Auton Agent Multi-Agent Syst
DOI 10.1007/s10458-007-9026-5

Formal models and algorithms for decentralized decision making under uncertainty

Sven Seuken · Shlomo Zilberstein

Abstract Over the last 5 years, the AI community has shown considerable interest in decentralized control of multiple decision makers or "agents" under uncertainty. This problem arises in many application domains, such as multi-robot coordination, manufacturing, information gathering, and load balancing. Such problems must be treated as decentralized decision problems because each agent may have different partial information about the other agents and about the state of the world. It has been shown that these problems are significantly harder than their centralized counterparts, requiring new formal models and algorithms to be developed. Rapid progress in recent years has produced a number of different frameworks, complexity results, and planning algorithms. The objectives of this paper are to provide a comprehensive overview of these results, to compare and contrast the existing frameworks, and to provide a deeper understanding of their relationships with one another, their strengths, and their weaknesses. While we focus on cooperative systems, we do point out important connections with game-theoretic approaches. We analyze five different formal frameworks, three different optimal algorithms, as well as a series of approximation techniques. The paper provides interesting insights into the structure of decentralized problems, the expressiveness of the various models, and the relative advantages and limitations of the different solution techniques. A better understanding of these issues will facilitate further progress in the field and help resolve several open problems that we identify.

Keywords Decentralized decision making · Cooperative agents · Multi-agent planning

This work was done while S. Seuken was a graduate student in the Computer Science Department of the University of Massachusetts, Amherst.

S. Seuken (✉)
School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138, USA
e-mail: seuken@eecs.harvard.edu

S. Zilberstein
Department of Computer Science, University of Massachusetts, Amherst, MA 01003, USA
e-mail: shlomo@cs.umass.edu

1 Introduction

Decision-theoretic models for planning under uncertainty have been studied extensively in artificial intelligence and operations research since the 1950s. The *Markov decision process* (MDP), in particular, has emerged as a useful framework for centralized sequential decision making in fully observable stochastic environments ([38,41, chap. 17]). When an agent cannot fully observe the environment, it must base its decisions on partial information and the standard MDP framework is no longer sufficient. In the 1960s, Aström [3] introduced partially observable MDPs (POMDPs) to account for imperfect observations. In the 1990s, researchers in the AI community adopted the POMDP framework and developed numerous new exact and approximate solution techniques [21,23].

An even more general problem results when two or more agents have to coordinate their actions. If each agent has its own separate observation function, but the agents must work together to optimize a joint reward function, the problem that arises is called *decentralized control of a partially observable stochastic system*. This problem is particularly hard because each individual agent may have different partial information about the other agents and about the state of the world. Over the last 7 years, different formal models for this problem have been proposed, interesting complexity results have been established, and new solution techniques for this problem have been developed.

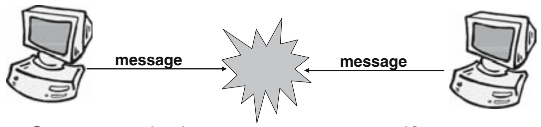
The decentralized partially observable MDP (DEC-POMDP) framework is one way to model this problem. In 2000, it was shown by Bernstein et al. [8] that finite-horizon DEC-POMDPs are NEXP-complete—even when just 2 agents are involved. This was an important turning point in multi-agent planning research. It proved that decentralized control of multiple agents is significantly harder than single agent control and provably intractable. Due to these complexity results, optimal algorithms have mostly theoretical significance and only little use in practice. Consequently, several new approaches to approximate the optimal solution have been developed over the last few years. Another fruitful research direction is to exploit the inherent structure of certain more practical problems, leading to lower complexity. Identifying useful classes of decentralized POMDPs that have lower complexity and developing special algorithms for these classes proved to be an effective way to address the poor scalability of general algorithms. However, although some approximate algorithms can solve significantly larger problems, their scalability remains a major research challenge.

1.1 Motivation

In the real world, decentralized control problems are ubiquitous. Many different domains where these problems arise have been studied by researchers in recent years. Examples include coordination of space exploration rovers [51], load balancing for decentralized queues [11], coordinated helicopter flights [39,49], multi-access broadcast channels [29], and sensor network management [27]. In all these problems, multiple decision makers jointly control a process, but cannot share all of their information in every time step. Thus, solution techniques that rely on centralized operation are not suitable. To illustrate this, we now describe three problems that have been widely used by researchers in this area.

1.1.1 Multi-access broadcast channel

The first example is an idealized model of a multi-access broadcast channel (adapted from [29]). Two agents are controlling a message channel on which only one message per time step can be sent, otherwise a collision occurs. The agents have the same goal of maximizing the



- **States**: who has a message to send?
- **Actions**: *send* or *not send*
- **Rewards** +1 for successful broadcast, 0 otherwise
- **Observations**: was there a collision? (noisy)

Fig. 1 Multi-access broadcast channel (courtesy of Daniel Bernstein)

global throughput of the channel. Every time step the agents have to decide whether to send a message or not. They receive a global reward of 1 when a message is sent and a reward of 0 if there was a collision or no message was sent at all. At the end of a time step, every agent observes information about its own message buffer, about a possible collision and about a possible successful message broadcast. The challenge of this problem is that the observations of possible collisions are noisy, and thus the agents can only build up potentially uncertain beliefs about the outcome of their actions. Figure 1 illustrates the general problem setup. Experimental results can be found in [5].

1.1.2 The multi-agent tiger problem

Probably the most widely used problem for single agent POMDPs is the tiger problem introduced by Kaelbling et al. [23]. A multi-agent version of the tiger problem was introduced by Nair et al. [26]. It involves two agents standing in front of two closed doors. Behind one of the doors there is a hungry tiger, and behind the other door there is valuable treasure. The agents do not know the position of either. By listening, rather than simply opening one of the doors, the agents can gain information about the position of the tiger. But listening has a cost and is not entirely accurate (i.e. it only reveals the correct information about the location of the tiger with a certain probability <1). Moreover, the agents cannot communicate their observations to each other. In each step, each agent can independently either listen or open one of the doors. If one of the agents opens the door with the treasure behind it, they both get the reward. If either agent opens the door with the tiger, a penalty is incurred. However, if they both open the tiger door at the same time, they receive less of a penalty. The agents have to come up with a joint policy for listening and finally opening a door. After a door is opened and the agents receive a reward/penalty, the problem starts over again.

1.1.3 Meeting under uncertainty

This more realistic example was first presented by Bernstein et al. [6]. It is a simplified version of the real-life problem of multi-robot planning [45]. Here, two agents have to meet as soon as possible on a 2D grid where obstacles are blocking some parts of the environment. The possible actions of the agents include moving North, South, East, West and staying at the same grid position. Every time step the agents make a noisy transition, that is, with some probability P_i, agent i arrives at the desired location and with probability $1 - P_i$ the agent remains at the same location. After making a transition, the agents can sense some information. This might simply be its own location on the grid or this might include some information about the terrain topology. In either case, an agent's partial information is insufficient to determine the global state of the system. Due to the uncertain transitions of the agents, the optimal solution is not easy to compute, as every agent's strategy can only depend

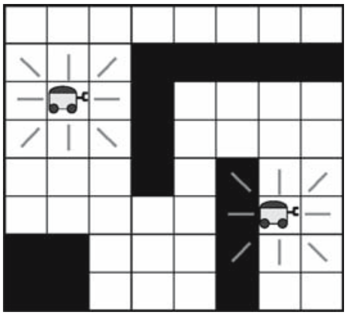


Fig. 2 Meeting under uncertainty on a grid (courtesy of Daniel Bernstein)

on some *belief* about the other agent's location. An optimal solution for this problem is a sequence of moves for each agent such that the expected time taken to meet is as low as possible. Figure 2 shows a sample problem involving an 8×8 grid. Experimental results can be found in [20].

1.2 Outline

We have described three examples of problems involving decentralized control of multiple agents. It is remarkable that prior to 2000, the computational complexity of these problems had not been established, and no optimal algorithms were available. In Sects. 2.1 and 2.2, four different proposed formal models for this class of problems are presented and compared with one another. Equivalence of all these models in terms of expressiveness and complexity is established in Sect. 2.3. In Sect. 2.4, another formal model for describing decentralized problems is presented—one orthogonal to the others. This framework differs from the ones presented earlier as regards both expressiveness and computational complexity. Sect. 2.5 goes on to consider interesting sub-classes of the general problem that lead to lower complexity classes. Some of these sub-problems are computationally tractable. Unfortunately, many interesting problems do not fall into one of the easy sub-classes, leading to the development of non-trivial optimal algorithms for the general class, which are described in Sect. 3. This is followed by a description of general approximation techniques in Sects. 4–6. For some approaches, specific limitations are identified and discussed in detail. In Sect. 7, all the presented algorithms are compared along with their advantages and drawbacks. Finally, in Sect. 8, conclusions are drawn and a short overview of open research questions is given.

2 Decision-theoretic models for multi-agent planning

To formalize the problem of decentralized control of multiple agents, a number of different formal models have recently been introduced. In all of these models, at each step, each agent takes an action, a state transition occurs, and each agent receives a local observation. Following this, the environment generates a global reward that depends on the set of actions taken by all the agents. Figure 3 gives a schematic view of a decentralized process with two agents. It is important to note that each agent receives an individual observation, but the reward generated by the environment is the same for all agents. Thus, we focus on *cooperative* systems in which each agent wants to maximize a joint global reward function. In contrast, non-cooperative multi-agent systems, such as partially observable stochastic games (POSGs), allow each agent to have its own private reward function. Solution techniques for

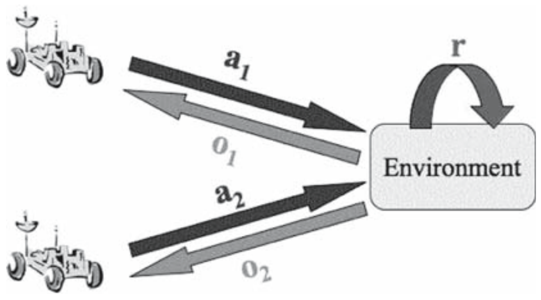


Fig. 3 Schematic view of a decentralized process with 2 agents, a global reward function and private observation functions (courtesy of Christopher Amato)

Table 1 Overview of formal models for decentralized control of multiple agents

	Implicit belief representation		Explicit belief representation
Implicit communication	DEC-POMDP (Sect. 2.1.1)	MTDP (Sect. 2.1.2)	I-POMDP (Sect. 2.4.1)
Explicit communication	DEC-POMDP-COM (Sect. 2.2.1)	MTDP-COM (Sect. 2.2.2)	

non-cooperative multi-agent systems are quite different from the algorithms described in this paper and often rely on techniques from algorithmic game theory [28] and computational mechanism design [34].

The five different formal models we will present in Sects. 2.1–2.4 can all be used to solve the general problem as introduced via the examples in Sect. 1.1. One important aspect that differentiates them is the treatment of communication. Some frameworks explicitly model the communication actions of the agents and others subsume them under the general action sets. Each approach has different advantages and disadvantages, depending on the focus of the analysis. A second aspect that differentiates the models is whether they use an implicit or explicit representation of agent beliefs. An overview of the models is given in Table 1.

We begin with two formal models without explicit communication, followed by two frameworks that explicitly model communication actions. We prove that all four models are equivalent in terms of expressiveness as well as computational complexity.

2.1 Models without explicit communication

2.1.1 The DEC-POMDP model

Definition 1 (*DEC-POMDP*) A decentralized partially observable Markov decision process (DEC-POMDP) is a tuple $\langle I, S, \{A_i\}, P, \{\Omega_i\}, O, R, T\rangle$ where

- I is a finite set of agents indexed $1, \ldots, n$.
- S is a finite set of states, with distinguished initial state s_0.
- A_i is a finite set of actions available to agent i and $\vec{A} = \otimes_{i\in I} A_i$ is the set of joint actions, where $\vec{a} = \langle a_1, \ldots, a_n\rangle$ denotes a joint action.
- $P : S \times \vec{A} \rightarrow \Delta S$ is a Markovian transition function. $P(s'|s, \vec{a})$ denotes the probability that after taking joint action $\vec{a}$ in state s a transition to state s' occurs.
- Ω_i is a finite set of observations available to agent i and $\vec{\Omega} = \otimes_{i\in I}\Omega_i$ is the set of joint observation, where $\vec{o} = \langle o_1, \ldots, o_n\rangle$ denotes a joint observation.

- $O : \vec{A} \times S \rightarrow \Delta\vec{\Omega}$ is an observation function. $O(\vec{o}|\vec{a}, s')$ denotes the probability of observing joint observation $\vec{o}$ given that joint action $\vec{a}$ was taken and led to state s'. Here $s' \in S$, $\vec{a} \in \vec{A}$, $\vec{o} \in \vec{\Omega}$.
- $R : \vec{A} \times S \rightarrow \Re$ is a reward function. $R(\vec{a}, s')$ denotes the reward obtained after joint action $\vec{a}$ was taken and a state transition to s' occurred.
- If the DEC-POMDP has a finite horizon, that horizon is represented by a positive integer T.

This framework was first proposed by Bernstein et al. [8]. In 2002, they introduced another version of this model [6] that allows for more general observation and reward functions that differentiate between different originating states. However, these more general functions can easily be simulated in the original model by growing the state space to the cross product of S with itself. Here, the earlier definition is used, because it simplifies showing the equivalence with the MTDP model (see Theorem 1).

Definition 2 (*Local policy for a DEC-POMDP*) A local policy for agent i, δ_i, is a mapping from local histories of observations $\overline{o}_i = o_{i_1} \ldots o_{i_t}$ over Ω_i to actions in A_i, $\delta_i : \Omega_i^* \rightarrow A_i$.

Definition 3 (*Joint policy for a DEC-POMDP*) A joint policy, $\delta = \langle \delta_1, \ldots, \delta_2 \rangle$, is a tuple of local policies, one for each agent.

Solving a DEC-POMDP means finding a joint policy that maximizes the expected total reward. This is formalized by the following *performance criteria*.

Definition 4 (*Solution value for a finite-horizon DEC-POMDP*) For a finite-horizon problem, the agents act for a fixed number of steps, which is called the horizon and denoted by T. The value of a joint policy δ for a finite-horizon DEC-POMDP with initial state s_0 is

$$V^{\delta}(s_0) = E\left[\sum_{t=0}^{T-1} R(\vec{a}_t, s_t)|s_0, \delta\right].$$

When the agents operate over an unbounded number of time-steps or the time horizon is so large that it can best be modeled as being infinite, we use the infinite-horizon discounted performance criterion. In this case, a discount factor, γ^t, is used to weigh the reward collected t time-steps into the future. For a detailed discussion of infinite-horizon models see Bernstein [5].

Definition 5 (*Solution value for an infinite-horizon DEC-POMDP*) The value of a joint policy δ for an infinite-horizon DEC-POMDP with initial state s_0 and discount factor $\gamma \in [0, 1)$ is

$$V^{\delta}(s_0) = E\left[\sum_{t=0}^{\infty} \gamma^t R(\vec{a}_t, s_t)|s_0, \delta\right].$$

All the models presented in this section can be used for finite-horizon or infinite-horizon problems. However, for the introduction of the following formal models we will use the finite-horizon version because this is the version needed to prove the main complexity result in Sect. 2.3.2. Note that, as for DEC-POMDPs, the infinite-horizon version of the other models only differ from their finite-horizon counterparts in that they do not have the parameter T but instead a discount factor γ. We will differentiate explicitly between the two versions when we present solution algorithms later because representation of policies and solution techniques in general can differ significantly for finite and infinite-horizon problems respectively.

2.1.2 The MTDP model

In 2002, Pynadath and Tambe [39] presented the MTDP framework, which is very similar to the DEC-POMDP framework. The model as presented below uses one important assumption:

Definition 6 (*Perfect recall*) An agent has *perfect recall* if it has access to all of its received information (this includes all local observations as well as messages from other agents).

Definition 7 (*MTDP*) A multiagent team decision problem (MTDP) is a tuple $\langle \alpha, S, \mathbf{A}_\alpha, P, \mathbf{\Omega}_\alpha, \boldsymbol{O}_\alpha, \boldsymbol{B}_\alpha, R, T \rangle$, where

- α is a finite set of agents, numbered $1, \dots, n$.
- $S = \Xi_1 \times \cdots \times \Xi_m$ is a set of world states, expressed as a factored representation (a cross product of separate features).
- $\{A_i\}_{i \in \alpha}$ is a set of actions for each agent, implicitly defining a set of combined actions, $\mathbf{A}_\alpha \equiv \Pi_{i \in \alpha} A_i$.
- $P : S \times \mathbf{A}_\alpha \times S \to [0, 1]$ is a probabilistic distribution over successor states, given the initial state and a joint action, i.e. $P(s, \mathbf{a}, s') = Pr(S^{t+1} = s' | S^t = s, \mathbf{A}_\alpha^t = \mathbf{a})$.
- $\{\Omega\}_{i \in \alpha}$ is a set of observations that each agent i can experience, implicitly defining a set of combined observations, $\mathbf{\Omega}_\alpha \equiv \Pi_{i \in \alpha} \Omega_i$.
- $\mathbf{O}_\alpha$ is a joint observation function modeling the probability of receiving a joint observation $\boldsymbol{\omega}$ after joint action $\mathbf{a}$ was taken and a state transition to s' occurred, i.e. $\mathbf{O}_\alpha(s', \boldsymbol{a}, \boldsymbol{\omega}) = Pr(\mathbf{\Omega}_\alpha^{t+1} = \boldsymbol{\omega} | S^{t+1} = s', \mathbf{A}_\alpha^t = \mathbf{a})$.
- $\mathbf{B}_\alpha$ is the set of possible combined belief states. Each agent $i \in \alpha$ forms a belief state $b_i^t \in B_i$, based on its observations seen through time t, where B_i circumscribes the set of possible belief states for agent i. This mapping of observations to belief states is performed by a *state estimator function* under the assumption of perfect recall. The resulting combined belief state is denoted $\mathbf{B}_\alpha \equiv \Pi_{i \in \alpha} B_i$. The corresponding random variable $\mathbf{b}_\alpha^t$ represents the agents' combined belief state at time t.
- $R : S \times \mathbf{A}_\alpha \to \Re$ is a reward function representing a team's joint preferences.
- If the MTDP has a finite horizon, that horizon is represented by a positive integer T.

Note that due to the assumption of perfect recall, the definition of the state estimator function is not really necessary as in that case it simply creates a list of observations. However, if this assumption is relaxed or a way of mapping observations to a *compact* belief state is found, this additional element might become useful. So far, no state estimator function without perfect recall has been proposed, and no compact representation of belief states for multi-agent settings has been introduced.

Definition 8 (*Domain-level policy for an MTDP*) The set of possible domain-level policies in an MTDP is defined as the set of all possible mappings from belief states to actions, $\pi_{iA} : B_i \to A_i$.

Definition 9 (*Joint domain-level policy for an MTDP*) A joint domain-level policy for an MTDP, $\boldsymbol{\pi}_{\alpha A} = \langle \pi_{1A}, \dots, \pi_{nA} \rangle$, is a tuple of domain-level policies, one for each agent.

Solving an MTDP means finding a joint policy that maximizes the expected global reward.

2.1.3 Computational complexity and equivalence results

Before the equivalence of the DEC-POMDP model and the MTDP model can be established, the precise notion of equivalence must be defined. This requires a short excursion to complexity theory. All models presented in this paper so far describe decentralized *optimization* *problems*. Thus, finding a solution for those problems means finding a policy that maximizes the expected value. However, most complexity classes are defined in terms of *decision problems*. A decision problem is a formal question with a "yes/no" answer, or equivalently it is the *set of problem instances* where the answer to the question is "yes". A *complexity class* is then defined by the set of all decision problems that it contains. Alternatively, a complexity class can also be defined by a complexity measure (e.g. time, space) and a corresponding resource bound (e.g. polynomial, exponential). P for example denotes the complexity class that contains all decision problems that can be solved in polynomial time.

All models/problems presented in this paper can be converted to "yes/no" decision problems to analyze their complexity. This is done by adding a parameter K to denote a threshold value. For example, the decision problem for a DEC-POMDP then becomes: given DEC-POMDP $\langle I, S, \{A_i\}, P, \{\Omega_i\}, O, R, T, K\rangle$, is there a joint policy for which the value of the initial state s_0 exceeds K? Obviously, actually finding the optimal policy for a DEC-POMDP problem can be no easier than the threshold problem. Note that for all equivalence proofs in this paper we are reducing decision problems to each other.

In Sect. 2.3.2 we will show that the DEC-POMDP decision problem is NEXP-complete, i.e. it can be solved in *nondeterministic exponential time* and every other problem in NEXP can be efficiently reduced to this problem, where here "efficiently" means in polynomial time. More formally this means: DEC-POMDP $\in$ NEXP and $\forall C \in$ NEXP: $C \leq_p$ DEC-POMDP. For more details on complexity theory see Papadimitriou [31].

Definition 10 (*Equivalence of models/problems*) Two models are called *equivalent* if their corresponding decision problems are complete for the same complexity class.

To show that two problems are equivalent, a completeness proof such as in Sect. 2.3.2 could be performed for both problems. Alternatively, formal equivalence can be proved by showing that the two problems are reducible to each other. To reduce decision problem A to decision problem B, a function f has to be found that formally defines a mapping of problem instances $x \in A$ to problem instances $f(x) \in B$, such that the answer to x is "yes" if and only if the answer to f(x) is "yes." This mapping-function f has to be computable in polynomial time. The reduction of problem A to problem B in polynomial time is denoted $A \leq_p B$.

Note that this notion of equivalence implies that if two models are equivalent they are equivalent in terms of expressiveness as well as in terms of complexity. That is, the frameworks can describe the same problem instances and are complete for the same complexity class. To know the complexity class for which they are complete, at least one completeness proof for one of the models has to be obtained. Any framework for which equivalence with this model has been shown is then complete for the same complexity class.

Theorem 1 *The DEC-POMDP model and the MTDP model are equivalent under the perfect recall assumption.*

Proof We need to show: 1. DEC-POMDP $\leq_p$ MTDP and 2. MTDP $\leq_p$ DEC-POMDP.

1. DEC-POMDP $\leq_p$ MTDP:
   A DEC-POMDP decision problem is a tuple $\langle I, S, \{A_i\}, P, \{\Omega_i\}, O, R, T, K\rangle$ and an MTDP decision problem is a tuple $\langle \alpha, S', \mathbf{A}_\alpha, P', \boldsymbol{\Omega}_\alpha, \boldsymbol{O}_\alpha, \boldsymbol{B}_\alpha, R', T', K'\rangle$. There is an obvious mapping between:
   - the finite sets of agents, $I = \alpha$.
   - the finite set of world states, $S = S'$.
   - the finite set of joint actions for each agent, $\vec{A} = \mathbf{A}_\alpha$.

- the probability table for state transitions, $P(s'|s, \vec{a}) = P'(s, \mathbf{a}, s')$.
- the finite set of joint observations, $\vec{\Omega} = \mathbf{\Omega}_\alpha$.
- the observation function, $O(\vec{o}|\vec{a}, s') = \mathbf{O}_\alpha(s, \boldsymbol{a}, \boldsymbol{\omega})$.
- the reward function, $R(\vec{a}, s') = R'(\mathbf{a}, s)$.
- the finite horizon parameter, $T = T'$.
- the threshold parameter, $K = K'$.

The possible local histories of observations available to the agents in a DEC-POMDP are mapped to the possible belief states b_i^t of the MTDP, i.e. b_i^t is simply the sequence of observations of agent i until time t. Finding a policy for an MTDP means finding a mapping from belief states to actions. Thus, a policy for the resulting MTDP that achieves value K also constitutes a policy for the original DEC-POMDP with value K, where the final policies are mappings from local histories of observations to actions.

2. MTDP $\leq_p$ DEC-POMDP:
For α, S', $\mathbf{A}_\alpha$, P', $\mathbf{\Omega}_\alpha$, $\boldsymbol{O}_\alpha$, R', T', K', the same mapping as in part 1 is used. The only remaining element is the set of possible combined belief states, $\boldsymbol{B}_\alpha$. In an MTDP, each agent forms its belief state, $b_i^t \in B_i$, based on its observations seen through time t. Due to the perfect recall assumption in the MTDP model, the agents recall all of their observations. Thus, their belief states represent their entire histories as sequences of observations. Obviously, with this restriction, the state estimator function and the belief state space of the MTDP do not add anything beyond the DEC-POMDP model. Thus, the history of observations can simply be extracted from the belief state and then the resulting DEC-POMDP can be solved, i.e. a policy that is a mapping from histories of observations to actions can be found. Thus, a solution with value K for the DEC-POMDP exists if and only if a solution with value K for the original MTDP exists.

Therefore, the DEC-POMDP model and the MTDP model are equivalent. □

It becomes apparent that the only syntactical difference between MTDPs and DEC-POMDPs is the additional state estimator function and the resulting belief state of the MTDP model. But, with the assumption that the agents have perfect recall of all of their observations, the resulting belief state is nothing but a sequence of observations and therefore exists implicitly in a DEC-POMDP, too. We prefer the DEC-POMDP notation because it is somewhat cleaner and more compact. Therefore, from now on we use the DEC-POMDP as the "standard" model for decentralized decision making.

2.2 Models with explicit communication

Both models presented in Sect. 2.1 have been extended with explicit communication actions. In the resulting two models, the interaction among the agents is a process in which agents perform an action, then observe their environment and send a message that is instantaneously received by the other agents (no delays in the system). Both models allow for a general syntax and semantics for communication messages. Obviously, the agents need to have conventions about how to interpret these messages and how to combine this information with their own local information. One example of a possible communication language is $\Sigma_i = \Omega_i$, where the agents simply communicate their observations.

This distinction between the two different types of actions might seem unnecessary for practical applications but it is useful for analytical examinations. It provides us with a better way to analyze the effects of different types of communication in a multi-agent setting (cf. [19]).

2.2.1 The DEC-POMDP-COM model

The following model is equivalent to the one described in Goldman and Zilberstein [18]. The formal definition of some parts have been changed to make the DEC-POMDP-COM model presented here a straightforward extension of the previously described DEC-POMDP model.

Definition 11 (*DEC-POMDP-COM*) A decentralized partially observable Markov decision process with communication (DEC-POMDP-COM) is a tuple $\langle I, S, \{A_i\}, P, \{\Omega_i\}, O, \Sigma, C_\Sigma, R_A, R, T\rangle$ where:

- $I,\ S,\ \{A_i\},\ P,\ \{\Omega_i\},\ O$, and T are defined as in the DEC-POMDP.
- Σ is the alphabet of communication messages. $\sigma_i \in \Sigma$ is an atomic message sent by agent i, and $\vec{\sigma} = \langle \sigma_1, \ldots, \sigma_n\rangle$ is a joint message, i.e. a tuple of all messages sent by the agents in one time step. A special message belonging to Σ is the null message, ε_σ, which is sent by an agent that does not want to transmit anything to the others.
- $C_\Sigma : \Sigma \rightarrow \Re$ is the message cost function. $C_\Sigma(\sigma_i)$ denotes the cost for transmitting atomic message σ_i. We set $C_\Sigma(\varepsilon_\sigma) = 0$, i.e. agents incur no cost for sending a null message.
- $R_A : \vec{A} \times S \rightarrow \Re$ is the action reward function identical to the reward function in a DEC-POMDP, i.e. $R_A(\vec{a}, s')$ denotes the reward obtained after joint action $\vec{a}$ was taken and a state transition to s' occurred.
- $R : \vec{A} \times S \times \vec{\Sigma} \rightarrow \Re$ denotes the total reward function, defined via R_A and C_Σ: $R(\vec{a}, s', \vec{\sigma}) = R_A(\vec{a}, s') - \sum_{i\in I} C_\Sigma(\sigma_i)$.

Note that if R is represented explicitly, the representation size of R is at least $\Omega(|\Sigma|^n)$. If R is only represented implicitly via R_A and C_Σ, we will assume that $|\Sigma|$ is polynomially bounded by either $|\Omega_i|$ or $|A_i|$, which implies that the observation function O will be of size at least $\Omega(|\Sigma|^n)$.

Definition 12 (*Local policy for action for a DEC-POMDP-COM*) A local policy for action for agent i, δ_i^A, is a mapping from local histories of observations $\overline{o}_i$ over Ω_i and histories of messages $\overline{\sigma}_j$ received ($j \neq i$) to control actions in A_i, $\delta_i^A : \Omega_i^* \times \vec{\Sigma}^* \rightarrow A_i$.

Definition 13 (*Local Policy for communication for a DEC-POMDP-COM*) A local policy for communication for agent i, δ_i^Σ, is a mapping from local histories of observations $\overline{o}_i$ over Ω_i, and histories of messages $\overline{\sigma}_j$ received ($j \neq i$) to communication actions in Σ, $\delta_i^\Sigma : \Omega_i^* \times \vec{\Sigma}^* \rightarrow \Sigma$.

Definition 14 (*Joint policy for a DEC-POMDP-COM*) A joint policy $\delta = \langle \delta_1, \ldots, \delta_n\rangle$ is a tuple of local policies, one for each agent, where each δ_i is composed of the communication and action policies for agent i.

Solving a DEC-POMDP-COM means finding a joint policy that maximizes the expected total reward, over either an infinite or a finite horizon. See Goldman and Zilberstein [18] for further discussion.

Theorem 2 *The DEC-POMDP model and the DEC-POMDP-COM model are equivalent.*

Proof We need to show: 1. DEC-POMDP $\leq_p$ DEC-POMDP-COM and 2. DEC-POMDP-COM $\leq_p$ DEC-POMDP.

1. DEC-POMDP $\leq_p$ DEC-POMDP-COM:
For the first reduction, a DEC-POMDP can simply be mapped to a DEC-POMDP-COM with an empty set of communication messages and no cost function for the communication messages. Thus, $\Sigma = \emptyset$ and C_Σ is the empty function. Obviously a solution with value K for the resulting DEC-POMDP-COM is also a solution with value K for the original DEC-POMDP.
2. DEC-POMDP-COM $\leq_p$ DEC-POMDP:
The DEC-POMDP-COM decision problem is a tuple $\langle I, S, \{A_i\}, P, \{\Omega_i\}, O, \Sigma, C_\Sigma, R_A, R, T, K\rangle$ and a DEC-POMDP decision problem is a tuple $\langle I', S', \{A_i\}', P', \{\Omega_i\}', O', R', T', K'\rangle$. The only difference between these models is the explicit modeling of the communication actions in the DEC-POMDP-COM model, but those can be simulated in the DEC-POMDP in the following way:

   - The size of the state set S' has to be doubled to simulate the interleaving of control actions and communication actions. Thus, $S' = S \times \{0, 1\}$. Here, $\langle s, 0\rangle$ represents a system state where only control actions are possible, denoted a *control state*. And $\langle s, 1\rangle$ represents a system state where only communication actions are possible, denoted a *communication state*.
   - The length of the horizon is doubled, i.e. $T' = 2 \cdot T$, to account for the fact that for every time step in the original DEC-POMDP-COM we now have two time steps, one for the control action and one for the communication action.
   - Each communication message is added to the action sets: $\{A_i\}' = \{A_i\} \cup \Sigma$. To differentiate control actions from communication actions, joint control actions are denoted $\vec{a}$ and joint communication actions are denoted $\vec{\sigma}$.
   - For each possible joint communication message, one new joint observation is added to the joint observation set: $\vec{\Omega}' = \vec{\Omega} \cup \Sigma^n$. Thus, the observation function has to be defined for consistent joint messages/observations only, i.e. where every agent has information about the same n messages (1 sent and $n-1$ received). Note that we can do this mapping only because message transmission is deterministic. Alternatively, adding n-tuples of single-agent observations consisting of $n-1$ received messages would contain redundant information and, more importantly, could not be done in polynomial time. Although this simpler mapping is still exponential in n, it can be done in time polynomial in the input size, because according to the definition of a DEC-POMDP-COM the representations of R or O are of size at least $\Omega(|\Sigma|^n)$. In the following, we let o denote a normal observation and $\vec{\sigma}$ denote an observation representing n communication messages from all agents.
   - The transition function P' is constructed in such a way that if the agents are in a control state, every joint control action leads to a communication state. Similarly, if the agents are in a communication state, every joint communication action leads to a control state. The transition probabilities for a control action taken in a control state are implicitly defined by the original transition function. A communication action taken in a communication state deterministically leads to a successor state where only the second component of the state changes from 1 to 0. More formally:

$$\begin{aligned} \forall s, s', \vec{a}: &\quad P'(\langle s', 1\rangle | \langle s, 0\rangle, \vec{a}) = P(s'|s, \vec{a}) \\ \forall s, \vec{\sigma}_{-i}: &\quad P'(\langle s, 0\rangle | \langle s, 1\rangle, \vec{\sigma_{-i}}) = 1 \end{aligned}$$

   The transition function must be 0 for all impossible state transitions. This includes a control action taken in a control state leading to another control state, a communication action taken in a communication state but leading to a successor state where the first component of the state also changes, and a communication action taken in a communication state leading to a communication state again. More formally:

$$\begin{aligned} \forall s, s', \vec{a}: &\quad P'(\langle s', 0\rangle | \langle s, 0\rangle, \vec{a}) = 0 \\ \forall s' \neq s, \vec{\sigma}_{-i}: &\quad P'(\langle s', 0\rangle | \langle s, 1\rangle, \vec{\sigma}_{-i}) = 0 \\ \forall s, s', \vec{\sigma}_{-i}: &\quad P'(\langle s', 1\rangle | \langle s, 1\rangle, \vec{\sigma}_{-i}) = 0 \end{aligned}$$

Furthermore, the transition function has to be defined for the cases of actions not possible in the original model, i.e. when a communication action is taken in a control state or when a control action is taken in a communication state. In these cases the state must simply remain the same. More formally:

$$\begin{aligned} \forall s = s', \vec{\sigma}_{-i}: &\quad P'(\langle s', 0\rangle | \langle s, 0\rangle, \vec{\sigma}_{-i}) = 1 \\ \forall s \neq s', \vec{\sigma}_{-i}: &\quad P'(\langle s', 0\rangle | \langle s, 0\rangle, \vec{\sigma}_{-i}) = 0 \end{aligned}$$

and

$$\begin{aligned} \forall s, s', \vec{\sigma}_{-i}: &\quad P'(\langle s', 1\rangle | \langle s, 0\rangle, \vec{\sigma}_{-i}) = 0 \\ \forall s = s', \vec{a}: &\quad P'(\langle s', 1\rangle | \langle s, 1\rangle, \vec{a}) = 1 \end{aligned}$$

and

$$\begin{aligned} \forall s \neq s', \vec{a}: &\quad P'(\langle s', 1\rangle | \langle s, 1\rangle, \vec{a}) = 0 \\ \forall s, s', \vec{a}: &\quad P'(\langle s', 0\rangle | \langle s, 1\rangle, \vec{a}) = 0 \end{aligned}$$

Note that with this definition of the transition function P', the agents could theoretically now make time go by without really taking any actions, e.g. by taking a communication action in a control state. We will prevent this by adjusting the reward function accordingly.

- The observation function O' has to be defined for control actions and for communication actions. The observation probabilities for a control action taken in a control state are implicitly defined by the original observation function. As explained above, for communication states we use the simplification that each joint communication action corresponds to only one joint observation consisting of n messages (which is possible due to the deterministic message transmission). Thus, the observation function has to be defined such that after taking a joint communication action $\vec{\sigma}$ the probability of observing the corresponding joint communication message is 1 and the probability of observing any other observation is 0. Thus:

$$\forall \vec{a}, s', \vec{o}: \quad O'(\vec{o}|\vec{a}, \langle s', 1\rangle) = O(\vec{o}|\vec{a}, s')$$

and

$$\forall \vec{o}, \vec{\sigma}, s': \quad O'(\vec{o}|\vec{\sigma}, \langle s', 0\rangle) = \begin{cases} 1 & \text{if} \quad \vec{o} = \vec{\sigma} \\ 0 & \text{else} \end{cases}$$

For completeness, the observation function must be defined for all impossible action-state combinations. One simple way to do this is to set the observation probability equal to 1 for observing $\vec{\varepsilon_{\sigma}}$, the joint message only consisting of null messages, and to set all other observation probabilities equal to 0. More formally:

$$\forall \vec{o}, \vec{\sigma}, s' : \quad O'(\vec{o}|\vec{\sigma}, \langle s', 1\rangle) = \begin{cases} 1 & \text{if} \quad \vec{o} = \vec{\varepsilon_\sigma} \\ 0 & \text{else} \end{cases}$$

and

$$\forall \vec{o}, \vec{a}, s' : \quad O'(\vec{o}|\vec{a}, \langle s', 0\rangle) = \begin{cases} 1 & \text{if} \quad \vec{o} = \vec{\varepsilon_\sigma} \\ 0 & \text{else} \end{cases}$$

Again, although we are adding $O(|\Sigma|^n)$ joint observations to the observation function, this mapping can be done in time polynomial in the input size, because according to the definition of a DEC-POMDP-COM the representations of R or O are of size at least $\Omega(|\Sigma|^n)$.

- The reward function R' has to be changed to account for the communication costs as defined by C_Σ. If $\langle s', 0\rangle$ is the control state reached after joint communication action $\vec{\sigma} = \langle \sigma_1, \ldots \sigma_n\rangle$ was taken, then the reward must represent the sum of the communication costs $C_\Sigma(\sigma_i)$. For control actions taken in a control state, the reward is defined implicitly via the original reward function. More formally:

$$\forall \vec{\sigma}, s' : \quad R'(\vec{\sigma}, \langle s', 0\rangle) = \sum_i^n C_\Sigma(\sigma_i)$$

and

$$\forall \vec{a}, s' : \quad R'(\vec{a}, \langle s', 1\rangle) = R_A(\vec{a}, s')$$

Again, for completeness the reward function has to be defined for impossible action-state combinations. For those cases we have made sure that the agents simply stay in the same state and observe the null message. Unfortunately, if the original DEC-POMDP-COM was constructed such that for certain (or all) state transitions the agents received a negative reward, finding an optimal plan in the new DEC-POMDP would result in an undesired solution. The optimal plan might include steps where an agent takes a "null action" by choosing an impossible state-action combination. To prevent this from happening, we set the reward to negative infinity whenever a control action leads to a control state or a communication action leads to a communication state. Thus:

$$\forall \vec{a}, s' : \quad R'(\vec{a}, \langle s', 0\rangle) = -\infty$$

and

$$\forall \vec{\sigma}, s' : \quad R'(\vec{\sigma}, \langle s', 1\rangle) = -\infty$$

In the resulting DEC-POMDP, the agents start off taking a joint control action. Then a state transition to a communication state occurs. After taking a joint communication action, a state transition to a control state occurs. This corresponds exactly to the course of action in the original DEC-POMDP-COM where the agents also had designated action and communication phases. Receiving messages is simulated by observations that correspond to the joint messages. Therefore, the information available to an agent in the DEC-POMDP is the same as in the original DEC-POMDP-COM. By constructing P', O' and R' appropriately, we ensure that a policy with value K can be found for the resulting DEC-POMDP if and only if a policy with value K can be found for the original DEC-POMDP-COM. This completes the reduction of a DEC-POMDP-COM to a DEC-POMDP.

Thus, the DEC-POMDP and the DEC-POMDP-COM models are equivalent. □

2.2.2 The COM-MTDP model

Definition 15 (*COM-MTDP*)[1] A communicative multiagent team decision problem (COM-MTDP) is a tuple $\langle \alpha, S, \mathbf{A}_\alpha, P, \boldsymbol{\Omega}_\alpha, \boldsymbol{O}_\alpha, \boldsymbol{\Sigma}_\alpha, \boldsymbol{B}_\alpha, R\rangle$, where

- $\alpha,\ S,\ \mathbf{A}_\alpha,\ P,\ \boldsymbol{\Omega}_\alpha$, and $\boldsymbol{O}_\alpha$ remain defined as in an MTDP.
- $\boldsymbol{\Sigma}_\alpha$ is a set of combined communication messages, defined by $\boldsymbol{\Sigma}_\alpha \equiv \Pi_{i\in\alpha}\Sigma_i$, where $\{\Sigma_i\}_{i\in\alpha}$ is a set of possible messages for each agent i.
- $\mathbf{B}_\alpha$ is an extended set of possible combined belief states. Each agent $i \in \alpha$ forms belief state $b_i^t \in B_i$, based on its observations and on the messages received from the other agents. This mapping of observations and messages to belief states is performed by the state estimator function under the assumption of perfect recall.
- R is an extended reward function, now also representing the cost of communicative acts. It is defined by $R : S \times \mathbf{A}_\alpha \times \boldsymbol{\Sigma}_\alpha \to \Re$.

As with MTDPs, in COM-MTDPs, the state estimator function does not add any additional functionality due to the assumption of perfect recall. However, if this assumption is relaxed, a state estimator has the potential to model any noise or temporal delays that might occur in the communication channel. Furthermore, it could possibly perform some preprocessing of the observations and messages. So far, no such state estimators have been proposed.

Definition 16 (*Communication policy for a COM-MTDP*) A communication policy for a COM-MTDP is a mapping from the extended belief state space to communication messages, i.e. $\pi_{i\Sigma} : B_i \to \Sigma_i$.

Definition 17 (*Joint communication policy for a COM-MTDP*) A joint communication policy for an MTDP-COM, $\boldsymbol{\pi}_{\alpha\Sigma} = \langle \pi_{1\Sigma}, \ldots, \pi_{n\Sigma}\rangle$, is a tuple of communication policies, one for each agent.

Definition 18 (*Joint policy for a COM-MTDP*) A joint policy for a COM-MTDP is a pair, consisting of a joint domain-level policy and a joint communication policy, $\langle \boldsymbol{\pi}_{\alpha\Sigma}, \boldsymbol{\pi}_{\alpha A}\rangle$.

Solving a COM-MTDP means finding a joint policy that maximizes the expected total reward over either an infinite or a finite horizon.

2.3 Consolidation of 4 different formal models

2.3.1 Equivalence results

Theorem 3 *The DEC-POMDP-COM model and the COM-MTDP model are equivalent under the perfect recall assumption.*

Proof We need to show that: 1. DEC-POMDP-COM $\leq_p$ COM-MTDP and 2. COM-MTDP $\leq_p$ DEC-POMDP-COM.

[1] Adapted from Pynadath and Tambe [39].

As has been shown in Theorem 1, the DEC-POMDP model and the MTDP model are equivalent. The extensions of these models introduce/change Σ, C_Σ, R_A and R for the DEC-POMDP-COM model and $\boldsymbol{\Sigma}_\alpha$, $\boldsymbol{B}_\alpha$, and R' for the COM-MTDP model. For the parts of the models that did not change, the same mapping as presented in the earlier proof is used.

1. DEC-POMDP-COM $\leq_p$ COM-MTDP:
   Again, there are obvious mappings between:

   - The set of communication messages, $\Sigma = \boldsymbol{\Sigma}_\alpha$.
   - The extended reward function, $R(\vec{a}, s', \vec{\sigma}) = R'(\mathbf{a}, s', \boldsymbol{\sigma})$.

   The possible local histories of observations and communication messages received that are available to the agent in a DEC-POMDP are mapped to the extended set of possible combined belief states $\boldsymbol{B}_\alpha$ of the COM-MTDP, i.e. b_i^t is simply the sequence of observations and communication messages received by agent i until time t. Finding a policy for a COM-MTDP means finding a mapping from the extended set of combined belief states to actions (domain level actions and communication actions). Thus, a policy with value K for the resulting COM-MTDP is at the same time a policy with value K for the original DEC-POMDP-COM, where the final policies are mappings from local histories of observations and communication messages to actions (control actions and communication actions).
2. COM-MTDP $\leq_p$ DEC-POMDP-COM:
   For Σ and R, the same mappings as in part 1 are used. The only remaining element is the extended set of possible combined belief states, $\boldsymbol{B}_\alpha$, which again exists implicitly in a DEC-POMDP-COM, too. In a COM-MTDP, each agent forms its belief state, $b_i^t \in B_i$, based on its observations and the messages it has received up to time t. The *extended state estimator function* now forms the belief state with the following three operations:

   (a) Initially, the belief state is an empty history.
   (b) Every new observation agent i receives is appended to its belief state.
   (c) Every new message agent i receives is also appended to its belief state.

   Again, the COM-MTDP model assumes that the agents have perfect recall, i.e. they recall all of their observations and messages. Thus, the extended state estimator function builds up a complete history of observations and messages as in a DEC-POMDP-COM. The history of observations and communication messages can therefore simply be extracted from the belief state of the COM-MTDP and the resulting DEC-POMDP-COM can then be solved. This means finding a policy that is a mapping from histories of observations and communication messages to actions. Thus, a solution to the DEC-POMDP-COM with value K is also a solution to the original COM-MTDP with value K.

Therefore, the DEC-POMDP-COM and COM-MTDP models are equivalent. □

The following corollary now immediately follows from Theorems 1, 2 and 3.

Corollary 1 *All of the aforementioned models (DEC-POMDP, DEC-POMDP-COM, MTDP, and COM-MTDP) are equivalent.*

As explained in Sect. 2.1, we consider the DEC-POMDP model to be somewhat cleaner and more compact than the MTDP model. Obviously, the same holds true for DEC-POMDP-COMs versus COM-MTDPs. Whether one uses the model with or without explicit communication messages depends on the specific purpose. The models are equivalent, but using one of the models might be advantageous for a particular analysis. For example, when analyzing the effects of different types of communication, the DEC-POMDP-COM model is more suitable than the DEC-POMDP model (see for example [19]). If communication actions are taken to be no different than any other action, the DEC-POMDP model is more compact and thus more attractive.

2.3.2 Complexity results

The first complexity results for Markov decision processes go back to Papadimitriou and Tsitsiklis [33], where they showed that the MDP problem is P-complete and that the POMDP problem is PSPACE-complete for the finite-horizon case. For a long time, no tight complexity results for decentralized processes were known. In 1986, Papadimitriou and Tsitsiklis [32] proved that decentralized control of MDPs must be at least NP-hard; however, a complete complexity analysis also giving tight upper bounds was still missing.

In 2000, Bernstein et al. [8] were the first to prove that DEC-POMDPs with a finite horizon are NEXP-complete. This was a breakthrough in terms of understanding the complexity of decentralized control of multiple agents. In particular, due to the fact that it has been proven that P and EXP are distinct, this shows that optimal decentralized control of multiple agents is infeasible in practice. Moreover, it is strongly believed by most complexity theorists that EXP $\neq$ NEXP and that solving NEXP-complete problems needs double exponential time in the worst case. Note that this has *not* been proven formally, but for the remainder of this paper it is assumed to be true.

Below, a sketch of the original proof is presented. Obviously, due to the equivalence results from Corollary 1, all complexity results obtained for DEC-POMDPs hold also for DEC-POMDP-COMs, MTDPs and COM-MTDPs. The complexity proof involves two agents because this is sufficient to show NEXP-completeness. Let DEC-POMDP$_n$ denote a DEC-POMDP with n agents. In the finite-horizon version of the DEC-POMDP model, we assume that $T \leq |S|$. The theorems and proofs in this section are based on Bernstein et al. [6].

Theorem 4 *For any $n \geq 2$, a finite-horizon DEC-POMDP$_n$ $\in$ NEXP.*

Proof The following process shows that a non-deterministic Turing machine can solve any instance of a DEC-POMDP$_n$ in at most exponential time.

1. Guess a joint policy and write it down in exponential time. This is possible, because a joint policy consists of n mappings from observation histories to actions. Since $T \leq |S|$, the number of possible histories is exponentially bounded by the problem description.
2. The DEC-POMDP together with the guessed joint policy can be viewed as an exponentially bigger POMDP using n-tuples of observations and actions.
3. In exponential time, convert each of the exponentially many observation sequences into a belief state.
4. In exponential time, compute transition probabilities and expected rewards for an exponentially bigger belief state MDP.
5. This MDP can be solved in polynomial time, which is exponential in the original problem description.

Thus, there is an accepting computation path in the non-deterministic machine if and only if there is a joint policy that can achieve reward K. □

To present the full complexity result, we need to define a few additional properties. Note that some of these properties have different names in the DEC-POMDP and MTDP frameworks. We provide both names below, but only the first one listed in each definition is used in this paper.

Definition 19 (*Joint full observability* $\equiv$ *collective observability*) A DEC-POMDP is jointly fully observable if the n-tuple of observations made by all the agents uniquely determines the current global state. That is, if $O(\vec{o}|\vec{a}, s') > 0$ then $Pr(s'|\vec{o}) = 1$.

Definition 20 (*DEC-MDP*) A decentralized Markov decision process (DEC-MDP) is a DEC-POMDP with joint full observability.

Note that in a DEC-MDP, each agent alone still only has partial observability and does not have full information about the global state.

Theorem 5 *For any $n \geq 2$, a finite-horizon DEC-MDP$_n$ is NEXP-hard.*

Proof sketch The detailed proof of this lower bound is quite involved and can be found in Bernstein et al. [6]. We include a sketch of the proof below. The proof is based on a reduction of a known NEXP-complete problem called TILING to DEC-MDP$_n$. A TILING problem instance includes a board of size $n \times n$ (the size n is represented compactly in binary), a set of tile types $L = \{\text{tile-0}, \ldots, \text{tile-k}\}$, and a set of binary horizontal and vertical compatibility relations $H, V \subseteq L \times L$. A *tiling* is a mapping $f : \{0, \ldots, n-1\} \times \{0, \ldots, n-1\} \rightarrow L$. A tiling f is *consistent* if and only if (a) $f(0, 0)$ =tile-0 and (b) for all x, y, $\langle f(x, y), f(x+1, y)\rangle \in H$, and $\langle f(x, y), f(x, y+1)\rangle \in V$. The decision problem is to determine, given L, H, V, and n, whether a consistent tiling exists. Figure 4 shows an example of a tiling instance and a corresponding consistent tiling.

For the following reduction, a fixed but arbitrarily chosen instance of the tiling problem is assumed, i.e. L, H, V, and n are fixed. A corresponding DEC-MDP is constructed, with the requirement that it is solvable (i.e. there is a joint policy for which the value of the initial state s_0 exceeds K) if and only if the selected tiling instance is solvable. The basic idea is to create a DEC-MDP where the agents are given tile positions from the environment in each step and then select tiles to fill the pair of positions in the next step. The environment then checks whether this leads to a consistent tiling.

A naive approach could lead to the creation of a state for every pair of tile positions. Unfortunately, this would result in an exponential blow-up in problem size, rather than a polynomial reduction. Thus, the environment itself cannot "remember" all the information about the process. As it turns out, it is sufficient to only remember information about the relative position of two tiles (whether the locations are the same, horizontally adjacent, or

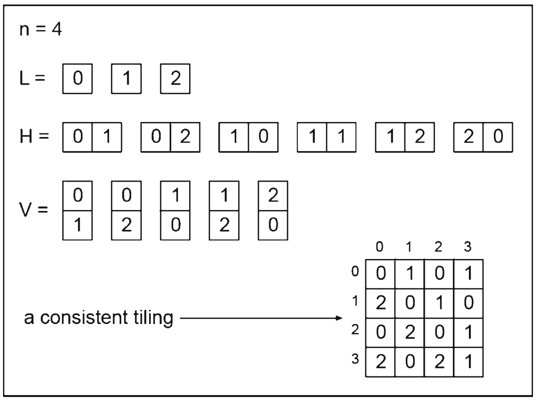

n = 4
L = 0 1 2
H = 0 1, 0 2, 1 0, 1 1, 1 2, 2 0
V = 0/1, 0/2, 1/0, 1/2, 2/0
a consistent tiling
0 1 2 3
0: 0 1 0 1
1: 2 0 1 0
2: 0 2 0 1
3: 2 0 2 1
Given n, L, H, and V, is there a consistent tiling?

Fig. 4 The tiling problem (courtesy of Daniel Bernstein)

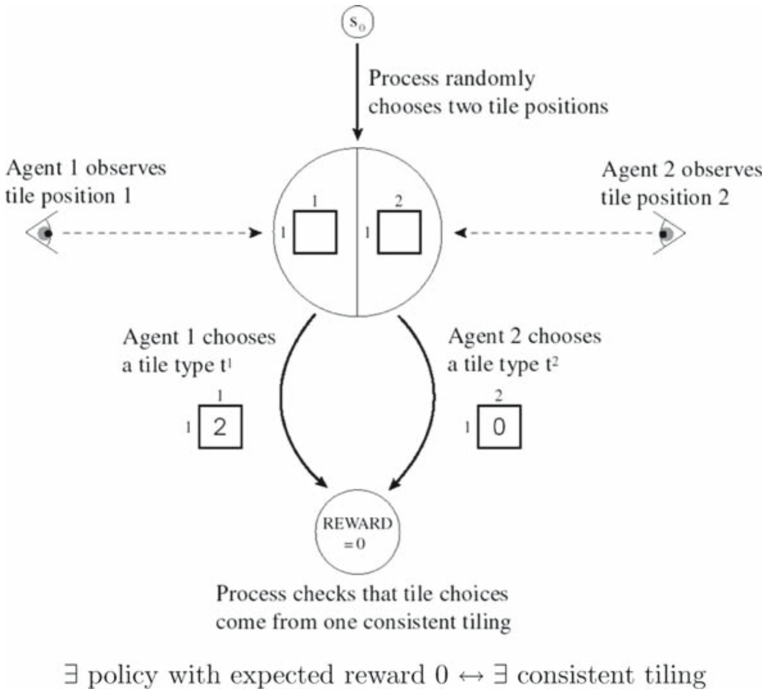

s_0
Process randomly chooses two tile positions
Agent 1 observes tile position 1
Agent 2 observes tile position 2
Agent 1 chooses a tile type t^1
Agent 2 chooses a tile type t^2
2
0
REWARD =0
Process checks that tile choices come from one consistent tiling
$\exists$ policy with expected reward 0 $\leftrightarrow$ $\exists$ consistent tiling

Fig. 5 Reduction from the tiling problem (courtesy of Daniel Bernstein)

vertically adjacent). The agent's local policies are based on their observation histories, and so if they observe a tile position produced by the environment, they can select an action in the next step based on this observation. Figure 5 illustrates how this process can be subdivided into the following five phases:

1. **Select phase**: Select two bit indices and values, each identifying a bit position and the value at that position in the location given to one of the agents.
2. **Generate phase**: Generate two tile locations at random, revealing one to each agent.
3. **Query phase**: Query each agent for a tile type to place in the location that was specified to that agent.
4. **Echo phase**: Require the agents to echo the tile locations they received in the generate phase bit by bit.
5. **Test phase**: Check whether the tile types provided in the query phase come from a single consistent tiling.

Note that because the tiling problem is reduced to a *DEC-MDP*, joint observability must be maintained throughout the whole process. That is, all aspects of the DEC-MDP must be available to the agents through their observations. This is achieved by making all the information observable by both agents, except for the indices and values selected in the select phase and the tile types that are chosen by the agents during the query phase.

Because the environment has the information about the relative position of the tiles, it can generate a reward depending on the chosen actions (the tile types) after each step. The agents receive a non-negative reward only if the tiling conditions hold. Thus, for the whole problem, the agents can expect a non-negative reward (exceeding the threshold value K) if and only if the tiling conditions can be met for all pairs of positions, i.e. there exists a consistent tiling.

Thus, a solution for the DEC-MDP$_n$ at the same time constitutes a solution to the tiling problem. Accordingly, since the tiling problem is NEXP-complete, solving a finite-horizon DEC-MDP$_n$ is NEXP-hard. □

Corollary 2 *For any $n \geq 2$, DEC-POMDP$_n$ is NEXP-hard.*

Proof Because the set of all DEC-POMDPs is a true superset of the set of all DEC-MDPs, this follows immediately from Theorem 5. □

Because all four formal models for optimal multi-agent planning are equivalent, the last complexity result also holds for DEC-POMDP-COMs, MTDPs and COM-MTDPs. Thus, in future research, any one of the models can be used for purposes of theoretical analysis, as well as for the development of algorithms. The results will always apply to the other models as well. Note that we have only analyzed the complexity of finite-horizon problems in this section. This is because the situation is significantly different for the infinite-horizon case. In 1999, Madani et al. [25] showed that infinite-horizon POMDPs are undecidable. Thus, because a DEC-POMDP with one agent reduces to a single-agent POMDP, it is obvious that infinite-horizon DEC-POMDPs are also undecidable.

2.3.3 Approximation and complexity

The notion of equivalence used so far means that the decision problem versions of all four models have the same worst-case complexity. However, it does not immediately imply anything about the approximability of their corresponding optimization problems. An understanding of this approximability is nevertheless desirable in light of the high worst-case complexity. The standard polynomial time reductions we have used are inadequate to show equivalence of two optimization problems in terms of approximability. Two problems A and B may be equivalent in terms of worst-case complexity, but it can still be conceivable that A has an efficient approximation algorithm while B does not.

A more careful kind of reduction that preserves approximability is called an *L-Reduction*. Showing that there exists an L-reduction from optimization problem A to optimization problem B when there exists an approximation algorithm for B implies that there also exists an approximation algorithm for A. The formal details of this reduction technique are beyond the scope of this paper (see Papadimitriou [31, chap. 13] for a detailed discussion of this topic). However, we can give some intuition and an informal proof of why the four models considered here are also equivalent in terms of approximability.

First, we use the result shown by Rabinovich et al. [40] in 2003 that even ε-optimal history-dependent joint policies are still NEXP-hard to find in DEC-POMDPs. In other words, DEC-POMDPs are not efficiently approximable. As the DEC-POMDP-COM model is a straightforward extension of the DEC-POMDP model, DEC-POMDP-COMs are also not efficiently approximable. An analogous argument holds for the MTDP and the MTDP-COM model. If the MTDP model is not efficiently approximable the same would hold true for the MTDP-COM model. Thus, it suffices to show that there is an L-reduction from the DEC-POMDP model to the MTDP model.

Looking at the reduction used in Sect. 2.1.3, we can observe that we used a very easy one-to-one mapping of the elements of a DEC-POMDP to the resulting MTDP. Most importantly, this mapping can be done in logarithmic space, one of the main requirements of L-reductions. A second requirement is that the solution for the resulting MTDP can be transformed into a solution for the original DEC-POMDP using logarithmic space. As we have noted before, a solution to the resulting MTDP is at the same time a solution to the original DEC-POMDP with the same solution value, and thus this requirement is trivially fulfilled. The last requirement of L-reductions concerns the optimal values of the two problem instances—the original DEC-POMDP and the MTDP resulting from the mapping. The optimal solutions for both instances must be within some multiplicative bound of each other. Again, because the solutions for the two problem instances will actually have the exact same value, this last requirement of the L-reduction is also fulfilled, and we have shown that MTDPs are not efficiently approximable. Thus, all four models are also equivalent in terms of approximability; finding ε-approximations for them is NEXP hard.

2.4 Models with explicit belief representation

In this section, a different approach to formalize the multi-agent planning problem is presented. The approach proves to be fundamentally different from the models presented in the previous sections, in terms of both expressiveness and complexity.

Single-agent planning is known to be P-complete for MDPs and PSPACE-complete for POMDPs. The unfortunate jump in complexity to NEXP when going from 1 to 2 agents is due to the fact that in the DEC-POMDP model there is probably no way of compactly representing a policy (see also Sect. 2.5.2 for further discussion of this topic). Agents must remember their complete histories of observations, resulting in exponentially large policies. If there were a way to construct a compact belief state about the whole decentralized process, one could hope to get a lower complexity than NEXP. For each agent, this would include not only expressing a belief about its own local state, but also its belief about the other agents (their states, their policies, etc.). Such belief states are a central component of the I-POMDP model introduced by Gmytrasiewicz and Doshi [16]. This framework for sequential planning in partially observable multi-agent systems is even more expressive than the DEC-POMDP model, as it allows for non-cooperative as well as cooperative settings. Thus, it is closely related to partially observable stochastic games (POSGs). In contrast to classical game theory, however, this approach does not search for equilibria or other stability criteria. Instead, it focuses on finding the best response action for a single agent with respect to its belief about the other agents, thereby avoiding the problems of equilibria-based approaches, namely the existence of multiple equilibria.

The formal I-POMDP model and corresponding solution techniques are presented below. In Sect. 2.4.3, we examine its complexity and in Sect. 2.4.4, we discuss the applicability of this model and its relationship to DEC-POMDPs.

2.4.1 The I-POMDP model

I-POMDPs extend the POMDP model to the multi-agent case. Now, in addition to a belief over the underlying system state, a belief over the other agents is also maintained. To model this richer belief, an *interactive state space* is used. A belief over an interactive state subsumes the belief over the underlying state of the environment as well as the belief over the other agents. Notice that—even if just two agents are considered—expressing a belief over another agent might include a belief over the other agent's belief. As the second agent's belief might also include a belief over the first agent's belief, this technique leads to a nesting of beliefs which makes finding optimal solutions within this model problematic. This topic is discussed later in Sects. 2.4.3 and 2.4.4.

To capture the different notions of beliefs formally, some definitions have to be introduced. The nesting of beliefs then leads to the definition of *finitely nested I-POMDPs* in the next section.

Definition 21 (*Frame*) A frame of an agent i is $\hat{\theta}_i = \langle A_i, \Omega_i, T_i, O_i, R_i, OC_i \rangle$, where:

- A_i is a set of actions agent i can execute.
- Ω_i is a set of observations the agent i can make.
- T_i is a transition function, defined as $T_i : S \times A_i \times S \rightarrow [0, 1]$.
- O_i is the agent's observation function, defined as $O_i : S \times A_i \times \Omega_i \rightarrow [0, 1]$.
- R_i is a reward function representing agent $i's$ preferences, defined as $R_i : S \times A_i \rightarrow \Re$.
- OC_i is the agent's optimality criterion. This specifies how rewards acquired over time are handled. For a finite horizon, the expected value of the sum is commonly used. For an infinite horizon, the expected value of the discounted sum of rewards is commonly used.

Definition 22 (*Type* $\equiv$ *intentional model*)[2] A type of an agent i is $\theta_i = \langle b_i, \hat{\theta}_i \rangle$, where:

- b_i is agent i's state of belief, an element of $\Delta(S)$, where S is the state space.
- $\hat{\theta}_i$ is agent i's frame.

Assuming that the agent is Bayesian-rational, given its type θ_i, one can compute the set of optimal actions denoted $OPT(\theta_i)$.

Definition 23 (*Models of an agent*) The set of possible models of agent j, M_j, consists of the subintentional models, SM_j, and the intentional models IM_j. Thus, $M_j = SM_j \cup IM_j$. Each model, $m_j \in M_j$ corresponds to a possible belief about the agent, i.e. how agent j maps possible histories of observations to distributions of actions.

- *Subintentional models* $\boldsymbol{SM_j}$ are relatively simple as they do not imply any assumptions about the agent's beliefs. Common examples are no-information models and fictitious play models, both of which are history independent. A more powerful example of a subintentional model is a finite state controller.
- *Intentional models* $\boldsymbol{IM_j}$ are more advanced, because they take into account the agent's beliefs, preferences and rationality in action selection. Intentional models are equivalent to types.

I-POMDPs generalize POMDPs to handle the presence of, and interaction with, other agents. This is done by including the types of the other agents into the state space and then expressing a belief about the other agents' types. For simplicity of notation, here just two agents are considered, i.e. agent i interacting with one other agent j, but the formalism is easily extended to any number of agents.

Definition 24 (*I-POMDP*) An interactive POMDP of agent i, I-POMDP$_i$, is a tuple $\langle IS_i, A, T_i, \Omega_i, O_i, R_i \rangle$, where:

- IS_i is a set of *interactive* states, defined as $IS_i = S \times M_j$, where S is the set of states of the physical environment, and M_j is the set of possible models of agent j. Thus agent i's belief is now a probability distribution over states of the environment and the models of the other agent: $b_i \in \Delta(IS_i) \equiv b_i \in \Delta(S \times M_j)$.
- $A = A_i \times A_j$ is the set of joint actions of all agents.

[2] The term "model" is used in connection with the definition of the state space (see the following definition of I-POMDP). The term "type" is used in connection with everything else.

- T_i is the transition function, defined as $T_i : S \times A \times S \rightarrow [0, 1]$. This implies the *model non-manipulability assumption (MNM)*, which ensures agent autonomy, since an agent's actions do not change the other's models directly. Actions can only change the physical state and thereby may indirectly change the other agent's beliefs via received observations.
- Ω_i is the set of observations that agent i can make.
- O_i is an observation function, defined as $O_i : S \times A \times \Omega_i \rightarrow [0, 1]$. This implies the *model non-observability assumption (MNO)*, which ensures another aspect of agent autonomy, namely that agents cannot observe each other's models directly.
- R_i is the reward function, defined as $R_i : IS_i \times A \rightarrow \Re$. This allows the agents to have preferences depending on the physical states as well as on the other agent's models, although usually only the physical state matters.

Definition 25 (*Belief update in I-POMDPs*) Under the MNM and MNO assumptions, the belief update function for an I-POMDP $\langle IS_i, A, T_i, \Omega_i, O_i, R_i \rangle$ given the interactive state $is^t = \langle s^t, m_j^t \rangle$ is:

$$b_i^t\left(is^t\right) = \beta \sum_{is^{t-1}:\hat{m}_j^{t-1}=\hat{\theta}_j^t} b_i^{t-1}\left(is^{t-1}\right) \sum_{a_j^{t-1}} Pr\left(a_j^{t-1}|\theta_j^{t-1}\right) O_i\left(s^t, a^{t-1}, o_i^t\right)$$
$$\cdot T_i\left(s^{t-1}, a^{t-1}, s^t\right) \sum_{o_j^t} \tau_{\theta_j^t}\left(b_j^{t-1}, a_j^{t-1}, o_j^t, b_j^t\right) O_j\left(s^t, a^{t-1}, o_j^t\right)$$

where:

- β is a normalizing constant.
- $\sum_{is^{t-1}:\hat{m}_j^{t-1}=\hat{\theta}_j^t}$ is the summation over all interactive states where agent j's frame is $\hat{\theta}_j^t$.
- b_j^{t-1} and b_j^t are the belief elements of agent j's types θ_j^{t-1} and θ_j^t respectively.
- $Pr(a_j^{t-1}|\theta_j^{t-1})$ is the probability that for the last time step action a_j was taken by agent j given its type. This probability is equal to $\frac{1}{OPT(\theta_j)}$ if $a_j^{t-1} \in OPT(\theta_j)$, else it is equal to zero. Hereby OPT denotes the set of optimal actions.
- O_j is agent j's observation function in m_j^t.
- $\tau_{\theta_j^t}(b_j^{t-1}, a_j^{t-1}, o_j^t, b_j^t)$ represents the update of j's belief.

An agent's belief over interactive states is a sufficient statistic, i.e. it fully summarizes its history of observations. For a detailed proof of the sufficiency of the belief update see [16].

Definition 26 (*State estimation function SE_{θ_i}*) As an abbreviation for belief updates, the following state estimation function is used: $b_i^t(is^t) = SE_{\theta_i}(b_i^{t-1}, a_i^{t-1}, o_i^t)$.

As in POMDPs, the new belief b_i^t in an I-POMDP is a function of the previous belief state, b_i^{t-1}, the last action, a_i^{t-1}, and the new observation, o_i^t. Two new factors must also be taken into account, however. First, the change in the physical state now depends on the actions performed by both agents, and agent i must therefore take into account j's model in order to infer the probabilities of j's actions. Second, changes in j's model itself have to be included in the update, because the other agent also receives observations and updates its belief. This update in the other agent's belief is represented by the τ-term in the belief update function.

Obviously, this leads to a nested belief update: i's belief update invokes j's belief update, which in turn invokes i's belief update and so on. The update of the possibly infinitely nested beliefs raises the important question of how far optimality could be achieved with this model. This problem is discussed in more detail in Sects. 2.4.3 and 2.4.4.

Definition 27 (*Value function in I-POMDPs*) Each belief state in an I-POMDP has an associated value reflecting the maximum payoff the agent can expect in that belief state:

$$U(\theta_i) = \max_{a_i \in A_i} \left\{ \sum_{is} ER_i(is, a_i) b_i(is) + \gamma \sum_{o_i \in \Omega_i} Pr(o_i | a_i, b_i) \cdot U(\langle SE_{\theta_i}(b_i, a_i, o_i), \hat{\theta}_i \rangle) \right\}$$

where $ER_i(is, a_i)$ is the expected reward for state is taking action a_i, defined by: $ER_i(is, a_i) = \sum_{a_j} R_i(is, a_i, a_j) Pr(a_j | m_j)$. This equation is the basis for value iteration in I-POMDPs.

Definition 28 (*Optimal actions in I-POMDPs*) Agent i's optimal action, a_i^*, for the case of an infinite-horizon criterion with discounting, is an element of the set of optimal actions for the belief state, $OPT(\theta_i)$, defined as:

$$OPT(\theta_i) = \text{argmax}_{a_i \in A_i} \left\{ \sum_{is} ER_i(is, a_i) b_i(is) \right.$$
$$\left. + \gamma \sum_{o_i \in \Omega_i} Pr(o_i | a_i, b_i) U(\langle SE_{\theta_i}(b_i, a_i, o_i), \ \hat{\theta}_i \rangle) \right\}$$

Due to possibly infinitely nested beliefs, a step of value iteration and optimal actions are only asymptotically computable [16].

2.4.2 Finitely nested I-POMDPs

Obviously, infinite nesting of beliefs in I-POMDPs leads to non-computable agent functions. To overcome this, the nesting of beliefs is limited to some finite number, which then allows for computing value functions and optimal actions (with respect to the finite nesting).

Definition 29 (*Finitely nested I-POMDP*) A finitely nested I-POMDP of agent i, I-POMDP$_{i,l}$, is a tuple $\langle IS_{i,l}, A, T_i, \Omega_i, O_i, R_i \rangle$, where everything is defined as in normal I-POMDPs, except for the parameter l, the *strategy level* of the finitely nested I-POMDP. Now, the belief update, value function, and the optimal actions can all be computed, because recursion is guaranteed to terminate at the 0-th level of the belief nesting. See [16] for details of the inductive definition of levels of the interactive state space and the strategy.

At first sight, reducing the general I-POMDP model to finitely nested I-POMDPs seems to be a straightforward way to achieve asymptotic optimality. But the problem with this approach is that now every agent is only boundedly optimal. An agent with strategy level l might fail to predict the actions of a more sophisticated agent because it has an incorrect model of that other agent. To achieve unbounded optimality, an agent would have to model the other agent's model of itself. This obviously leads to an impossibility result due to the self-reference of the agents' beliefs. Gmytrasiewicz and Doshi [16] note that some convergence results from Kalai and Lehrer [24] strongly suggest that approximate optimality is achievable. But the applicability of this result to their framework remains an open problem. In fact, it is doubtful that the finitely nested I-POMDP model allows for approximate optimality. For any fixed value of the strategy level, it may be possible to construct a corresponding problem such that the difference between the value of the optimal solution and the approximate solution can be arbitrarily large. This remains an interesting open research question.

2.4.3 Complexity analysis for finitely nested I-POMDPs

Finitely nested I-POMDPs can be solved as a set of POMDPs. At the 0-th level of the belief nesting, the resulting types are POMDPs, where the other agent's actions are folded into the T, O, and R functions as noise. An agent's first level belief is a probability distribution over S and 0-level models of the other agents. Given these probability distributions and the solutions to the POMDPs corresponding to 0-level beliefs, level-1 beliefs can be obtained by solving further POMDPs. This solution then provides probability distributions for the next level model, and so on. Gmytrasiewicz and Doshi assume that the number of models considered at each level is bounded by a finite number, M, and show that solving an I-POMDP$_{i,l}$ is then equivalent to solving $O(M^l)$ POMDPs. They conclude that the complexity of solving an I-POMDP$_{i,l}$ is PSPACE-hard for finite time horizons, and undecidable for infinite horizons, just as for POMDPs. Furthermore, PSPACE-completeness is established for the case where the number of states in the I-POMDP is larger than the time horizon.

This complexity analysis relies on the assumption that the strategy level is fixed. Otherwise, if the integer l is a variable parameter in the problem description, this may change the complexity significantly. If l is encoded in binary, its size is $log(l)$. Thus, if solving an I-POMDP$_{i,l}$ is in fact equivalent to solving $O(M^l)$ POMDPs, this amounts to solving a number of POMDPs doubly exponential in the size of l, which requires at least double exponential time. Because PSPACE-hard problems can be solved in exponential time, this implies that solving an I-POMDP$_{i,l}$ is strictly harder than solving a POMDP. To establish tight complexity bounds, a formal reduction proof would be necessary. However, it seems that any optimal algorithm for solving an I-POMDP$_{i,l}$ will take double exponential time in the worst case.

2.4.4 Applicability of I-POMDPs and relationship to DEC-POMDPs

As mentioned earlier, the I-POMDP model is more expressive than the DEC-POMDP model, because I-POMDPs represent an individual agent's point of view on both the environment and the other agents, thereby also allowing for non-cooperative settings. For cooperative settings addressed by the DEC-POMDP framework, multiple I-POMDPs have to be solved, where all agents share the same reward function. Furthermore, the models of the other agents must include the information that all the agents are cooperating in order to compute an adequate belief. In practice, exact beliefs cannot be computed when they are infinitely nested. Thus, the I-POMDP model has the following drawbacks:

1. Unlike the DEC-POMDP model, the I-POMDP model does not have a corresponding optimal algorithm. The infinite nesting of beliefs leads to non-computable agent functions. Due to self-reference, finitely nested I-POMDPs can only yield bounded optimal solutions.
2. Solving finitely nested I-POMDPs exactly is also challenging because the number of possible computable models is infinite. Thus, in each level of the finitely nested POMDP, agent i would have to consider infinitely many models of agent j, compute their values and multiply them by the corresponding probabilities. The complexity analysis relies on the additional assumption that the number of models considered at each level is bounded by a finite number, which is another limiting factor. No bounds on the resulting approximation error have been established. While in some case it may be possible to represent the belief over infinitely many models compactly, there is no general technique for doing so.

3. Even with both approximations (i.e. finite nesting and a bounded number of models), I-POMDPs seem to have a double-exponential worst-case time complexity, and are thus likely to be at least as hard to solve as DEC-POMDPs.

Despite these drawbacks, the I-POMDP model presents an important alternative to DEC-POMDPs. While no optimal algorithm exists, this is not much of a drawback, since optimality can only be established for very small instances of the other models we have described. To solve realistic problems, approximation techniques, such as those described in Sects. 4 and 5, must be used. With regard to existing benchmark problems, DEC-POMDP approximation algorithms have been the most effective in solving large instances (see Sect. 4.2). Additional approximation techniques for the various models are under development. To better deal with the high belief-space complexity of I-POMDPs, a recently developed technique based on particle filtering has been shown to improve scalability[13,14]. Thus, the relative merits of I-POMDP and DEC-POMDP approximation algorithms remains an open question.

2.5 Sub-classes of DEC-POMDPs

2.5.1 Motivation for analyzing sub-classes

Decentralized control of multiple agents is NEXP-hard, even for the 2-agent case with joint full observability. Thus, all these problems are computationally intractable. Furthermore, it has recently been shown by Rabinovich et al. [40] that even finding ε-optimal solutions for a DEC-MDP remains NEXP-hard. To overcome this complexity barrier, researchers have identified specific sub-classes of the DEC-POMDP model whose complexity range from P to NEXP. Some of these sub-classes are also of practical interest, because they nicely describe real-world problems.

Although all the problems introduced in Sect. 1.1 are decentralized control processes and are NEXP-complete in the worst case, they still differ in their level of decentralization. In some decentralized problems, agents are very dependent on each other in that their actions have greater influence on each other's next state (e.g. in the multi-access broadcast channel problem). In other problems, agents solve (mostly) independent local problems and only interact with one another very rarely, or their actions only slightly influence each other's state (e.g. meeting on the grid, Mars rover navigation).

If one describes this "level of decentralization" more formally, an interesting sub-class of DEC-MDPs emerges, namely transition and observation independent DEC-MDPs. In this model, the agents' actions do not affect each other's observations or local states. Moreover, the agents cannot communicate and cannot observe each other's observations or states. The only way agents interact is through a global value function, which makes the problem decentralized, because it is not simply the sum of the rewards obtained by each agent but some non-linear combination.

A real-world example of such a problem is that of controlling the operation of multiple planetary exploration rovers, such as those used by NASA to explore the surface of Mars [50] (illustrated in Fig. 6). Each rover explores and collects information about its own particular region. Periodically, the rovers can communicate with the ground control center but continuous communication is not possible. Some of the exploration sites overlap, and if two rovers collect information about these sites (e.g. take pictures) the reward is sub-additive. For other parts of the planet, the reward might be superadditive, for example if both rovers work together to build a 3D model of a certain area.

Fig. 6 Mars exploration rover

2.5.2 *Formal models*

As mentioned in Sect. 2.5.1, the level of interaction between the agents can be captured formally. To do so, we need to introduce the following definitions, adapted from Becker et al. [4] and Goldman and Zilberstein [19].

Definition 30 (*Factored n-agent DEC-MDP*) A factored n-agent DEC-MDP is a DEC-MDP such that the world state can be factored into $n + 1$ components, $S = S_0 \times S_1 \times \ldots \times S_n$.

This way, the features that only belong to one agent are separated from those of the others and from the external features. S_0 denotes external features that all agents observe and are affected by, but that are not changed by the agents' actions themselves. In our rover example, this might be features such as weather or time. S_i $(i > 0)$ refers to the set of state features for agent i, which denotes the part of the state set that only agent i may affect and observe. The local state of agent i then depends on its private state features and on the external features.

Definition 31 (*Local state/observation/action*) $\hat{s}_i \in S_i \times S_0$ is referred to as the local state, $a_i \in A_i$ as the local action, and $o_i \in \Omega_i$ as the local observation for agent i.

Now, with the formal definition of a factored state space in hand, transition independence of the agents can be formalized.

Definition 32 (*DEC-MDP with independent transitions*) A factored, n-agent DEC-MDP is transition independent if there exist P_0 through P_n such that

$$Pr(s_i' | (s_0, \ldots, s_n), \vec{a}, (s_1', \ldots, s_{i-1}', s_{i+1}', \ldots, s_n')) = \begin{cases} P_0(s_0' | s_0) & i = 0 \\ P_i(s_i' | \hat{s}_i, a_i, s_0') & 1 \le i \le n \end{cases}$$

That is, the external features change based only on the previous external features, and the new local state of each agent depends only on its previous local state, the local action taken and the current external features. A similar independence can be formalized for the observation function.

Definition 33 (*DEC-MDP with independent observations*) A factored, n-agent DEC-MDP is observation independent if there exist O_1 through O_n such that:

$$\forall o_j \in \Omega_i : Pr(o_j | (s_0, ..., s_n), \vec{a}, (s_0', ..., s_n'), (o_1, ..., o_{i-1}, o_{i+1}, ...o_n)) = O_i(o_j | \hat{s}_i, a_i, \hat{s}_i').$$

That is, the observation an agent receives depends only on that agent's current and next local state and current action.

Definition 34 (*Local full observability*) A factored, n-agent DEC-MDP is locally fully observable if

$$\forall o_i \, \exists \hat{s}_i : Pr(\hat{s}_i | o_i) = 1.$$

That is, at each step, an agent's observation fully determines its own local state. Note that while observation independence and local full observability are related, it is possible for one to be true without the other. However, if both hold, both the set of observations and the observation function in the definition of a factored n-agent DEC-MDP are redundant, and can be omitted.

It is intuitive that if an agent's transitions and observations in a DEC-MDP are independent from those of the other agents, the agent should be able to determine the local state given its local observation. Goldman and Zilberstein [19] proved the following theorem which formalizes this intuition:

Theorem 6 *If a DEC-MDP has independent observations and transitions, then the DEC-MDP is locally fully observable.*

Definition 35 (*Reward independence*) A factored, n-agent DEC-MDP is said to be reward independent if there exist f and R_1 through R_n such that

$$R((s_0, \ldots, s_n), \vec{a}, (s'_0, \ldots, s'_n)) \;=\; f(R_1(\hat{s}_1, a_1, \hat{s}'_1), \ldots, R_n(\hat{s}_n, a_n, \hat{s}'_n))$$

and

$$R_i(\hat{s}_i, a_i, \hat{s}'_i) \leq R_i(\hat{s}_i, a'_i, \hat{s}''_i) \;\Leftrightarrow$$
$$f(R_1 \ldots R_i(\hat{s}_i, a_i, \hat{s}'_i) \ldots R_n) \leq f(R_1 \ldots R_i(\hat{s}_i, a'_i, \hat{s}''_i) \ldots R_n)$$

That is, the overall reward is composed of a function of the local reward functions, each of which depends only on the local state and local action of one of the agents. Maximizing each of the local reward functions individually is sufficient to maximize the overall reward function.

Notice that a factored DEC-MDP with transition independence, observation independence and reward independence decomposes into n independent local MDPs. Thus, the problem becomes P-complete and uninteresting from the perspective of decentralized control. However, if just one of these independence relations is absent, the problem remains a non-trivial subclass of DEC-MDPs.

The following section concentrates on the class of factored DEC-MDPs that are transition and observation independent, but are not reward independent. Instead, the reward function consists of two different components. The first is a set of local reward functions that are independent, just as in the case with multiple independent MDPs. The second component is an additional global reward (or penalty) that depends on the actions of all the agents, along with the global state. Thus, the overall model is not reward independent. It is formally defined by the *joint reward structure*, denoted ρ. This allows for defining interdependent rewards for cases where the whole system gets a reward after multiple agents have successfully completed a task. The formal definition of a joint reward structure is beyond the scope of this paper. We mention it here because it is necessary for the first theorem in the next section. See Becker et al. [4] for a formal definition and further details.

2.5.3 Complexity results

The proofs for the theorems presented in this section can be found in Becker et al. [4] and Goldman and Zilberstein [19].

Theorem 7 *Deciding a DEC-MDP with independent transitions and observations, and joint reward structure ρ is NP-complete.*

At first sight, this model might seem very specialized and only applicable to a very small set of problems. It turns out, however, that the class of problems addressed is quite general. Any reward dependence between multiple agents can be formalized using the joint reward structure ρ. There can still be different levels of independence in the reward function, affecting the efficiency of the representation of the joint reward structure.

The Mars rover example presented earlier falls into this class of problems. The rovers have independent transitions and observations. They mainly solve their tasks independently, but there may be tasks where they must work together to get a reward, e.g. by taking different pictures of the same rock from different angles to be used in 3D model construction. A more detailed example and an optimal algorithm (Coverage Set Algorithm) for this model can be found in Becker et al. [4]. A significant improvement of the coverage set algorithm has been presented by Petrik and Zilberstein [36]. A discussion of the complexity gap between decentralized problems that are NEXP-complete and problems that are NP-complete can be found in [44]. Beynier and Mouaddib [9] have also studied a restricted model of interacting MPDs in which the interaction is defined by temporal constraints. Additional restrictions on the model give rise to another class of problems, which is even more specialized, and can be solved in polynomial time.

Definition 36 (*Finite-horizon goal-oriented DEC-MDPs (GO-DEC-MDPs)*)[3] A finite-horizon DEC-MDP is goal-oriented if the following conditions hold:

1. There exists a special subset G of S of global goal states. At least one of the global goal states $g \in G$ is reachable by some joint policy.
2. The process ends at time T (the finite horizon of the problem).
3. All actions in A incur a cost, $C(a_i) < 0$. For simplicity, it is assumed that the cost of an action depends only on the action. In general, this cost may also depend on the state.
4. The global reward is $R(s, \vec{a}, s') = C(a_1) + \ldots + C(a_n)$.
5. If at time T the system is in a state $s \in G$, there is an additional reward $JR(s) \in \Re$ that is awarded to the system for reaching a global goal state.

Definition 37 (*Uniform cost*) A goal-oriented DEC-MDP has uniform cost when the costs of all actions are the same.

Theorem 8 *Deciding a goal-oriented DEC-MDP with independent transitions and observations, with a single global goal state and with uniform cost is P-complete.*

The following theorem helps explain the drop in complexity to NP and P respectively with the last two models.

[3] Adapted from Goldman and Zilberstein [19].

Theorem 9 *The current local state of agent i, $\hat{s}_i$, is a sufficient statistic for the past history of observations ($\overline{o_1}$) of a locally fully observable DEC-MDP with independent transitions and observations.*

This implies that if the DEC-MDP is locally fully observable, the policy is no longer a mapping from observation histories to actions, but a mapping from local states to actions. A local policy of agent i is of size polynomial in $|S_i|T$. Figure 7 shows schematically the difference in policy representation size which provides an intuition for the aforementioned differences in complexity.

We conclude this section with an overview of the different problems that emerge and how the complexity depends on the restrictions one imposes on the way the agents interact and the way they observe their environment. We start with some definitions regarding observability and communication.

Definition 38 (*Partial observability ≡ collective partial observability*) A DEC-POMDP is partially observable if the observations of all agents together still do not determine the global state.

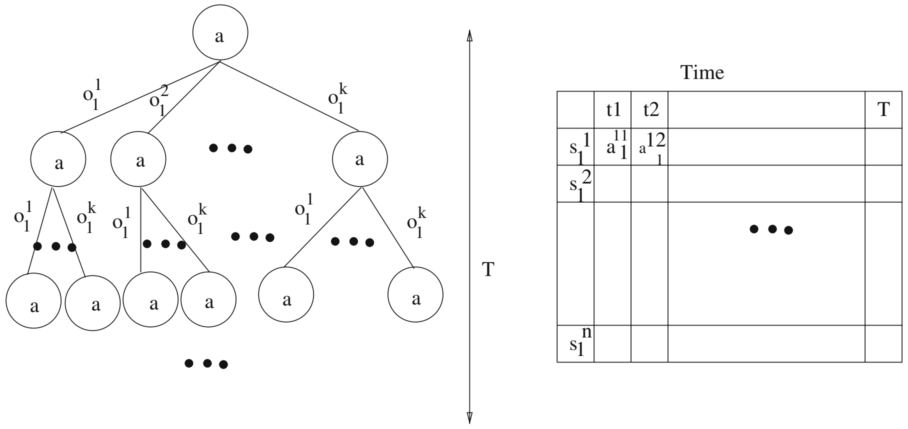
Definition 39 (*Full observability ≡ individual observability*) A DEC-POMDP is fully observable if there exists a mapping for each agent i, $f_i : \Omega_i \rightarrow S$ such that whenever $O(\vec{o}|s, \vec{a}, s')$ is non-zero then $f_i(o_i) = s'$.

That is, each agent's partial view (local observation) is sufficient to uniquely determine the global state of the system.

Definition 40 (*MMDP*) A multi-agent Markov decision process is a DEC-POMDP with full observability [10].

An MMDP is a straightforward extension of the completely observable MDP model, where single actions are simply replaced by vectors of actions. Thus, the complexity remains P-complete.

So far, the main focus has been on complexity due to different levels of observability. The way agents interact via communication is also of great interest, however, from a practical



Agent 1's local policy when a sequence of observations should be remembered.

Agent 1's local policy, when the process is locally fully observable.

Fig. 7 Exponential versus polynomial size policies (courtesy of Claudia Goldman)

Table 2 Complexity results given observability and communication

	General communication	Free communication
Full observability	MMDP (P-complete)	MMDP (P-complete)
Joint full observability	DEC-MDP (NEXP-complete)	MMDP (P-complete)
Partial observability	DEC-POMDP (NEXP-complete)	MPOMDP (PSPACE-complete)

point of view as well as for complexity analyses. See [18,19,39] and for broader coverage of the subject.

Definition 41 (*Free communication*) Agents in a decentralized process have free communication if they can share all information with each other at every step without incurring any cost.

Definition 42 (*General communication*) In the case of general communication, costs for communication actions can be defined either implicitly by the reward function or explicitly by a separate cost function, provided the cost for at least one communication action must be strictly greater than 0 (otherwise it is the case of free communication).

The resulting models and corresponding complexity classes depend on the level of observability and on the level of communication between the agents. A summary overview for finite-horizon problems is given in Table 2.

Obviously, free communication does not lower the complexity in the case of full observability, because every single agent already observes all the information that is available. But when full observability is not given, the agents benefit significantly from free communication in that they can share all the information available at no cost, resulting in lower complexity classes. Note that these results are based on the assumptions that (a) the communication language is expressive enough to share all the observations, and (b) the agents agree beforehand on the meaning of all the messages. If this is not the case, free communication does not automatically lead to full observability, and the problem of learning the semantics of messages arises [17].

3 Algorithms for optimal multi-agent planning

The complexity results from Sect. 2.3.2 suggest that any optimal algorithm for solving problems stated as a DEC-POMDP will use double exponential time in the worst case. Nevertheless, analyzing the performance of different approaches to solving DEC-POMDPs optimally can provide insights into specific computational barriers and facilitate the design of good approximation techniques. We begin with the description of two optimal algorithms for finite-horizon DEC-POMDPs in the next section, followed by the description of an ε-optimal algorithm for infinite-horizon DEC-POMDPs in Sect. 3.2.

3.1 Optimal finite-horizon algorithms

For finite-horizon problems, a policy for a single agent can be represented as a decision tree q, where nodes are labeled with actions and arcs are labeled with observations. A solution to a DEC-POMDP with horizon t can then be seen as a vector of horizon-t policy trees $\delta^t = (q_1^t, q_2^t, \ldots, q_n^t)$, one for each agent, where $q_i^t \in Q_i^t$. (In this section, it is assumed that

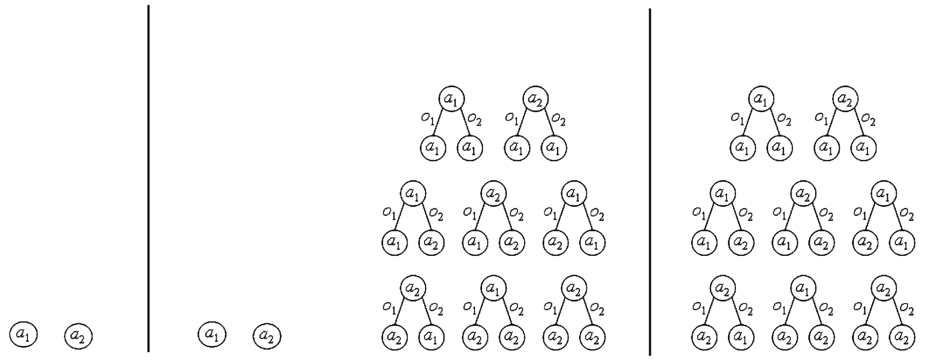


Fig. 8 Possible policies for horizon 1 and 2 (courtesy of Daniel Bernstein)

the initial state of the DEC-POMDP is known. If it is not known, and a probability distribution over states is given instead, $|S|$-dimensional vectors of policy vectors are considered, one for every possible initial state, and then the overall value is computed relative to that probability distribution.) Figure 8 gives an example for joint-decision trees and illustrates how fast the number of possible policies grows for a problem with 2 actions and 2 observations when going from horizon 1 to horizon 2.

An incremental construction for the set of possible policy trees is used. The set of horizon-$(t+1)$ policy trees Q^{t+1} can be constructed from a set of parent horizon-t policy trees Q^t by expanding the leaf nodes of every policy tree. For each leaf node in $q^t \in Q^t$, $|\Omega|$ child nodes are constructed. Each of the new leaf nodes has to be assigned an action. The total number of possible policy trees for one agent is the number of possible ways to assign different combinations of actions. For the horizon 1, only one action has to be picked, and the number of possible policy trees is $|A|$. For horizon 2, $|O|$ possible observations are added after taking the first action and then, for each observation, one new action has to be assigned. This leads to 1 action at the first level and $|O|$ actions at the second level. Thus, the resulting number of possible policy trees for horizon 2 is $|A|^{(1+|O|)}$ possible policy trees for horizon 2. For horizon 3, again $|O|$ observations are added for each leaf node in the horizon-2 policy tree and each of them gets assigned a new action. This leads to $|O|^2$ action assignments on the third level. Thus the resulting number of possible policy trees for horizon 3 is $|A|^{(1+|O|+|O|^2)}$. It follows that in general, for horizon t, the number of possible policy trees is $|A|^{(1+|O|+|O|^2+\cdots+|O|^{t-1})} = |A|^{\frac{|O|^t-1}{|O|-1}} \in O(|A|^{(|O|^t)})$. Thus the complexity of the algorithm is double exponential in the horizon t. Note that for $|O| \geq 2$ and $t \geq 2$: $|A|^{|O|^{t-1}} < |A|^{\frac{|O|^t-1}{|O|-1}} < |A|^{|O|^t}$.

Unfortunately, the problem is also exponential in the number of agents, as the number of joint policies is the product of the number of possible policies for all agents. Thus the number of possible joint policies is $(|A|^{\frac{|O|^t-1}{|O|-1}})^n$. Table 3 illustrates the infeasibility of a complete search through this space, even when only two agents are involved.

Despite the fact that every optimal solution technique for finite-horizon DEC-POMDPs will use double exponential time in the worst case, researchers have introduced two non-trivial algorithms that solve the problem optimally, but achieve a much better performance than does complete search. Two very different programming paradigms have been used in these efforts: dynamic programming in one case and heuristic search in the other. We describe the two algorithms below.

Table 3 Possible policies with 2 actions and 2 observations

Horizon	Policies for one agent	Joint-policies for 2 agents
1	2	4
2	8	64
3	128	16,384
4	32,768	1,073,741,824
5	2,147,483,648	$4.6 \cdot 10^{18}$

3.1.1 Dynamic programming for DEC-POMDPs and POSGs

The first non-trivial algorithm for finding optimal solutions in DEC-POMDPs was presented by Hansen et al. [22]. This algorithm finds sets of policies for partially observable stochastic games (POSGs), which are a special kind of extensive games with imperfect information [30]. POSGs only differ from DEC-POMDPs in that they allow for one private reward function per agent, thus allowing non-cooperative behavior. When applied to DEC-POMDPs, this algorithm finds an optimal joint policy. The algorithm generalizes two previously developed techniques: dynamic programming for POMDPs [21] and elimination of dominated strategies in normal form games.

The main idea of dynamic programming for POMDPs is to incrementally search through the space of possible policies, pruning dominated policies as early as possible in the construction process to avoid the full exhaustive backup. Therefore, the POMDP first has to be converted into a completely observable MDP with a state set $\mathbf{B} = \Delta S$ that consists of all possible beliefs about the current state. Furthermore, the algorithm exploits the fact that the value function for a POMDP can be represented exactly by a finite set of $|S|$-dimensional value vectors, denoted $V = \{v_1, v_2, \dots, v_k\}$, where

$$V(b) = \max_{1 \le j \le k} \sum_{s \in S} b(s) v_j(s)$$

Every value vector v_j corresponds to a complete conditional plan, i.e. a policy tree. The dynamic programming (DP) operator exploits this fact when pruning dominated policy tress.

Definition 43 (*Dominated policy trees*) A policy tree $q_j \in Q^t$ with corresponding value vector $v_j \in V^t$ is considered dominated if for all $b \in \mathbf{B}$ there exists a $v_k \in V^t \backslash v_j$ such that $b \cdot v_k \ge b \cdot v_j$.

In every iteration of dynamic programming, the DP operator is first given a set Q^t of depth-t policy tress and a corresponding set V^t of values vectors. The sets Q^{t+1} and V^{t+1} are then created by an *exhaustive backup*. This operation generates every possible depth-$t+1$ policy tree that makes a transition, after an action and observation, to the root node of some depth-t policy tree. In the next step, all dominated policy trees are eliminated from the set Q^{t+1} (the test for dominance is performed using linear programming). This can be done because a decision maker that is maximizing expected value will never follow a dominated policy and thus the DP operator does not decrease the value of any belief state. After the exhaustive backup and before the pruning step, the size of the set of policy trees is $|Q^{t+1}| = |A||Q^t|^{|O|}$.

Unfortunately, the DP operator for POMDPs cannot be generalized in a straightforward way, because there is no notion of a belief state in a DEC-POMDP. Only beliefs about the strategies of other agents can be formulated, similar to some ideas concerning normal form games (cf. [22]). A generalized belief state has to be defined, synthesizing a belief over possible states and a distribution over the possible policies of the other agents. Let Q_{-i}^t

denote the set of horizon-t policy trees for all agents except i. For each agent i, a belief for a horizon-t DEC-POMDP is now defined as a distribution over $S \times Q^t_{-i}$. The set of value vectors for agent i is thus of dimension $|S \times Q^t_{-i}|$. This set of value vectors is the basis for elimination of very weakly dominated policies in the multi-agent case, just as for POMDPs.

Definition 44 (*Very Weakly Dominated Policy Trees*) A policy tree $q_j \in Q^t_i$ with corresponding value vector $v_j \in V^t_i$ is very weakly dominated if

$$\forall b \in \Delta(S \times Q^t_{-i}), \exists v_k \in V^t_i \setminus v_j \quad \text{such that} \quad b \cdot v_k \geq b \cdot v_j$$

Now, multiple agents are considered instead of one single agent. When agent i eliminates its dominated policies this can affect the best policy of agent j. Thus, elimination of policies has to include alternation between agents until no agent can eliminate another policy. This procedure is called *iterated elimination of dominated strategies*. With the generalized belief state, given a horizon-t POSG it would now be possible to generate all horizon-t policy trees and the corresponding value vectors. Then, iterated elimination of very weakly dominated strategies could be applied to solve the problem and find the optimal policy. Unfortunately, as pointed out in the beginning, generating all possible horizon-t policy trees is infeasible due to the double exponential dependence on the time horizon t.

The DP operator for POMDPs can be generalized to the multi-agent case by interleaving exhaustive backups with pruning of dominated policies. In the first step of an iteration, the multi-agent DP operator is given Q^t_i and generates Q^{t+1}_i. In the second step, all very weakly dominated policies are pruned. As in the single agent case, this does not reduce the overall value because for every policy tree that has a very weakly dominated subtree there is a probability distribution over other policy trees that leads to a stochastic policy yielding the same or higher value. Thus, applying dynamic programming for POSGs eventually leads to a set of horizon-t policy tress including the optimal solution. For the special case of a DEC-POMDP, the algorithm preserves at least one optimal policy. When the multi-agent DP operator reaches horizon t, a policy can be chosen, by extracting the highest-valued policy tree according to the initial state distribution.

Of course, the algorithm is still doubly exponential in the time horizon t in the worst case. Any improvement over brute force search depends on the amount of possible pruning, which depends on the particular problem. The algorithm has been tested on the multi-access broadcast channel problem, described in Sect. 1.1.1. Table 4 illustrates the possible improvements in practice. For the horizon 4, the dynamic programming algorithm produces less than 1% of the number of policy trees produced by the brute force algorithm. Note that the brute force algorithm could not actually complete iteration 4.

Unfortunately, the dynamic programming algorithm runs out of memory after the 4th iteration due to the rapid growth in the number of policies trees. Even with pruning, an exhaustive backup going from horizon 4 to horizon 5 would need to produce $2 \cdot 300^4$, or more than 16

Table 4 Performance comparison: number of policies for each agent

Horizon	Brute force search	Dynamic programming algorithm
1	(2, 2)	(2, 2)
2	(8, 8)	(6, 6)
3	(128, 128)	(20, 20)
4	(32,768, 32,768)	(300, 300)

billion S-dimensional vectors of policy trees, before even beginning the process of pruning. This explains why the algorithm runs out of memory long before running out of time.

3.1.2 MAA: a heuristic search algorithm for DEC-POMDPs*

In 2005, Szer et al. [48] presented an approach orthogonal to dynamic programming for DEC-POMDPs, based on heuristic search. As we discuss in Sect. 3.1.3, the method has both advantages and disadvantages over the dynamic programming algorithm. The algorithm is based on the widely used A* algorithm, and performs best-first search in the space of possible joint policies.

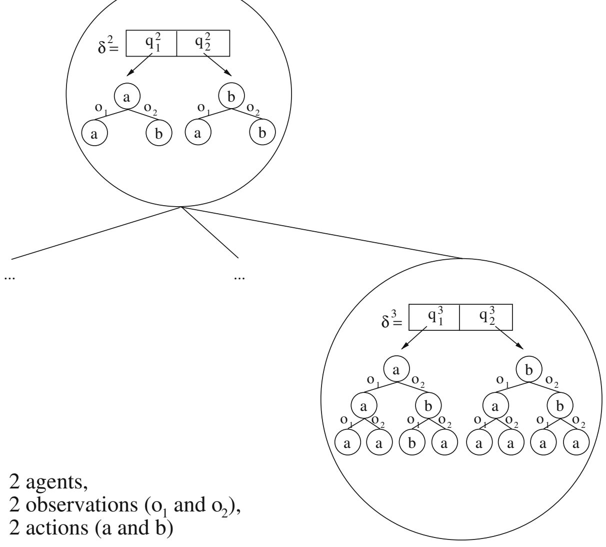
The algorithm uses the same representation for joint policies as presented earlier in this section: q_i^t is a depth-t policy tree for agent i and $\delta^t = (q_1^t, \ldots, q_n^t)$ is a policy vector of trees. Furthermore, let $V(s_0, \delta)$ denote the *expected value* of executing policy vector δ from state s_0. Finding the optimal joint policy is thus identical to finding $\delta^{*T} = \text{argmax}_{\delta^T} V(s_0, \delta)$. As in brute force search, the algorithm searches in the space of policy vectors, where nodes at level t of the search tree correspond to partial solutions of the problem, namely policy vectors of horizon t. But unlike complete search, not all nodes at every level are fully expanded. Instead, a heuristic function is used to evaluate the leaf nodes of the search tree. The node with the highest heuristic estimate is expanded in each step. Figure 9 shows a section of such a multi-agent search tree.

To compute the heuristic estimate of a search node, the evaluation function is decomposed into two parts: an exact evaluation of the partial solution (the policy vector up to the current level), and a heuristic estimate of the remaining part, the so called *completion*.

Definition 45 (*Completion of a policy vector*) A completion Δ^{T-t} of an arbitrary depth-t policy vector is a set of depth-$(T-t)$ policy trees that can be attached at the leaf nodes of a policy vector δ^t such that $\{\delta^t, \Delta^{T-t}\}$ constitutes a complete policy vector of depth T.

Now, the value of any depth-T policy vector also decomposes into two parts:

$$V(s_0, \{\delta^t, \Delta^{T-t}\}) = V(s_0, \delta^t) + V(\Delta^{T-t}|s_0, \delta^t)$$



$\delta^2 =$ q_1^2 q_2^2
a b o1 o2
...
$\delta^3 =$ q_1^3 q_2^3
2 agents,
2 observations (o_1 and o_2),
2 actions (a and b)

Fig. 9 A section of the multi-agent A* search tree, showing a horizon 2 policy vector with one of its expanded horizon 3 child nodes (courtesy of Daniel Szer)

Obviously, the value of the completion depends on the state distribution that is reached after executing policy vector δ^t in state s_0. Instead of computing its value exactly, we will estimate it efficiently via heuristic function H. We thus define the value estimate of state s_0 and policy δ^t as:

$$F(s_0, \delta^t) = V(s_0, \delta^t) + H^{T-t}(s_0, \delta^t)$$

For the heuristic search to be optimal and complete, the function H must be *admissible*, i.e. an overestimate of the exact value of the optimal completion of policy vector δ^t:

$$\forall \Delta^{T-t} : H^{T-t}(s_0, \delta^t) \geq V(\Delta^{T-t} | s_0, \delta^t)$$

It is crucial for the algorithm to have an admissible heuristic function that is efficiently computable and as close as possible to the true value of the state (to maximize the amount of pruning). To describe the admissible heuristics used by the algorithm, we use the following notation:

- $P(s|s_0, \delta)$ is the probability of being in state s after executing the policy tree vector δ from s_0.
- $h^t(s)$ is an optimistic value function heuristic for the expected sum of rewards when executing the best vector of depth t policy trees from state s, i.e. $h^t(s) \geq V^{*t}(s)$.

Now the following class of heuristic functions can be defined:

$$H^{T-t}(s_0, \delta^t) = \sum_{s \in S} P(s|s_0, \delta^t) h^{T-t}(s)$$

Any function in this class simulates the situation where the real underlying state is revealed to the agents at time t after execution of policy vector δ^t. Intuitively, any such heuristic H is admissible, if h is admissible (for a detailed proof see [48]). Fortunately, when computing the heuristic estimate for the value of a policy vector of depth t and state s_0, the double exponential number of possible completions does not have to be computed. Instead, h only has to be computed for every possible state, thus $|S|$ times. This leads to major savings. Szer et al. [48] have presented three different heuristics for the MAA* algorithm:

1. **The MDP heuristic:** A strong simplification is to consider the underlying centralized MDP with remaining finite horizon $T - t$: $h^{T-t}(s) = V^{\text{MDP}}_{T-t}(s)$. Because solving an MDP only takes polynomial time, this is an efficient way to compute the heuristic, but leads to a huge overestimate.
2. **The POMDP heuristic:** Considering the underlying POMDP leads to a tighter value function heuristic and thus allows more pruning. Unfortunately, solving POMDPs is PSPACE-complete, thus this heuristic is also more complex to compute.
3. **Recursive MAA*:** Among the heuristics that simulate revealing the underlying system state, the tightest heuristic possible is given by the optimal value itself: $h^t(s) = V^{*t}(s)$. This value can be efficiently computed by applying MAA* recursively: $h^{T-t}(s) = MAA^{*T-t}(s)$. Calling MAA^{*T} invokes $|S|$ subcalls of MAA^{*T-1}. Note that this procedure is exponential in t as compared to a complete search, which is double exponential in t.

Obviously, there is a tradeoff between the tightness of the heuristic and the complexity of computing it. Due to the double exponential growth of the search tree, a tighter yet more complex heuristic has proven to be best in experimental results. So far, the MAA* algorithm using the recursive heuristic function is the only algorithm found to be able to solve the multi-agent tiger problem (as presented in Sect. 1.1.2) with 2 states, 2 agents, 3 actions and 2 observations up to horizon 4. MAA* runs out of time when computing horizon 5. To improve scalability, approximation techniques such as those presented in Sect. 4 must be used.

*3.1.3 Limitations of MAA**

Szer et al. [48] note that due to the double exponential dependence on the time horizon T, evaluating the search tree until depth $T-1$ is "easy," i.e. almost all of the computational effort is dedicated to the last level of the policy tree. Completely expanding a node at level $T-1$ and creating all its level T children takes a lot of time. However, if the value estimate of one of these children is as high as the value estimate of its parent (which is called a *tie*), the other children do not have to be considered. Thus, in the implementation of MAA*, the authors have used *incremental node expansion*. This means that only one child assignment of actions is constructed in each iteration and then the value of this child is computed. As long as some children need to be generated, the parent remains in an *open list*. The authors argue that this might lead to considerable savings in both memory and runtime.

An obstacle is that incremental node expansion only saves time when the value estimate of a node actually coincides with the value estimate of the best child: $F^T(s_0, \delta^{t-1}) = F^T(s_0, \delta^{*t})$. Even if this were the case, however, many children would have to be generated before the best is picked. So far, it is not known if there is a fast way to find the best child. Heuristics for finding the best child might exist, but good ones are likely to be problem-dependent. In fact, for this procedure to be effective, revealing the true system state to all agents at must lead to the same value estimate as before, regardless of the state. This seems very unlikely in decentralized problems. For example, consider the multi-agent tiger problem, where revealing the true system state would actually have a huge impact on the agent's expected value. If the true system state is revealed to the agents, the agents know the location of the tiger and thus they can open the door that leads to treasure with probability 1. In most problems, these ties will not occur and thus incremental node expansion will not be very effective. Obviously, heuristics that do not reveal the underlying system state do not suffer from this problem. But so far, no such heuristic has been proposed.

The low likelihood of ties is a significant drawback, limiting the applicability of MAA*. When at least one node is expanded at level $T-1$ with no ties, all of its children must be generated to find the optimal solution for horizon T. Completely expanding just one of the last search nodes, in going from level $T-1$ to level T, creates $(|A|^{(|\Omega|^{T-1})})^n$ new child nodes. Thus, in general, the MAA* algorithm can at best solve problems whose horizon is only 1 greater than those that can already be solved by naive brute force search. This conclusion is illustrated by the empirical results shown in Table 5.

As shown, the MAA* algorithm can solve the multi-agent tiger problem with horizon 4, whereas brute force search can only solve horizon 3. As already explained, however, MAA* evaluates more nodes for the horizon 4 problem than the brute force algorithm does for the horizon 3 problem. The same would be true in comparing horizon 4 with horizon 5. This

Table 5 Performance comparison: evaluated policy pairs

Horizon	Brute force search	Recursive MAA*
1	9	9
2	729	171
3	4,782,969	26,415
4	$2.06 \cdot 10^{14}$	344,400,183
5	$3.82 \cdot 10^{29}$	$>2.06 \cdot 10^{14}$

means that MAA* would have to evaluate more than $2.06 \cdot 10^{14}$ nodes to solve the horizon 5 problem, independent of the heuristic used.

Another recent research direction has examined weighted heuristics, i.e. $f(s) = g(s) + w \cdot h(s)$. If the weight w is greater than 1, the heuristic becomes non-admissible and the search is no longer optimal, but can find suboptimal solutions faster, as it goes deep into the search tree more quickly. The idea is that after suboptimal solutions have been found, large chunks of the search tree can be pruned, thus reducing significantly the size of the open list. If the open list becomes empty, the last solution generated is optimal. Because keeping a large open list in memory has an impact on the entire search process, this can lead to a considerable speedup. Unfortunately, with double exponential dependence on the time horizon, this approach does not yield considerable savings because at least one node at level $T-1$ must be fully expanded (if no tie occurs). Again, as this is already as much work as a brute force search would do for a problem with horizon $T-1$, the problem remains almost as hard to solve.

Note that the MAA* algorithm runs out of time before it runs out of memory, in contrast to the dynamic programming algorithm, which runs out of memory long before it runs out of time. Furthermore, the MAA* algorithm makes use of the common reward function of the agents but does not take into account at all that it is dealing with a DEC-POMDP problem. On the other hand, the dynamic programming algorithm does exploit the fact that it is dealing with a DEC-POMDP by pruning dominated strategies. A more promising approach seems to be combining heuristic search with dynamic programming, yielding a better and more sophisticated algorithm for finding approximate solutions for finite-horizon DEC-POMDPs. Such an approach is presented in Sect. 4.2.

3.2 An ε-optimal infinite-horizon algorithm

The solution techniques presented in the previous section do not work for infinite-horizon DEC-POMDPS because we cannot build infinitely long policy tress which would require an infinite amount of memory. Thus, a different representation for the policies has to be used that only uses a finite amount of memory. Note that while we no longer have a finite time horizon T, all definitions of infinite-horizon DEC-POMDPs need a discount factor $\gamma \in [0, 1)$.

Because infinite-horizon DEC-POMDPs are undecidable, one cannot hope for a truly optimal algorithm for this class. In 2005, Bernstein [5] presented a policy iteration algorithm for infinite-horizon DEC-POMDPs that converges to ε-optimal solutions in the limit. We outline this approach below.

Definition 46 (*Local finite-state controller*) A local finite-state controller for agent i is a tuple $\langle Q_i, \psi_i, \eta_i \rangle$, where:

- Q_i is a finite set of controller nodes.
- $\psi_i : Q_i \rightarrow \Delta A_i$ is a stochastic action selection function.
- $\eta_i : Q_i \times A_i \times O_i \rightarrow \Delta Q_i$ is a stochastic transition function.

Definition 47 (*Independent joint controller*) A set of local finite-state controllers, one for each agent, determines the conditional distribution $P(\vec{a}, \vec{q}\,' | \vec{q}, \vec{o})$, referred to as an independent joint controller.

This way of representing a joint policy is called an *independent* joint controller because there is no direct correlation between the agents. Each agent's local policy depends only on its own memory state. As Bernstein has demonstrated, however, some form of correlation between the agents might be beneficial.

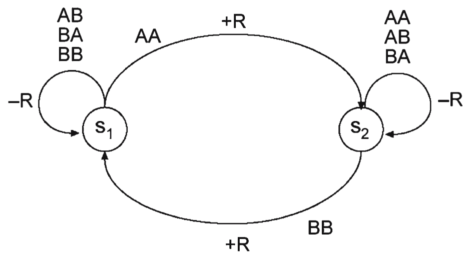


Fig. 10 A DEC-POMDP whose optimal memory-bounded joint policy requires correlation (courtesy of Daniel Bernstein)

3.2.1 The utility of correlation

Figure 10 shows a DEC-POMDP that illustrates the importance of correlation. In this example, there are two states, two agents, two actions per agent (A and B) and one observation. Because there is just one observation, the agents cannot distinguish between the two different states. Below, two memoryless policies for this problem are compared:

- Because the agents do not know the state they are in, each deterministic policy will be arbitrarily bad in the long run. If both agents can randomize independently, the best they could do is choose either action A or B according to a uniform distribution in each step. With an expected reward of $-R/2$ per time step this yields an expected long-term reward of $-R/2(1-\gamma)$.
- If the agents can correlate their policies using a common source of randomness, the best policy would be to execute the joint action AA with probability $1/2$ and BB with probability $1/2$. This leads to an expected long term reward of 0. Thus, the difference between correlated and uncorrelated strategies can be made arbitrarily large by increasing R.

This example shows the need for correlation when representing joint policies with bounded memory. The importance of stochastic correlation was also proposed by Peshkin et al. [35]. They presented an algorithm that uses gradient descent in policy space to find locally optimal solutions for multi-agent problems. While that technique was focused on multi-agent learning rather than planning, it has provided important insights regarding the usefulness of correlation.

3.2.2 Correlated joint controllers

To allow for correlation, Bernstein et al. [7] introduce a *correlation device* in the form of an additional finite-state machine. The correlation device is some random process that does not have any access to the observations available to the agents. All agents have access to extra signals from the device in each time step, but cannot exchange information via the correlation device.

Definition 48 (*Correlation device*) A correlation device is a tuple $\langle C, \psi \rangle$, where:

- C is a finite set of states.
- $\psi : C \rightarrow \Delta C$ is a state transition function.

At each time step, the device makes a transition and all agents observe the new state.

Now, the definition of a local controller has to be extended to consider the correlation device. The action taken by each agent and the controller state transitions become dependent on the input signal c. Thus, the local controller for agent i is a conditional distribution of the form $P(a_i, q_i' | c, q_i, o_i)$.

Definition 49 (*Correlated joint controller*) A correlation device together with the local controllers for each agent form a joint conditional distribution $P(c', \vec{a}, \vec{q}' | c, \vec{q}, \vec{o})$, called a correlated joint controller.

The value of a correlated joint controller can then be computed by solving a set of linear equations, one for each $s \in S$, $\vec{q} \in \vec{Q}$, and $c \in C$:

$$V(s, \vec{q}, c) = \sum_{\vec{a}} P(\vec{a}|c, \vec{q}) \left[R(s, \vec{a}) \right.$$
$$\left. + \gamma \sum_{s', \vec{o}, \vec{q}', c'} P(s', \vec{o}|s, \vec{a}) P(\vec{q}'|c, \vec{q}, \vec{a}, \vec{o}) P(c'|c) V(s', \vec{q}', c') \right]$$

3.2.3 Policy iteration for infinite-horizon DEC-POMDPs

As there are infinitely many possible observation sequences for the infinite-horizon case, any finite set of states of a correlated joint controller may be insufficient to produce an optimal solution. To guarantee ε-convergence, it is necessary to increase the number of controller states successively. This can be accomplished using an *exhaustive backup*, which we have already introduced for the finite-horizon DP algorithm. Here, instead of growing policy trees in a top-down manner, we iteratively grow the local controllers, for all agents at once, leaving the correlation device unchanged.

To formalize this process, let Q_i^t denote the set of controller nodes for agent i after iteration t. For each possible one-step policy, a new controller node is added. Thus, for each agent i, $|A_i||Q_i|^{|\Omega_i|}$ nodes are added to the controller. In the finite-horizon algorithm, the exhaustive backup was followed by a pruning step, eliminating dominated policy trees. Here, an analogous procedure can be used [5].

Definition 50 (*Value-preserving transformation*) Given two correlated joint controllers C and D with node sets $\vec{Q}$ and $\vec{R}$ respectively, we say that changing controller C to D is a value-preserving transformation if there exist mappings $f_i : Q_i \rightarrow \Delta R_i$ for each agent i and $f_c : Q_c \rightarrow \Delta R_c$ such that:

$$V(s, \vec{q}) \leq \sum_{\vec{r}} P(\vec{r}|q) V(s, \vec{r}) \quad \forall s \in S, \vec{q} \in \vec{Q}$$

The goal of a value-preserving transformation is to reduce the size of a controller without decreasing its value, or to improve the value without changing the size. In general, reducing the size of the controller is necessary between exhaustive backup steps because those steps increase the size of the controller in a double exponential manner. Bernstein [5] formulated several such transformations that can be implemented efficiently using linear programming.

Similar to the finite-horizon case, exhaustive backups are interleaved with value-preserving transformations. However, for the infinite-horizon case we need an explicit stopping criterion. Because there is no Bellman residual for testing convergence as in single agent MDPs, it is necessary to use the discount factor γ and the number of iterations to define a simpler ε-convergence test. Let $|R_{max}|$ denote the largest absolute value of a one-step reward in the DEC-POMDP. Then the algorithm terminates after iteration t if $\gamma^{t+1} \cdot |R_{max}|/(1-\gamma) \leq \varepsilon$. Intuitively, the algorithm exploits the fact that due to discounting, at some point the future rewards collected are negligible. The complete procedure is sketched in Algorithm 1.

Algorithm 1: Policy iteration for infinite-horizon DEC-POMDPs

```
input  : DEC-POMDP problem, random correlated joint controller, convergence parameter $\varepsilon$
output: A correlated joint controller that is $\varepsilon$-optimal for all states
begin
    $t \leftarrow 0$
    while $\gamma^{t+1} \cdot |R_{max}|/(1-\gamma) > \varepsilon$ do
        $t \leftarrow t+1$
        Evaluate the correlated joint controller by solving a system of linear equations
        Perform an exhaustive backup to add nodes to the local controllers
        Perform value-preserving transformations on the controller
    return correlated joint controller
end
```

The convergence of the algorithm is characterized by the following theorem [5].

Theorem 10 *For any ε, policy iteration returns a correlated joint controller that is ε-optimal for all initial states in a finite number of iterations.*

As with optimal algorithms for finite-horizon DEC-POMDPs, the policy iteration algorithm for infinite-horizon problems is mainly of theoretical significance. In practice, the value-preserving transformations cannot reduce the size of the controllers sufficiently to continue executing the algorithm until the convergence criterion is met. The algorithm runs out of memory after a few iterations even for small problems. However, several approximate techniques for infinite-horizon DEC-POMDPs have been developed that restrict the size of each controller and optimize value with a bounded amount of memory. We present two such approaches in Sect. 5.

4 Approximate algorithms for finite-horizon problems

Due to the high computational complexity of DEC-POMDPs, developing approximation techniques rather than exact algorithms seems more fruitful in practice. Unfortunately, as was shown by Rabinovich et al. [40], even ε-optimal history-dependent joint policies are still NEXP-hard to find in both DEC-POMDPs and DEC-MDPs. Thus, when looking for approximate solutions that are computationally tractable, the global value has to be sacrificed to lower the complexity and to handle larger problems. At this point, a small ambiguity in the term "approximation" must be discussed. In the field of theoretical computer science, an "approximation algorithm" generally denotes an algorithm that provides some provable guarantees on the solution quality. These *performance guarantees* can for example be relative guarantees (the approximation will be no worse than a factor of c of the optimal solution) or absolute guarantees (the approximation will be within ε of the optimal solution). Approximation algorithms are normally used to find solutions for NP-hard optimization problems, and a polynomial running time is generally required.

In the AI community, the term "approximation" is used in a less strict sense. In this paper, we differentiate between *optimal algorithms* (that always find the optimal solution), *ε-optimal algorithms* (that guarantee a performance within ε of the optimal solution) and *approximate algorithms* (i.e. heuristics that have no performance guarantees at all). In this section we present approximate algorithms that do not guarantee any bounds on the solution quality with regard to the optimal solution. Furthermore, some of them do not even guarantee a polynomial worst case running time, which is still acceptable in light of the NEXP-completeness result. Due to these missing guarantees, however, it is difficult to compare the different approximation techniques. A few benchmark problems have been established and can be used for initial comparison of running time and solution value. In the long run, a whole suite of problems has to be established such that good comparable benchmark tests can be performed. Comparable competitions already exist for the field of centralized deterministic planning algorithms but are still to be established for the field of decentralized planning under uncertainty. In Sect. 7 we will summarize the different approaches presented in this section and try a comparison where possible.

4.1 Overview of recent approximation techniques

One approach for finding approximate algorithms for decentralized problems is to generalize existing approximate algorithms for POMDPs. Unfortunately, most of those algorithms use a compact representation of the belief state space. So far, no such compact representation has been found for DEC-POMDPs, mostly because an agent's belief is not only dependent on its own local view of the problem, but also on the other agents. Thus, generalizing existing algorithms for POMDPs is not straightforward.

In 2003, Nair et al. [26] presented a class of algorithms, called "Joint Equilibrium-Based Search for Policies" (JESP), that do not search for the globally optimal solution of a DEC-POMDP, but instead aim for local optimality. The best algorithm, DP-JESP, incorporates three different ideas. First, the policy of each agent is modified while keeping the policies of the others fixed (a similar idea is presented in Sect. 5.1). Second, dynamic programming is used to iteratively construct policies. This is analogous to the idea presented in Sect. 3.1.1. Third, and most notably, only reachable belief states of the DEC-POMDP are considered for policy construction. This leads to a significant improvement, because there is only an exponential number of different belief states for one agent as opposed to the double exponential number of possible joint policies. The algorithm is able to solve small problems such as the multi-agent tiger problem up to horizon 7. Thus, it is an improvement over the exact algorithms (which work only up to horizon 4), although it does not scale well for larger problem sizes.

Emery-Montemerlo et al. [15] take a game-theoretic approach, modeling the problem as a POSG with common payoffs. In contrast to the other algorithms discussed in this paper, this is an online algorithm which interleaves planning and execution. The POSG is approximated as a series of smaller Bayesian games. In each of these simple games, the agents only have to decide on the next action to take based on a one-step lookahead. The future reward after taking one action is estimated using heuristics such as Q_{MDP}, which turns the remaining problem into a fully-observable problem. Similar to the dynamic programming algorithm proposed by Nair et al. [26], the algorithm solves the resulting Bayesian games using an alternating-maximization algorithm, where the best response strategy of one agent is computed while the strategies of the other agents are held fixed. In each iteration, before the agents select an action they match their history of local observations and actions with one of the types in their local type space found by solving the Bayesian game. To limit memory requirements for remembering the history of actions and observations, low-probability histories are pruned. These simplifying assumptions (1. treating the problem as a series of smaller games with one-step lookahead, 2. computing future rewards using a heuristic, and 3. using the alternating-maximization algorithm) lead to some obvious value loss. However, the algorithm is able to solve substantially larger problems compared to Nair et al. [26], largely due to the fact that planning and execution are interleaved in an intelligent way.

Within the scope of this study, it is not possible to cover all recently developed approximate algorithms for DEC-POMDPs in detail. Instead, we limit ourselves to the details of the Improved Memory-Bounded Dynamic Programming (IMBDP) algorithm, as it has outperformed all recently proposed approximation methods. IMBDP can solve problems with horizons that are multiple orders of magnitude larger than what was previously possible, while achieving the same or better solution value.

4.2 Improved memory-bounded dynamic programming (IMBDP)

In 2007, we introduced the original MBDP [42] as well as the improved MBDP algorithm [43]. This section is based on these two papers, summarizing the main ideas.

The method is based on the dynamic programming algorithm, presented in Sect. 3.1.1. As we have seen, even if pruning techniques are used (elimination of dominated policies) in the bottom-up construction of the policy trees, the number of trees still grows too quickly and the algorithm runs out of memory. An analysis of the construction process reveals that most policies kept in memory are useless because a policy tree can only be eliminated if it is dominated for *every* belief state. For many DEC-POMDPs, however, only a small subset of the belief space is actually reachable. Furthermore, a policy tree can only be eliminated if it is dominated for every possible belief over the other agents' policies. Obviously, during the construction process, these other agents also maintain a large set of policy trees that will eventually prove to be useless.

Unfortunately, with a bottom-up approach, these drawbacks of the pruning process cannot be avoided. Before the algorithm reaches the roots of the final policy trees, it cannot predict which beliefs about the state and about the other agents' policies will eventually be useful. Thus, considering the entire set of reachable belief states does not improve scalability by much because that set still grows exponentially (see [47]). However, top-down approaches such as the MAA* algorithm presented in Sect. 3.1.2 do not need to know all of the reachable belief states to compute joint policies. This observation leads to the idea of combining the bottom-up and top-down approaches: using top-down heuristics to identify relevant belief states for which the dynamic programming algorithm can then evaluate the bottom-up policy trees and select the best joint policy.

4.2.1 Top-down heuristics and the MBDP algorithm

Even though the agents do not have access to a central belief state during policy execution, it can still be used to evaluate bottom-up policy trees computed by the DP algorithm. We found that policy trees that are good relative to a centralized belief state are often also good candidates for the decentralized policy. Obviously, a belief state that corresponds to the optimal joint policy is not available during the construction process. Fortunately, however, a set of belief states can be computed using multiple top-down heuristics—efficient algorithms that find useful top-down policies. In [42], we describe a portfolio of heuristics suitable for DEC-POMDPs. In practice, the MDP-heuristic (revealing the underlying system state after each time step) and a random-policy heuristic have proven useful. The usefulness of the heuristics and, more importantly, the computed belief states are highly dependent on the specific problem. But once the algorithm has computed a complete solution, we have a joint policy that definitely leads to relevant belief states when used as a heuristic. This is exactly the idea of *Recursive MBDP*. The algorithm can be applied recursively with an arbitrary recursion-depth.

In the original MBDP algorithm the policy trees for each agent are constructed incrementally using the bottom-up DP approach. To avoid the double exponential blow-up, the parameter *maxTrees* is chosen such that a full backup with this number of trees does not exceed the available memory. Every iteration of the algorithm consists of the following steps. First, a full backup of the policies from the last iteration is performed. This creates policy tree sets of the size $|A||maxTrees|^{|O|}$. Next, top-down heuristics are chosen from the portfolio and used to compute a set of belief states. Then, the best policy tree pairs for these belief states are added to the new sets of policy trees. Finally, after the Tth backup, the best joint policy tree for the start distribution is returned.

4.2.2 Motivation for improved MBDP algorithm

Although the original MBDP algorithm has linear time and space complexity with respect to the horizon T, it is still exponential in the size of the observation space. This is a severe limitation when tackling just slightly larger problems than the multi-agent broadcast channel problem or the multi-agent tiger problem. Even for a small problem with 2 actions and 5 observations, setting $maxTrees = 5$ would be prohibitive, because $(2 \cdot 5^5)^2 = 39,062,500$ policy tree pairs would have to be evaluated.

Consequently, tackling the size of the observation set is crucial. The key observation is that considering the whole set of possible observations in every belief state and for every possible horizon is neither useful nor even actually necessary. Consider for example robots navigating a building. After turning away from the entrance door, the robots are highly unlikely to observe that same door in the next time step. In general, depending on the belief state and the action choice, only a very limited set of observations might be possible. This is the main idea behind the Improved Memory-Bounded Dynamic Programming (IMBDP) algorithm [43].

4.2.3 The improvement: partial backups

In order to select a limited set of useful observations, we propose a technique similar to that used for selecting promising bottom-up policy trees. As before, in step one the algorithm first identifies a set of belief states using the top-down heuristics. For every identified belief state b^t at horizon t, the best joint policy is added. Additionally, in step two the set of most likely observations for every agent is identified, bounded by a pre-defined number of observations *maxObs*. More specifically, for the most likely belief state b^{t-1} for horizon $t-1$ and joint action $\vec{a}$ prescribed by the heuristic, the probability $Pr(\vec{o}) = \sum_s b^{t-1}(s) \cdot O(\vec{o}|\vec{a}, s)$ is

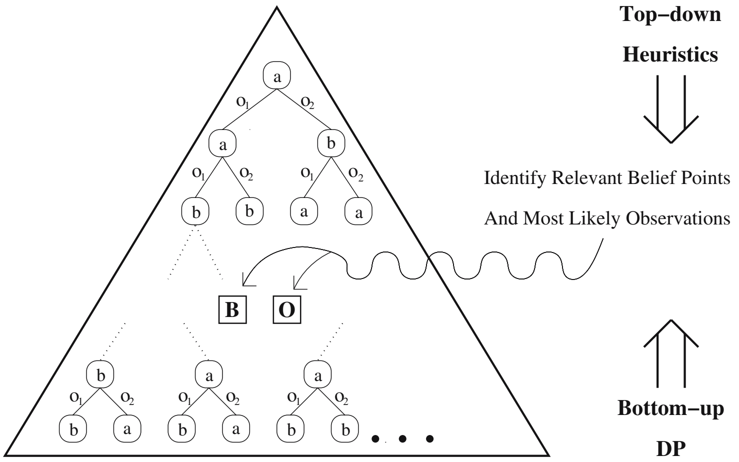


Fig. 11 The construction of a single policy tree, combining top-down and bottom-up approaches

computed for each joint observation $\vec{o}$. Then all joint observations $\vec{o}$ are ranked according to $Pr(\vec{o})$. This ranking is used to select the k most likely observations for each agent separately, where $k = maxObs$. For the joint policy trees selected in step one the algorithm then performs a *partial backup* with just k observations for every agent, leading to a feasible number of policy trees (with missing branches). Note that although only k observations are used for the partial backup of every agent's policy trees, implicitly k^n joint observations are taken into consideration. After the partial backups the resulting trees are filled up with the missing observation branches using local search (hill climbing: for every missing observation check which of the available subnodes of the next level will lead to the highest expected utility for the given belief state). This concludes a single iteration. Figure 11 illustrates the main idea of the improved MBDP algorithm, combining the two approximation techniques.

The algorithm can be applied to DEC-POMDPs with an arbitrary number of agents, but to simplify notation, the description shown in Algorithm 2 is for two agents, i and j.

Algorithm 2: The improved MBDP algorithm

```
begin
    $maxTrees \leftarrow$ max number of trees before backup
    $maxObs \leftarrow$ max number of observations for backup
    $T \leftarrow$ horizon of the DEC-POMDP
    $H \leftarrow$ pre-compute heuristic policies for each $h \in H$
    $Q_i^1, Q_j^1 \leftarrow$ initialize all 1-step policy trees
    for $t = 1$ to $T - 1$ do
        $Sel_i^t, Sel_j^t \leftarrow$ empty
        for $k = 1$ to $maxTrees$ do
            choose $h \in H$ and generate belief state $b^{T-t}$
            foreach $q_i \in Q_i^t, q_j \in Q_j^t$ do
                evaluate pair $(q_i, q_j)$ with respect to $b^{T-t}$
            add best policy trees to $Sel_i^t$ and $Sel_j^t$
            delete these policy trees from $Q_i^t$ and $Q_j^t$
        choose $h \in H$ and generate belief state $b^{T-t-1}$
        $O_i, O_j \leftarrow$ $maxObs$ most likely obs. for $h(b^{T-t-1})$
        $Q_i^{t+1}, Q_j^{t+1} \leftarrow$ partialBackup($Sel_i^t, Sel_j^t, O_i, O_j$)
        fill up $Q_i^{t+1}$ and $Q_j^{t+1}$ with missing observations
        improve $Q_i^{t+1}$ and $Q_j^{t+1}$ via hill climbing
    select best joint policy tree $\delta^T$ for $b^0$ from $\{Q_i^T, Q_j^T\}$
    return $\delta^T$
end
```

4.2.4 Theoretical properties

The quality of the joint policy trees produced by the improved MBDP algorithm depends on the maximum number of trees per horizon level *maxTrees* and the maximum number of observations allowed per partial backup *maxObs*. In [43], we prove the following theorem.

Theorem 11 *For a given maxObs $< |O|$ and any belief state $b \in \Delta S$, the error of improved MBDP due to partial backups with selected observations on a horizon-T problem, $|V(\delta^T, b) - V(\delta^T_{maxObs}, b)|$, is bounded by:*

$$\mu_T = T^2 \cdot (1 - \varepsilon) \cdot (R_{\max} - R_{\min})$$

where ε denotes the minimal sum of joint observation probabilities ever used in a partial backup (see [43] for a detailed definition).

The following corollary is an immediate consequence.

Corollary 3 *Increasing maxObs* $\rightarrow |O|$ *the error bound* $\mu_T = |V(\delta^T, b) - V(\delta^T_{maxObs}, b)|$ *is strictly decreasing and reaches 0 in the limit.*

Here, we can only describe the main ideas of the MBDP algorithm and its successor, IMBDP. Please see [42,43] for a more detailed presentation, theoretical proofs, and experimental results.

5 Approximate algorithms for infinite-horizon problems

The ε-optimal dynamic programming algorithm for infinite-horizon problems presented in Sect. 3.2 suffers from the problem that it runs out of memory very quickly, just as do the optimal algorithms for the finite-horizon case. One might think that the various approximation techniques proposed for finite-horizon DEC-POMDPs could be generalized to the infinite-horizon case. In fact, while some of the ideas carry over, in general the process is not straightforward. The challenges in each case are significantly different, requiring very different solution representation techniques to work well. Thus, new approximate techniques have had to be developed for the infinite-horizon case. We present three representatives in this section.

5.1 Bounded policy iteration for decentralized POMDPs

We have already discussed the finite-state controller representation for policies in the infinite-horizon case. Poupart and Boutiliers [37] used this representation to develop a POMDP algorithm that uses a bounded amount of memory. This technique, called *bounded policy iteration*, was extended by Bernstein et al. [7] to DEC-POMDPs. The approach optimizes the value of controllers of some fixed size. This implies that for general problems the agents will sometimes be in the same controller state for different observation sequences and thus that the resulting policy can be suboptimal. The great advantage, however, is the memory-bounded way of representing the policy, which allows for solving larger problems.

When using a limited amount of memory for policy representation, there are many different ways to allocate that memory. It turns out that the number of nodes used for each controller is crucial for the resulting policy. The bounded policy iteration (BPI) algorithm works with a fixed number of nodes for each controller, including the local controllers for each agent and the correlation device. (Note that to be useful at all, a correlation device must have at least two nodes.)

When the BPI algorithm begins, an arbitrary distribution for the controller nodes is chosen. Obviously, this may correspond to an arbitrarily bad policy. The correlated joint controller is then improved iteratively via *bounded backup*, until a local maximum is reached. In each iteration of the algorithm, a node q_i of one controller (or of the correlation device) is chosen. The improvement for a local controller works by searching for new parameters of conditional distribution $P(a_i, q_i'|c, q_i, o_i)$, for each $o_i \in O_i$. This search assumes that the new controller will be used from the second step on, and can be performed by solving a linear program ensuring that the new parameters improve the value of the correlated joint controller for each system state, each local controller state and each state of the correlation device. If an improvement can be found, the new parameters are set and the next iteration starts. The improvement of the correlation device works in an analogous way.

Bernstein et al. [7] proved that bounded backup, applied to a local controller or a correlation device, always produces a correlated joint controller with value at least as high as before, for any initial state distribution. This results in a monotonic improvement in value in every iteration. Unfortunately, the algorithm only leads to local optima because only one controller node is improved at a time while all the others are held fixed. Thus the algorithm reaches suboptimal Nash equilibria. So far, no linear program is known that can update more than one controller at a time.

Experimental results with this algorithm support the intuition that larger numbers of controller nodes lead to higher values. As explained, with a fixed number of controller nodes this algorithm can only compute approximate solutions in general. The interesting features of the algorithm are its use of a bounded amount of memory, a monotonic improvement of value in each iteration, the possibility for correlated randomness and a polynomial running time (assuming a fixed number of agents).

5.2 A nonlinear optimization approach to find memory-bounded controllers

5.2.1 Limitations of bounded policy iteration

The BPI algorithm suffers from a common limitation of approximate algorithms—getting stuck in local optima. In particular, BPI improves only one controller node at a time with a one-step look-ahead, which leads to short-term improvements but does not generally lead to optimal controllers. Even though heuristics may be used to get unstuck from local maxima, the approach is somewhat limited.

The limitations of BPI have been recently illustrated by Amato et al. [2], using a simple example. Figure 12 shows a simple single-agent POMDP for which BPI fails to find an optimal controller, which obviously illustrates the limitations of the multi-agent version as well. In this example, there are two states, two actions and one observation. Thus, the observation does not provide any information about the underlying state. The state transitions deterministically if action 1 is taken in state 1 and if action 2 is taken in state 2, and otherwise remains the same. A state alternation results in a positive reward of R and remaining in the same state results in a negative reward of $-R$. Assuming a uniform initial state distribution, the optimal policy is to choose each action with probability 0.5. This can be realized using a stochastic finite-state controller with just one node whose value would be 0.

The BPI algorithm starts with an arbitrary initial controller. If this controller happens to be deterministic and chooses either action, say action 1, BPI will not converge to the optimal controller. The value of this initial controller is $R - \gamma R/(1-\gamma)$ in state 1 and $-R/(1-\gamma)$ in state 2. For $\gamma > 0.5$ this value is negative and thus less than the value of the optimal controller. A one-step look-ahead assigning any positive probability to action 2 raises the value for state 2 but lowers it for state 1. Because in every iteration the algorithm only accepts

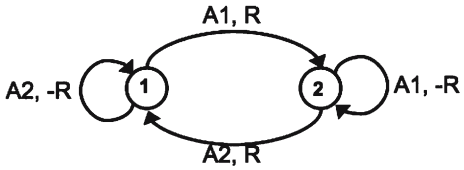


Fig. 12 Simple POMDP for which BPI fails to find an optimal controller (courtesy of Christopher Amato)

a new controller if the value increases or remains the same for all nodes and all states, BPI will not make any changes in this situation and thus get stuck with the suboptimal policy.

5.2.2 Optimal fixed-size controller for POMDPs/DEC-POMDPs

In 2006, Amato et al. [2] presented a more sophisticated approach to finding optimal fixed-size controllers for POMDPs. Unlike BPI, their algorithm improves and evaluates the controller in one phase. Furthermore, their approach allows optimizing the controller for a specific start distribution over states. In addition to the original linear program they also represent the value of each node in each state $V(q, s)$ as a variable. To ensure correct values, nonlinear constraints representing the Bellman equations are added to the optimization. After reducing the representation complexity the new optimization results in a quadratically constrained linear program (QCLP)—a mathematical program that must have a linear objective function, but may contain quadratic terms in the constraints. QCLPs are more difficult than linear programs, but simpler than general non-linear programs. A large number of algorithms have been developed that can exploit the additional structure. The full description of the QCLP is given in Table 6. Variable $x(q', a, q, o)$ represents $P(q', a|q, o)$, variable $y(q, s)$ represents $V(q, s)$, q_0 is the initial controller node and o_k is an arbitrary fixed observation.

It is easy to show that an optimal solution of this QCLP results in an optimal stochastic controller for the given size and initial state distribution. In this respect, the formulation of the optimization problem as a quadratically constrained linear program is a significant improvement over the original BPI algorithm, which often gets stuck in local maxima. Additionally, the possibility of exploiting an initial state distribution leads to further value improvement.

Unfortunately, the resulting problem is nonconvex and thus may have multiple local and global maxima. A wide range of algorithms is available that can solve nonconvex problems efficiently but that only guarantee local optimality, as they use various approximation methods to solve the QCLP. Only sometimes can the globally optimal solution be found. Thus, even with the QCLP formulation of the optimization problem, finding the optimal solution for a POMDP is hard and often not feasible. Nevertheless, experimental results by Amato et al. [2] show that in most cases near-optimal controllers with values higher than those achieved by BPI can be found by using standard nonlinear optimization techniques. In particular, for large POMDPs, very small controllers achieving a high value could be found in less time

Table 6 The quadratically constrained linear program for finding the optimal fixed-size controller

For variables: $x(q', a, q, o)$ and $y(q, s)$

Maximize $\sum_{s} b_0(s) y(q_0, s)$

Given the Bellman constraints

$\forall q, s \quad y(q, s)$

$$= \sum_{a} \left[\left(\sum_{q'} x(q', a, q, o) \right) R(s, a) + \gamma \sum_{s'} P(s'|s, a) \sum_{o} O(o|s', a) \sum_{q'} x(q', a, q, o) y(q', s') \right]$$

And probability constraints

$\forall q, o \quad \sum_{q', a} x(q', a, q, o) = 1$

$\forall q, o, a \quad \sum_{q'} x(q', a, q, o) = \sum_{q'} x(q', a, q, o_k)$

$\forall q', a, q, o \quad x(q', a, q, o) \geq 0$

than with BPI. The authors note that there is an interesting relationship between the nonlinear formulation of the problem and the performance of the optimization algorithms used. The detailed analysis of this relationship is the subject of ongoing research.

Recently, Amato et al. have extended this approach to DEC-POMDPs [1]. Now, the optimization requires that multiple controllers be taken into consideration at once. For example, for two agents, in addition to the original linear program, the values of each node (of each controller) in each state, $V(q_1, q_2, s)$, are added as a variable. Again, to ensure correct values, nonlinear constraints representing the Bellman equations are added to the optimization. Additional constraints are needed to guarantee that the controllers are independent, namely that the probability of each action and transition depends only on the information available to that agent. The resulting nonlinear program (NLP) is no longer a QCLP, but it is of a form that can be solved efficiently using existing solvers. As in the POMDP case, it is easy to show that an optimal solution of this NLP provides optimal fixed-size controllers for a given DEC-POMDP. Obviously, trying to find the optimal solution for the NLP is not simple; existing solvers guarantee only local optimality. Nevertheless, experimental results show that using nonlinear optimization techniques leads to better controllers than those produced by BPI. Thus, the NLP approach makes more efficient use of memory and provides better theoretical guarantees than other approximation methods.

In this context it is also worth mentioning the algorithm developed by Szer and Charpillet [46]. They present an optimal best-first search algorithm for solving infinite-horizon DEC-POMDPs. In their work, policies are also represented as finite-state controllers and search is done in the space of controllers. Here optimality is guaranteed with respect to a given controller size. But this algorithm only considers deterministic controllers as opposed to the stochastic controllers used by Bernstein et al. [7] and Amato et al. [1,2]. As has been shown before, stochasticity is crucial when using limited memory. Moreover, as shown in Sect. 3.2.1, correlated randomness improves the performance of finite state controllers. Thus, an approach that only considers deterministic controllers leads to suboptimal solutions with respect to the whole space of possible (stochastic) controllers.

5.3 Approximate dynamic programming

In 2003, de Farias and van Roy [12] developed an algorithm for approximate dynamic programming for centralized problems based on linear programming, using the notion of a Q function. The algorithm developed by Cogill et al. [11] generalizes this algorithm by extending it to decentralized problems. Their approach is the first to use centralized solution techniques to tackle a decentralized problem. To overcome the complexity of a decentralized control problem, they incorporate human knowledge about the specific problem to transform it into a set of easier sub-problems, which are still dependent on each other. Obviously, this mapping cannot always be value-preserving. Thus, by applying the transformation step, they approximate the *optimal decentralized solution*; in fact they aim to approximate the *optimal centralized solution*, whose value serves as an upper bound for the value of the decentralized solution. The difference in value between the decentralized and the centralized solution is bounded, depending on the approximation error. In a second step, they apply approximate linear programming to solve the decentralized problem.

Their algorithm uses the notion of a Q function to perform linear programming. The definitions necessary to introduce the linear program for centralized problems are first introduced in Sect. 5.3.1. Sect. 5.3.2 then presents the approximate linear programming algorithm for decentralized problems. In presenting the formal models and algorithms, some notation from [11] has been changed to facilitate a comparison with the DEC-POMDP model.

5.3.1 Dynamic programming for centralized problems

The framework used for centralized problems is equivalent to an MDP. It is defined by:

- A finite state space S.
- A finite set of actions A.
- Taking action a in state s incurs a cost $g(s, a)$.
- After taking action a in state s, a state transitions occurs leading to the new state $y \in S$ with probability $P(y|s, a)$.
- A static state-feedback policy is defined by $\mu : S \rightarrow A$.

The goal is to minimize the expected total discounted cost

$$J^{\mu}(s) = E\left[\sum_{t=0}^{\infty} \alpha^t g(s_t, \mu(s_t))|s_0 = s\right]$$

where α is a discount factor with $0 \leq \alpha < 1$. The minimum cost-to-go J^* is unique and satisfies Bellman's equation:

$$J^*(s) = \min_{a \in A}\left[g(s, a) + \alpha \sum_{y \in S} P(y|s, a) J^*(y)\right]$$

Bellman's equation can also be expressed directly in terms of a Q function as

$$Q^*(s, a) = g(s, a) + \alpha \sum_{y \in S} P(y|s, a)(\min_{w} Q^*(y, w))$$

A Q function defines the cost for a state-action pair. Q^* denotes the unique solution and μ^* denotes the corresponding optimal centralized policy. Solving the centralized problem means solving Bellman's Equation, i.e. finding the policy that has minimal cost for all state-action pairs. This can be done by solving the linear program shown in Table 7.

5.3.2 Approximate dynamic programming for decentralized problems

For the decentralized problem a specific structure is imposed on the Q function. To define this structure, the framework must first be extended to the multi-agent case. For a problem with n agents this yields:

- The action space $\mathbf{A} = A_1 \times \cdots \times A_n$, where A_i is the set of actions for agent i.
- The system state is described by a collection of state variables and the corresponding state space is $\mathbf{S} = S_1 \times \cdots \times S_m$.

For a centralized policy, each action a_i would be based on the entire state $s = (s_1, \ldots, s_m)$. For a decentralized policy, each action a_i only depends on some specified subset of the state

Table 7 Linear program for the centralized problem

maximize:	$\sum_{s \in S} \sum_{a \in A} Q(s, a)$	
subject to:	$Q(s, a) \leq g(s, a)$ $+\alpha \sum_{y \in S} P(y|s, a) J(y)$	for all $s \in S, a \in A$
	$J(s) \leq Q(s, a)$	for all $s \in S, a \in A$

variables $s_1, \ldots, s_m$. This makes the problem decentralized, because an agent does not have all information available when making its decisions. This corresponds to the notion of a mapping from observation sequences to actions in a DEC-POMDP, where each agent has to make decisions based on incomplete information.

If the problem is decentralized, the dependencies of the actions on the state variables are described by an *information structure*. To formalize this, the following definitions are needed:

- The set of variables used to decide action a_i is denoted $I_i = \{j | a_i \text{ depends on } s_j\}$.
- The Cartesian product of the spaces of the variables used to decide action a_i is denoted as $\mathbf{S}_i = \bigotimes_{j \in I_i} S_j$.
- A decentralized policy is a set of functions $\mu_i : \mathbf{S}_i \to A_i$ for $i = 1, \ldots, n$.

Recall that Q^* denotes the unique solution to the Bellman equation in the centralized case and that μ^* denotes the optimal centralized policy. To decompose the problem, a policy $\mu(s) = \text{argmin}_a \hat{Q}(s, a)$ is considered, where $\hat{Q}$ is a function of the form $\hat{Q} = \sum_{i=1}^{n} Q_i$ and each $Q_i : \mathbf{S}_i \times A_i \to \Re$. In the argument of Q_i there is only one decision variable, namely a_i. Thus, minimizing $\hat{Q}$ is equivalent to minimizing each Q_i, i.e. choosing a to be $\text{argmin}_a \hat{Q}(s, a)$ is equivalent to choosing each a_i to be $\text{argmin}_{a_i} Q_i(s, a_i)$. An appropriately structured $\hat{Q}$, which shall approximate Q^*, leads to a decentralized problem which can then be solved by the approximate linear programming algorithm shown in Table 8.

Note that the decentralized problem is not easier to solve than the corresponding centralized one. In fact, it is much harder, because the centralized problem is an MDP (which is P-complete) and the decentralized problem is equivalent to a DEC-MDP (which is NEXP-complete). But as the algorithm is only an approximation, it ignores some of the interdependencies of the true decentralized problem. It turns out that finding suboptimal policies for the decentralized problem using this algorithm is computationally easier than computing the optimal centralized solution.

In their paper, Cogill et al. [11] prove two theorems which relate the approximation error between Q^* and $\hat{Q}$ to the difference in the achieved value, i.e. the difference between J^* and $\hat{J}$. This is relevant for the goal of computing a decentralized solution that approximates the centralized one as closely as possible. The authors note that the weights $\omega(s)$ in the linear program have a huge impact on the quality of the policy obtained. It remains an open question how to find good weights (or the best weights) in general, and what bounds on the difference in value can be established for these weights.

In particular, it would be interesting to see how large the performance difference between an *optimal decentralized policy* and the *approximated decentralized policy* can be. The au-

Table 8 Approximate linear programming algorithm for the decentralized problem

```
1. For any information structure, let $\hat{Q} = \sum_{i=1}^{n} Q_i$ and $\hat{J} = \sum_{i=1}^{n} J_i$, where $Q_i : \mathbf{S}_i \times A_i \to \Re$ and
   $J_i : \mathbf{S}_i \to \Re$
2. For arbitrary positive values $\omega(s)$, solve the following linear program
       maximize:    $\sum_{s \in S} \sum_{a \in A} \omega(s) \hat{Q}(s, a)$
       subject to:  $\hat{Q}(s, a) \leq g(s, a) + \alpha \sum_{y \in S} P(y|s, a) \hat{J}(y)$    for all $s \in S, a \in A$
                    $\hat{J}(s) \leq \hat{Q}(s, a)$                                              for all $s \in S, a \in A$
3. Let $\mu(s) = \text{argmin}_a \hat{Q}(s, a)$. This policy is decentralized with the desired information structure
```

thors do not compare the empirical results of their algorithm with any other decentralized algorithm. A comparison between the bounded policy iteration algorithm and this algorithm would thus be useful, in order to better evaluate the two different approaches.

The authors illustrate their approach with an example of load balancing among a collection of queues. The goal is to keep the load at each queue as small as possible. Jobs arrive and are processed with a certain probability <1. If one of the queues overflows, a penalty incurs. The queues are networked, which enables them to pass a job to one of the neighboring queues at each time step. The problem is decentralized, as every queue only observes its own backlog as well as the backlogs of its immediate neighbors. The local policies for each queue are strategies concerning whether to pass a job to one of the neighbors or not, depending on the current backlog status.

For this problem, the Q_i's are defined such that each queue's action is only dependent on the current backlog of itself and its immediate neighbors at a given time step. This decentralization of the problem, however, is suboptimal, because an optimal decentralized solution would have to consider entire histories of observations. This is illustrated by an easy example with five queues, as shown in Fig. 13.

The arcs indicate incoming jobs and jobs that are passed from one queue to its neighbor. As one can see, at time step 1, queue number 1 passes one job to queue number 2. Furthermore, new jobs for queues 1, 2 and 3 arrive. Looking at queue number 3 at time step 2, it would be reasonable to pass a job to one of its neighbors. Considering only the current backlog status of its neighbors results in a tie, as both neighbors have a backlog of 2. Thus, the algorithm—only considering the current state—is unable to compute the optimal action. Obviously, if the algorithm would consider the history of observations, it could infer that in the last time step queue number 1 had passed a job to queue number 2 and that it is thus best to pass the job to queue number 4 instead of queue number 2. The lack of such consideration for history illustrates one weakness of the algorithm.

In fact, it is possible to incorporate history into the algorithm, by increasing the state space. A more important question, however, is whether approximating a decentralized problem by trying to approximate the optimal centralized solution is a reasonable approach. The problem is that the imposed information structure, i.e. the new Q_i's, might not make any sense for certain problems. In such cases, solving the linear program with those Q_i's might yield a result that is far from the optimal decentralized solution. For the load balancing problem, the definition of a Q_i that considers a local view of the problem still makes sense, i.e. optimizing the backlog locally does somehow optimize the global reward. However, this obviously relies on a special structure of the problem, namely that the coupling between non-adjacent queues is

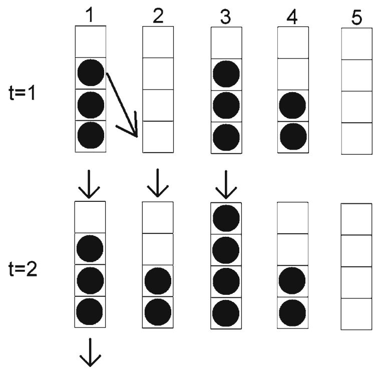


Fig. 13 An example load balancing problem, illustrating the importance of observation history

weak. Thus, the decentralized version of the problem is very close to the centralized one, and it makes sense to approximate the centralized solution. For other problems, where the local view of an agent says little if anything about the global state of the problem, the approach will not generally be suitable. In such cases, the value achieved by the algorithm can be arbitrarily bad.

6 Decomposition techniques

As shown in Sect. 2.3.2, even decentralized problems with joint full observability (DEC-MDPs) are NEXP-complete. The high complexity is due to the various ways multiple agents might influence each other's states, observations and rewards. The approximation technique considered in Sect. 5.3 exploits specific structures in the decentralized process and lowers the running time of a corresponding algorithm by reducing/ignoring some of the interdependencies found in the problem under consideration. A similar approach has been developed by Goldman and Zilberstein [20]. The important observation is that many decentralized problems have some structure influencing the "level of decentralization," as discussed in Sect. 2.5.1. For example, the special case of a goal-oriented DEC-MDP with independent transitions and observations, with a single global goal state and uniform costs is P-complete [19]. Obviously, not all decentralized problems fall into such special classes, but many only violate the membership conditions slightly. Thus, the idea of decomposing a decentralized problem into multiple single-agent MDPs can serve as a useful basis for finding a feasible approximate algorithm. To deal with the decentralized nature of the problem, Goldman and Zilberstein [20] introduce the notion of *communication-based* decomposition mechanisms, which involve a communication policy that determines when agents solving individual MDPs should synchronize their knowledge.

The technical details of this approach are quite involved and are beyond the scope of this study. However, because it offers yet another approach to decentralized decision making, we summarize the main ideas below. In Sect. 6.1, communication-based decomposition mechanisms are introduced. Sect. 6.2 presents the formal framework of decentralized semi-Markov decision problems that serves as a basis for the associated algorithms. Because finding an optimal decomposition mechanism for the general case is very hard, the focus turns to more specific sub-classes of decomposition mechanisms in Sect. 6.3, leading to more efficient algorithms.

6.1 Communication-based decomposition mechanisms

As discussed in Sect. 2.5.1, even problems with independent transitions and observations are still hard to solve, because the reward can be dependent on multiple agents and might not be decomposable into several independent reward functions. Furthermore, even goal-oriented DEC-MDPs do not necessarily decompose into multiple independent local goals; however, there remains a distinct sub-class with polynomial complexity. Thus, Goldman and Zilberstein [20] concentrate on finite-horizon goal-oriented problems with independent transitions and observations and direct communication. The general idea is to let the agents operate separately for certain periods of time until they need/want to re-synchronize. Obviously, these periods of separation may sacrifice some possible value, but the benefit is in the lower computational complexity of the single agent problems. Obviously, this is only beneficial if the whole decomposition algorithm also has a low complexity. Furthermore, such a decomposition mechanism is required to be *complete* in that there must exist a communication policy that guarantees the agents will reach one of the global goals whenever possible.

The agents' separate phases of execution are defined using temporally abstracted sequences of actions. These sequences, called options, include a deterministic single-agent policy, terminal actions that correspond to communication messages, and a set of initiation states:

Definition 51 (*Options*) An option for an agent i is given by a tuple $opt_i = \langle \pi : S_i \times \tau \rightarrow A_i \cup \Sigma,\ I \subseteq S_i \rangle$, where

- S_i is the set of local states of agent i.
- τ is the number of time steps since the process was started.
- A_i is the set of agent i's actions.
- Σ is the set of communication messages.
- I is the initiation set for the option, i.e. the set of states in which the agent can start the option.

Note that once an agent executes one of its options, it follows its local policy deterministically until it terminates the option by sending a communication message to all other agents. The only way that an option can be terminated earlier is when (a) the time limit T is reached or (b) another agent sends a communication message. A joint exchange of messages between agents is assumed, making it possible to reveal the global state of the system to all agents. As discussed earlier, it is normally too costly for agents to communicate after every time step. The challenge for the decomposition mechanism is to compute the states at which the agents can act separately, and when it is necessary to re-synchronize.

Definition 52 (*Communication-based decomposition mechanism*) A communication-based decomposition mechanism CDM is a function from any global state of the decentralized problem to multiple single agent policies, i.e. $CDM : S \rightarrow (Opt_1, Opt_2, \ldots, Opt_n)$.

Obviously, such a mechanism generally only approximates the optimal decentralized joint policy. To find the best mechanism, one must search over all possible mechanisms. For two agents, this requires search over all possible pairs of local single-agent policies and communication policies. The best mechanism then is the best approximation (that can be found through decomposition) for the decentralized problem. As we see in the next section, computing optimal mechanisms can be very complex. Thus, Sect. 6.3 turns to an easier sub-class of mechanisms that are computationally tractable.

6.2 Decentralized semi-Markov decision problems

The formal framework has to be modified slightly to work with options instead of basic agent actions. Based on an underlying DEC-MDP with direct communication, a goal-oriented decentralized SMDP with direct communication can be defined. The crucial point is the computation of the sets of individual and temporally abstracted actions that are used to build the options for each agent. Computing a decomposition mechanism for a DEC-MDP can be framed as a semi-Markov decision problem in the following way:

Definition 53 (*GO-DEC-SMDP-COM*) A factored finite-horizon goal-oriented DEC-SMDP-COM over an underlying goal-oriented DEC-MDP-COM, DMC, is a tuple $\langle DMC, Opt_1, Opt_2, \ldots, Opt_n,\ P^N,\ R^N \rangle$, where:

- DMC is a standard DEC-MDP-COM with components $S,\ \Sigma,\ C_\Sigma,\ \{\Omega_i\},\ P,\ O,\ T$.
- Opt_i is the set of actions available to agent i. It comprises the possible options for agent i, terminating necessarily with a communication act: $opt_i = \langle \pi : S_i \times \tau \rightarrow A_i \cup \Sigma,\ I \subseteq S_i \rangle$.

- $P^N(s', t + N|s, t, opt_1, \ldots, opt_n)$ is the probability that the system reaches state s' after N time units $(t + N \leq T)$, when at least one option terminates (necessarily with a communication act).
- $R^N(s, t, opt_1, opt_2, \ldots, opt_n, s', t + N)$ is the expected reward obtained by the system N time steps after agents start options $opt_1, \ldots, opt_n$ in state s at time t, when at least one terminates its option with a communication act (resulting in termination of the other agents' options).

The course of events in the semi-Markov decision process has changed compared to the DEC-MDP. At the beginning of the execution of an option, all agents have full observability of the state (as opposed to joint fully observability). Their local policy is a mapping from global states to options. After the first option of one agent terminates, all agents communicate, leading to joint exchange of messages and thus full observability of the global state. Since the global state is revealed every time the agents have to choose new options, the choice of local policies no longer depends on histories of observations. In fact, only the global state and system time are important: $\delta_i : S \times \tau \to Opt_i$. A joint policy is a tuple of local policies, one for each agent. Intuitively, optimally solving a decentralized semi-Markov decision problem with temporally abstracted actions (i.e. solving for an optimal mechanism) is equivalent to solving a multi-agent MDP (MMDP, see Definition 40), as shown by Goldman and Zilberstein [20].

Theorem 12 *A GO-DEC-SMDP-COM is equivalent to a multi-agent MDP.*

This may seem counterintuitive, since multi-agent MDPs are P-complete as opposed to DEC-MDPs, which are NEXP-complete. To clarify, the input to this problem now includes a very high number of possible options for each agent. Each option can be represented as a tree, with depth at most T and branching factor at most S. If A is the set of domain actions available to the agent, this results in more than $|A|^{|S|^T}$ possible assignments of primitive domain actions, and thus a double exponential number of possible options. Solving the problem requires computing the best option for each agent for each global state. This results in the same double exponential complexity as does the original problem. Naturally, any algorithm solving this problem optimally needs double exponential time in the worst case (see also the multi-step backup policy-iteration algorithm in [20]). As the number of possible options leads to such high complexity, a restricted class of options is now considered. Using these *goal-oriented options* significantly reduces the size of the search space and lowers the overall complexity.

6.3 GO-DEC-SMDP-COM with local goal-oriented behavior

The basic idea presented in this section is to find a mapping from global goals to local goals. That is, with local goal-oriented behavior, there is no need to consider the complete set of possible options for each agent. However, not every goal-oriented DEC-MDP automatically decomposes into local goal-oriented behavior for each agent. The question of how to generate local goals from a given global goal is beyond this study. We simply assume here that a set of local goal states is provided for each agent i, denoted $\hat{G}_i$, where $\hat{G}_i \subseteq S_i$. This set must include the components of the global goal states in G. But in addition, it may also include other local states from S_i, which will serve as temporary local goals. Given these local goal states, goal-oriented options can be defined as follows:

Definition 54 (*Goal-oriented option*) Let $\pi_{opt_i}(s_i, t)$ be the policy of an option opt_i. Let T be some time limit. If there exists an optimal policy $\delta : S_i \to A_i$ that minimizes the cost to some goal state component g_i then opt_i is goal-oriented if for every state s_i and $t < T$,

$\pi_{opt_i}(s_i, t) = \delta(s_i)$ and $\pi_{opt_i}(s_i, T) \in \Sigma$. It is assumed that the agents have the capability of executing a NOP action, which incurs zero cost and has no transition effect, when they are in some goal state component g_i.

Now, the decomposition mechanism has to map each global state to a tuple of goal-oriented options and a period of time k. After the agents execute their options for k time steps they re-synchronize, even if they have not reached their local goals. After exchanging messages, the global state is revealed and new goal-oriented options can be assigned.

The optimal decomposition mechanism now has to find the best mapping from global states to goal-oriented options. But once a local goal for agent i has been assigned, the policy π_{g_i} can be computed independently by optimally solving its local process $MDP_i = (S_i, P_i, R_i, \hat{G}_i, T)$, which can be done in polynomial time. Intuitively, this makes possible polynomial time decomposition mechanisms. No longer do all possible options have to be considered—instead, the mechanism can focus on the possible local goal states. Indeed, Goldman and Zilberstein [20] present an algorithm based on backup policy-iteration that computes the optimal mechanism based on local goal-oriented behavior and that is polynomial in the size of the state space. See their paper for the details of the algorithm and the corresponding convergence and complexity proofs.

Obviously, an approach based on local goal-oriented behavior is not suitable for every kind of problem. For some problems, the dependence between the agents is so strong (possibly only through the reward function) that every local goal assignment can lead to arbitrarily bad solutions. Note that this is the same problem as with the approximate dynamic programming approach discussed in Sect. 5.3. For other problems, however, these techniques make much more sense, and can actually make solving some larger problems possible. In their paper, Goldman and Zilberstein [20] choose the meeting under uncertainty example to illustrate their approach. In this domain it is reasonable to assign local goals (which are meeting points on a grid) and to communicate from time to time (to get updated information about where the other agent is). For these kind of problems, their algorithms prove to be suitable. Additional algorithms are presented for situations in which the mapping from global states to single agent behaviors is already given. In that case, the problem reduces to computing an optimal communication policy.

7 Summary and comparison of algorithms

This section summarizes the characteristics of all the multi-agent planning algorithms described in this paper. It illustrates the different approaches taken over the past 5 years and points out their advantages and disadvantages. First, the algorithms for solving finite-horizon DEC-POMDPs are presented. Then, Sect. 7.2 presents all the algorithms for the infinite-horizon case.

7.1 Algorithms for finite-horizon DEC-POMDPs

Algorithm	Dynamic Programming for DEC-POMDPs
Authors	Bernstein et al. [8]
Solution quality	Optimal
Solution technique	Bottom-up dynamic programming and iterated elimination of dominated strategies
Advantages	Can exploit specific DEC-POMDP structure of the problem
Disadvantages	Cannot exploit initial state, runs out of memory quickly

Algorithm	Multi-agent A*: heuristic search for DEC-POMDPs
Authors	Szer et al.[48]
Solution quality	Optimal
Solution technique	Top-down A*-search in the space of joint policies
Advantages	Can exploit an initial state, could use domain-specific knowledge for the heuristic function (when available)
Disadvantages	Cannot exploit specific DEC-POMDP structure, can at best solve problems with horizon 1 greater than problems that can be solved via brute force search (independent of heuristic)

Algorithm	Joint equilibrium-based search for policies
Authors	Nair et al. [26]
Solution quality	Approximate
Solution technique	Computation of reachable belief states, dynamic programming, improving policy of one agent while holding the others fixed
Advantages	Avoids unnecessary computation by only considering reachable belief states
Disadvantages	Suboptimal solutions: finds only local optima

Algorithm	Approximate solutions for POSGs with common payoffs
Authors	Emery-Montemerlo et al. [15]
Solution quality	Approximate
Solution technique	Online algorithm: usage of a series of smaller Bayesian games to approximate the POSG, matches local agent observations with type profile space in an online way
Advantages	Interleaves planning with executing, which reduces computational complexity, scales moderately well with problem size
Disadvantages	Suboptimal solutions: one-step lookahead reduces solution quality

Algorithm	(Improved) Memory-bounded dynamic programming
Authors	Seuken and Zilberstein [42,43]
Solution quality	Approximate
Solution technique	Combines top-down heuristics with bottom-up DP, uses heuristics to identify a bounded set of relevant belief points and observations
Advantages	Keeps a bounded number of policy trees in memory and as a result has only polynomial complexity in the time horizon
Disadvantages	Suboptimal solutions: trade-off between number of policy trees and solution quality

Algorithm	Communication-based decomposition mechanism
Authors	Goldman and Zilberstein [20]
Solution quality	Approximate
Solution technique	Decomposition into multiple MDPs and computation of a communication policy
Advantages	Exploits structure and takes into account the “level of decentralization,” scales moderately well for some problems
Disadvantages	Limited to those problems that bear special structure

Generally, comparing the performance of different algorithms requires at least that they be tested on the same machine, and implemented in the same programming language. This issue, however, is less of a problem for DEC-POMDPs because the double exponential time complexity dominates any linear speed-ups due to different implementations or machines. In fact, so far the challenge has not so much been finding the fastest or best algorithm for a given problem, but finding one that can solve even small benchmark problems for horizons larger than 3 or 4. Except for MBDP, all algorithms, whether optimal or approximate, cannot solve standard benchmark problems for horizons larger than 10. Table 9 presents a performance comparison of all off-line planning algorithms for finite-horizon DEC-POMDPs for which experimental results on the multi-agent tiger problem have been reported. Shown are the solution values achieved for each horizon. A “?” indicates that the algorithm

Table 9 Performance comparison of the different algorithms on the multi-agent tiger problem

Horizon	DP	MAA*	JESP	MBDP
2	−4.00	−4.00	−4.00	−4.00
3	5.19	5.19	−6.00	5.19
4	–	4.80	?	4.80
5	–	–	?	5.38
6	–	–	?	9.91
7	–	–	?	9.67
8	–	–	–	9.42
9	–	–	–	12.57
10	–	–	–	13.49
1,000	–	–	–	819.01
100,000	–	–	–	78252.18

could compute a solution for this horizon, but the achieved value was not reported in the corresponding paper. A "–" indicates that the algorithm ran out of time or memory for this horizon.

7.2 Algorithms for Infinite-Horizon DEC-POMDPs

Algorithm	Policy iteration for infinite-horizon DEC-POMDPs
Authors	Bernstein [5]
Solution quality	ε-Optimal
Solution technique	Representation of policies using correlated joint controllers, exhaustive backups and value-preserving transformation
Advantages	Converges to the optimal solution in the limit
Disadvantages	Computationally intractable due to memory requirements

Algorithm	Bounded policy iteration for DEC-POMDPs
Authors	Bernstein et al. [7]
Solution quality	Approximate
Solution technique	Representation of policies using correlated joint controllers, improving controller of one agent while holding the others fixed
Advantages	Limited memory fixed ahead of time, polynomial time complexity per iteration, allows for correlated randomness, monotonic value improvement for all initial state distributions
Disadvantages	Suboptimal solutions: gets stuck in local optima

Algorithm	An optimal best-first search algorithm for solving infinite-horizon DEC-POMDPs
Authors	Szer and Charpillet [46]
Solution quality	Approximate
Solution technique	Representation of policies using deterministic finite state-controllers, A*-search in the space of controllers
Advantages	Finds optimal deterministic finite state controller for a given size
Disadvantages	Suboptimal solutions: deterministic finite state controllers achieve significantly lower solution values than stochastic controllers

Algorithm	Optimal fixed-size control of decentralized POMDPs
Authors	Amato et al. [1]
Solution quality	Approximate
Solution technique	Non-linear programming, representation of policies using stochastic finite state controllers
Advantages	Defines optimal stochastic finite state controller for a given size, optimal controller can be found for some problems
Disadvantages	NLPs are more complex than normal LPs, optimal solution cannot always be found

Algorithm	Approximate dynamic programming for decentralized systems
Authors	Cogill et al. [11]
Solution quality	Approximate
Solution technique	Linear programming using Q-functions
Advantages	Algorithm guarantees a bound on value-loss depending on the approximation error
Disadvantages	Algorithms seeks to approximate the optimal centralized solution, which can lead away from the optimal decentralized solution

8 Conclusion and open research questions

We have examined the problem of sequential decision making for multiple cooperative agents in decentralized settings. These problems are much harder than single-agent problems (MDP, POMDP) or their centralized multi-agent variants (MMDP, MPOMDP) because in a decentralized setting each individual agent may have different partial knowledge about the other agents and about the current state of the world. Five different formal frameworks to model the problem were presented: DEC-POMDPs, MTDPs, DEC-POMDP-COMs, COM-MTDPs, and I-POMDPs. The relationships between these models have presented several open questions. In Sect. 2, we proved formally that the first four models are all equivalent—they are all NEXP-complete. It is widely believed that optimal algorithms for these models require double exponential time in the worst case. The orthogonal I-POMDP model is more expressive in that it allows the modeling of non-cooperative agents, but it leads to non-computable agent functions in the general case. We showed that even its approximate variant, namely the finitely-nested I-POMDP model, likely requires double exponential time in the worst case.

These complexity results illustrate the difficulty of the problems under consideration. There is little hope that realistic problem instances can be solved optimally. This has led to investigation of sub-classes of lower complexity, which we described in Sect. 2.5. Their complexity ranges from NEXP to P, and thus some of the most restricted classes are computationally tractable. Sect. 3 illustrates how quickly a naive algorithm for the complete problem becomes intractable even for very small problems. Nevertheless, two non-trivial general algorithms have been developed: Dynamic Programming for DEC-POMDPs and the heuristic search algorithm MAA*. Our analysis of the limitations of heuristic search showed that better heuristics, better ways of enumerating children, and weighted heuristics all have limited utility, as they cannot yield sufficient savings in runtime to allow for much deeper search.

Both of the optimal algorithms introduced in Sect. 3 quickly run out of time or memory, a problem that is addressed by the latest approximation techniques, presented in Sect. 4. While some of these algorithms establish certain performance bounds, in general only local optimality can be guaranteed. Value loss can therefore be arbitrarily large, depending on the specific problem. So far, memory-bounded dynamic programming (MBDP) is the only approximate algorithm for finite-horizon problems that can solve problems that are significantly larger than those solved by the optimal methods. The technique of keeping a bounded number of policy tress in memory while doing DP updates seems promising; for the current benchmark problems its running time scales well even for time horizons beyond 100,000 steps. Its scalability for larger problem instances remains to be examined. Another open research question is how to manage the trade-off between the number of policy trees and solution quality.

Solution techniques for infinite-horizon problems also present some interesting research questions. While the nonlinear programming approach is able to specify the optimal fixed-size controller, solving the resulting NLP to find the optimal solution is generally infeasible. For the approximate dynamic programming approach, we discussed the instances for which this approach might be suitable and why it fails for other problem classes. A similar discussion of decomposition techniques is presented in Sect. 6. This approach is designed specifically for those special classes of problems in which local (goal-oriented) behavior of the agents is suitable.

We have seen that some restricted sub-classes of DEC-POMDPs have worst-case complexity significantly lower than NEXP. For example, goal-oriented DEC-MDPs with independent transitions and observations, a single global goal, and uniform costs are P-complete. Thus, if the problem under consideration has this special structure, larger instances can be solved efficiently. For less restricted classes, many open research questions still remain. For example, the complexity of goal-oriented DEC-POMDPs with independent transitions and observations is unknown. Another open question relates to the impact of correlated randomness in finite-state controllers for infinite-horizon DEC-POMDPs. As we have seen, correlation can have a large impact on the value achieved by memory-bounded algorithms in decentralized settings. However, a full examination of the utility of correlation, in particular how to manage the tradeoff between controller size and the size of the correlation device remains to be completed.

The recently developed approximate algorithms for DEC-POMDPs demonstrate different degrees of effectiveness and scalability. Some approaches are still limited to toy problems. Others exhibit better scalability, producing good solutions for significantly large problems. The challenge for future research is to fully understand where complexity can be lowered without sacrificing value too much, and how to use the available memory most efficiently in dealing with complicated problem instances.

Acknowledgments We thank Daniel Bernstein, Marek Petrik, Martin Allen, Christopher Amato, Raphen Becker, Abdel-Illah Mouaddib, Daniel Szer, François Charpillet, Victor Lesser, and Claudia Goldman for helpful discussions. We also thank the anonymous reviewers for their helful comments and suggestions. Furthermore, we thank the authors of all the papers on the different models and algorithms presented in this study. This work was supported in part by the Air Force Office of Scientific Research (Grant No. FA9550-05-1-0254) and by the National Science Foundation (Grant No. 0535061). Any opinions, findings, conclusions or recommendations expressed in this manuscript are those of the authors and do not reflect the views of the US government.

References

1. Amato, C., Bernstein, D., & Zilberstein, S. (2007). Optimizing memory-bounded controllers for decentralized POMDP. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence (UAI)*. Vancouver, Canada, July 2007.
2. Amato, C., Bernstein, D. S., & Zilberstein, S. (2007). Solving POMDPs using quadratically constrained linear programs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 2418–2424). Hyderabad, India, January 2007.
3. Aström, K. J. (1965). Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications, 10*, 174–205.
4. Becker, R., Zilberstein, S., Lesser, V., & Goldman, C. V. (2004). Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research (JAIR), 22*, 423–455.
5. Bernstein, D. S. (2005). Complexity analysis and optimal algorithms for decentralized decision making. PhD thesis, Department of Computer Science, University of Massachusetts Amherst, Amherst, MA.
6. Bernstein, D. S., Givan, R., Immerman, N., & Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research, 27*(4), 819–840.
7. Bernstein, D. S., Hansen, E. A., & Zilberstein, S. (2005). Bounded policy iteration for decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1287–1292). Edinburgh, Scotland, July 2005.
8. Bernstein, D. S., Zilberstein, S., & Immerman, N. (2000). The complexity of decentralized control of Markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI)* (pp. 32–37). Stanford, California, June 2000.

9. Beynier, A., & Mouaddib, A. (2006). An iterative algorithm for solving constrained decentralized Markov decision processes. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI)* (pp. 1089–1094). Boston, MA, July 2006.
10. Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *Proceedings of the Conference on Theoretical Aspects of Rationality and Knowledge* (pp. 195–210). De Zeeuwse Stromen, The Netherlands.
11. Cogill, R., Rotkowitz, M., van Roy, B., & Lall, S. (2004). An approximate dynamic programming approach to decentralized control of stochastic systems. In *Proceedings of the Allerton Conference on Communication, Control, and Computing* (pp. 1040–1049). Urbana-Champaign, IL, 2004.
12. de Farias, D. P., & van Roy, B. (2003). The linear programming approach to approximate dynamic programming. *Operations Research, 51*(6), 850–865.
13. Doshi, P., & Gmytrasiewicz, P. J. (2005). A particle filtering based approach to approximating interactive POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)* (pp. 969–974). Pittsburg, Pennsylvania, July 2005.
14. Doshi, P., & Gmytrasiewicz, P. J. (2005). Approximating state estimation in multiagent settings using particle filters. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)* (pp. 320–327). Utrecht, Netherlands, July 2005.
15. Emery-Montemerlo, R., Gordon, G., Schneider, J., & Thrun, S. (2004). Approximate solutions for partially observable stochastic games with common payoffs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)* (pp. 136–143). New York, NY, July 2004.
16. Gmytrasiewicz, P. J., & Doshi, P. (2005). A framework for sequential planning in multiagent settings. *Journal of Artificial Intelligence Research JAIR), 24*, 49–79.
17. Goldman, C. V., Allen, M., & Zilberstein, S. (2007). Learning to communicate in a decentralized environment. *Autonomous Agents and Multi-Agent Systems, 15(1)*, 47–90.
18. Goldman, C. V., & Zilberstein, S. (2003). Optimizing information exchange in cooperative multi agent systems. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)* (pp. 137–144). Melbourne, Australia, July 2003.
19. Goldman, C. V., & Zilberstein, S. (2004). Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research (JAIR), 22*, 143–174.
20. Goldman, C. V., & Zilberstein, S. (2004). Goal-oriented DEC-MDPs with direct communication. Technical Report 04-44, Department of Computer Science, University of Massachusetts Amherst.
21. Hansen, E. A. (1998). Finite-memory control of partially observable systems. PhD thesis, Department of Computer Science, University of Massachuetts Amherst.
22. Hansen, E. A., Bernstein, D. S., & Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI)* (pp. 709–715). San Jose, California, July 2004.
23. Kaelbling, L. P., Littmann, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence, 101*(2), 99–134.
24. Kalai, E., & Lehrer, E. (1993). Rational learning leads to nash equilibrium. *Econometrica, 1*, 1231–1240.
25. Madani, O., Hanks, S., & Condon, A. (2003). On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence, 147*(1–2), 5–34.
26. Nair, R., Pynadath, D., Yokoo, M., Tambe, M., & Marsella, S. (2003). Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 705–711). Acapulco, Mexico, August 2003.
27. Nair, R., Varakantham, P., Tambe, M., & Yokoo, M. (2005). Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)* (pp. 133–139). Pittsburgh, Pennsylvania, July 2005.
28. Nisan, N., Roughgarden, T., Tardos, E., & Vazirani, V. V. (Eds.). (2007). *Algorithmic Game Theory*. New York, NY: Cambridge University Press.
29. Ooi, J. M., & Wornell, G. W. (1996). Decentralized control of a multiple access broadcast channel: Performance bounds. In *Proceedings of the Thirty-Fifth Conference on Decision and Control* (pp. 293–298). Kobe, Japan, December 1996.
30. Osborne, M. J., & Rubinstein, A. (1994). *A course in game theory*. Cambridge, MA: The MIT Press.
31. Papadimitriou, C. H. (1994). *Computational complexity*. Reading, MA: Addison Wesley.
32. Papadimitriou, C. H., & Tsitsiklis, J. N. (1986). Intractable problems in control theory. *SIAM Journal on Control and Optimization, 24*(4), 639–654.
33. Papadimitriou, C. H., & Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research, 12*(3), 441–450.

34. Parkes, D. C. (2008). Computational mechanism design. In *Lecture notes of Tutorials at 10th Conf. on Theoretical Aspectsof Rationality and Knowledge (TARK-05).* Institute of Mathematical Sciences, University of Singapore (to appear).
35. Peshkin, L., Kim, K.-E., Meuleau, N., & Kaelbling, L. P. (2000). Learning to cooperate via policy search. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI)* (pp. 489–496). Stanford, CA, July 2000.
36. Petrik, M., & Zilberstein, S. (2007). Anytime coordination using separable bilinear programs. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI)* (pp. 750–755). Vancouver, Canada, July 2007.
37. Poupart, P., & Boutilier, C. (2003). Bounded finite state controllers. In *Advances in Neural Information Processing Systems 16 (NIPS)*. Vancouver, Canada, December 2003.
38. Puterman, M. L. (1994). *Markov decision processes: discrete stochastic dynamic programming*. New York, NY: Wiley.
39. Pynadath, D. V., & Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research (JAIR), 16*, 389–423.
40. Rabinovich, Z., Goldman, C. V., & Rosenschein, J. S. (2003). The complexity of multiagent systems: The price of silence. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)* (pp. 1102–1103). Melbourne, Australia, 2003.
41. Russell, S., & Norvig, P. (2003). *Artificial intelligence: A modern approach*. Upper Saddle River, NJ: Prentice Hall.
42. Seuken, S., & Zilberstein, S. (2007). Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 2009–2015). Hyderabad, India, January 2007.
43. Seuken, S., & Zilberstein, S. (2007). Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI)*. Vancouver, Canada, July 2007.
44. Shen, J., Becker, R., & Lesser, V. (2006). Agent interaction in distributed MDPs and its implications on complexity. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. Hakodate, Japan, May 2006.
45. Suzuki, I., & Yamashita, M. (1999). Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing, 28*(4), 1347–1363.
46. Szer, D., & Charpillet, F. (2005). An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs. In *Proceedings of the Sixteenth European Conference on Machine Learning (ECML)* (pp. 389–399). Porto, Portugal, October 2005.
47. Szer, D., & Charpillet, F. (2006). Point-based dynamic programming for DEC-POMDPs. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI)* (pp. 1233–1238). Boston, MA, July 2006.
48. Szer, D., Charpillet, F., & Zilberstein, S. (2005). MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI)* (pp. 576–583). Edinburgh, Scotland, July 2005.
49. Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research (JAIR), 7*, 83–124.
50. Washington, R., Golden, K., Bresina, J., Smith, D. E., Anderson, C., & Smith, T. (1999). Autonomous rovers for Mars exploration. In *Proceedings of the IEEE Aerospace Conference* (pp. 237–251). Snowmass, CO, March 1999.
51. Zilberstein, S., Washington, R.,Bernstein, D. S., & Mouaddib, A.-I. (2002). Decision-theoretic control of planetary rovers. In Beetz, M., Guibas, L., Hertzberg, J., Ghallab, M., & Pollack, M.E. (Eds.), *Advances in Plan-Based Control of Robotic Agents*, Lecture Notes in Computer Science (Vol. 2466, pp. 270–289). Berlin, Germany: Springer.