. 195–201, 1996.

[BOU 99a] BOUTILIER C., DEAN T. and HANKS S., "Decision-theoretic planning: structural assumptions and computational leverage", *Journal of Artificial Intelligence Research*, vol. 11, pp. 1–94, 1999.

[BOU 99b] BOUTILIER G., "Sequential optimality and coordination in multiagent systems", *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, Stockholm, Sweden, pp. 478–485, 1999.

[CHA 02] CHADÈS I., SCHERRER B. and CHARPILLET F., "A heuristic approach for solving decentralized-POMDP: assessment on the pursuit problem", *Proceedings of the 2002 ACM symposium on Applied computing (SAC'02)*, Madrid, Spain, pp. 57–62, 2002.

[DEC 93] DECKER K. and LESSER V., "Quantitative modeling of complex computational task environments", *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93)*, pp. 217–224, 1993.

[DIB 09a] DIBANGOYE J. S., MOUADDIB A. I. and CHAIB-DRAA B., "Point-based incremental pruning heuristic for solving finite horizon DEC-POMDPs", *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Budapest, Hungary, pp. 569–576, 2009.

[DIB 09b] DIBANGOYE J. S., MOUADDIB A. I. and CHAIB-DRAA B., "Policy iteration algorithms for DEC-POMDPs with discounted rewards", *AAMAS 2009 Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains*, Budapest, Hungary, 2009.

[DUT 01] DUTECH A., BUFFET O. and CHARPILLET F., "Multi-agent systems by incremental gradient reinforcement learning", *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, Seattle, WA, pp. 833–838, 2001.

[EME 04] EMERY-MONTEMERLO R., GORDON G., SCHNEIDER J. and THRUN S., "Approximate solutions for partially observable stochastic games with common payoffs", *Proceedings of the 3rd Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, New York, pp. 136–143, 2004.

[GEF 98] GEFFNER H. and BONET B., "Solving large POMDPs by real time dynamic programming", *Working Notes – Fall AAAI Symposium on POMDPs*, Orlando, FL, 1998.

[GOL 03] GOLDMAN C. and ZILBERSTEIN S., "Optimizing information exchange in cooperative multi-agent systems", *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, Melbourne, Australia, pp. 137–144, 2003.

[GOL 04] GOLDMAN C. and ZILBERSTEIN S., "Decentralized control of cooperative systems: categorization and complexity analysis", *Journal of Artificial Intelligence Research*, vol. 22, pp. 143–174, 2004.

[HAN 04] HANSEN E. A., BERNSTEIN D. S. and ZILBERSTEIN S., "Dynamic programming for partially observable stochastic games", *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*, San Jose, CA, pp. 709–715, 2004.

[HAU 00] HAUSKRECHT M., "Value-function approximations for partially observable Markov decision processes", *Journal of Artificial Intelligence Research*, vol. 13, pp. 33–94, 2000.

[KAE 98] KAELBLING L. P., LITTMAN M. L. and CASSANDRA A., "Planning and acting in partially observable stochastic domains", *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[LIT 95] LITTMAN M. L., CASSANDRA A. R. and KAELBLING L. P., "Learning policies for partially observable environments: scaling up", *Proceedings of the 12th International Conference on Machine Learning (ICML'95)*, Morgan Kaufmann, San Francisco, CA, pp. 362–370, 1995.

[NAI 03] NAIR R., TAMBE M., YOKOO M., MARSELLA S. and PYNADATH D. V., "Taming decentralized POMDPs: towards efficient policy computation for multiagent settings", *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, Acapulco, Mexico, pp. 705–711, 2003.

[NAI 04] NAIR R., ROTH M., YOKOO M. and TAMBE M., "Communication for improving policy computation in distributed POMDPs", *Proceedings of the 3rd International Joint Conference on Agents and Multiagent Systems (AAMAS'04)*, New York, pp. 1098–1105, 2004.

[NAI 05] NAIR R., PRADEEP V., MILIND T. and MAKOTO Y., "Networked distributed POMDPs: a synthesis of distributed constraint optimization and POMDPs", *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'05)*, Pittsburgh, PA, pp. 133–139, 2005.

[OLI 07a] OLIEHOEK F., KOOIJ J. and VLASSIS N., "A cross-entropy approach to solving Dec-POMDPs", *Proceedings of the 1st International Symposium on Intelligent and Distributed Computing*, Craisova, Romania, pp. 145–154, 2007.

[OLI 07b] OLIEHOEK F. and VLASSIS N., "Q-value functions for decentralized POMDPs", *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)*, Honolulu, Hawaii, pp. 833–840, 2007.

[PAP 87] PAPADIMITRIOU C. H. and TSITSIKLIS J. N., "The complexity of Markov decision processes", *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.

[PES 00] PESHKIN L., KIM K., MEULEAU N. and KAELBLING L., "Learning to cooperate via policy search", *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI'00)*, Morgan Kaufman, San Francisco, CA, pp. 489–496, 2000.

[PUT 94] PUTERMAN M. L., *Markov decision processes: Discrete stochastic dynamic programming*, John Wiley & Sons, New York, 1994.

[PYN 02] PYNADATH D. V. and TAMBE M., "The communicative multiagent team decision problem: analyzing teamwork theories and models", *Journal of Artificial Intelligence Research*, vol. 16, pp. 389–423, 2002.

[ROY 00] ROY N., PINEAU J. and THRUN S., "Spoken dialogue management using probabilistic reasoning", *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-2000)*, Hong Kong, pp. 93–100, 2000.

[SCH 02] SCHERRER B. and CHARPILLET F., "Cooperative co-learning: a model-based approach for solving multi-agent reinforcement problems", *Proceedings of the 14th International Conference on Tools with Artificial Intelligence (ICTAI'02)*, pp. 463–468, 2002.

[SEU 07] SEUKEN S. and ZILBERSTEIN S., "Improved memory-bounded dynamic programming for decentralized POMDPs", *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI'07)*, Vancouver, British Columbia, Canada, 2007.

[SEU 08] SEUKEN S. and ZILBERSTEIN S., "Formal models and algorithms for decentralized decision making under uncertainty", *Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 2, pp. 190–250, 2008.

[SHA 53] SHAPLEY L. S., "Stochastic games", *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, vol. 39, pp. 1095–1100, 1953.

[SHE 06] SHEN J., BECKER R. and LESSER V., "Agent interaction in distributed MDPs and its implications on complexity", *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, ACM Press, New York, pp. 529–536, 2006.

[SPA 08] SPAAN M. T. J., OLIEHOEK F. A. and VLASSIS N. A., "Multiagent planning under uncertainty with stochastic communication delays", *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, Sydney, Australia, pp. 338–345, 2008.

[SZE 05] SZER D., CHARPILLET F. and ZILBERSTEIN S., "MAA*: A heuristic search algorithm for solving decentralized POMDPs", *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI'05)*, pp. 576–583, 2005.

[SZE 06] SZER D. and CHARPILLET F., "Point-based dynamic programming for DEC-POMDPs", *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, Boston, MA, pp. 1233–1238, 2006.

[TAO 01] TAO N., BAXTER J. and WEAVER L., "A multi-agent, policy-gradient approach to network routing", *Proceedings of the 18th International Conference on Machine Learning (ICML'01)*, Morgan Kaufmann, San Francisco, CA, pp. 553–560, 2001.

[WAS 96] WASHINGTON R., "Incremental Markov-model planning", *Proceedings of the 8th International Conference on Tools with Artificial Intelligence (ICTAI'96)*, pp. 41–47, 1996.

[XUA 01] XUAN P., LESSER V. and ZILBERSTEIN S., "Communication decisions in multi-agent cooperation: model and experiments", *Proceedings of the 5th International Conference on Autonomous Agents (Agents'01)*, Montreal, Canada, pp. 616–623, 2001.