Formal Models and Algorithms for Decentralized Control of Multiple Agents

Technical Report UM-CS-2005-068

Sven Seuken, Shlomo Zilberstein
Computer Science Department
University of Massachusetts
Amherst, MA 01003-9264
{seuken, shlomo}@cs.umass.edu

December 2005

Abstract

Over the last five years, the AI community has shown considerable interest in decentralized control of multiple decision makers or "agents". 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 control problems because each decision maker may have different partial information about the overall situation. Decentralized decision problems have been shown to be 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. One objective of this study is to provide a comprehensive overview of these results. The other objective is to compare and contrast the existing frameworks and to provide a deeper understanding of their relationship to each other, their merits, and their strengths and weaknesses. While we focus in this study on cooperative systems, we do point out important connections with game-theoretic approaches. We analyze five different formal frameworks, two different optimal algorithms, as well as six approximation techniques. The study 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 Control, Cooperative Agents, Multi-Agent Planning, Decision Theory, Formal Models.

Contents

1 Introduction

For over 50 years, researchers in the fields of Artificial Intelligence and Operations Research have been working on the problem of decision making for intelligent agents. The *Markov decision process* (MDP) framework has been proven to be useful for centralized sequential decision making under uncertainty in fully observable stochastic environments ([33], [36, chap. 17]). If an agent does not fully observe the environment and has to base its decisions on partial information, the MDP framework is no longer sufficient. In the 1960's, Astrom [3] introduced partially observable MDPs (POMDPs) to account for imperfect observations. In the 1990's, researchers in the AI community have adopted the POMDP framework and since then, a lot of progress in the development of practical algorithms has been made. For an extensive coverage on POMDPs see Hansen [21] and for an overview of the latest algorithms see Feng and Zilberstein [15].

An even more general problem results when two or more agents have to jointly control a system. If each agent has its own observation function but the agents must work together to optimize a joint reward function, this problem is called **decentralized control of a partially observable stochastic system**. In the last 5 years, different formal models for this problem have been proposed and interesting complexity results could be shown.

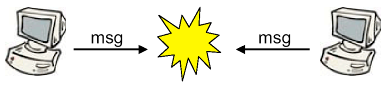
The decentralized partially observable MDP (DEC-POMDP) framework is one way to model this problem. In 2000, it has been shown by Bernstein et al. [6] that finite-horizon DEC-POMDPs are NEXP-complete. Thus, decentralized control of multiple agents is significantly harder than single agent control and provably intractable. Due to these complexity results, optimal algorithms have mostly theoretical significance and only little use in practice. Therefore, in the last few years, different approaches to approximate the optimal solution have been developed. Some problems contain certain structure that is exploitable and sometimes leads to less complex but still interesting sub-classes of the general problem. Different algorithms for these special problems have been introduced and proven to have much better running time than algorithms for the general problem. To achieve better scalability for the complete problem, researchers have recently focused on the development of feasible approximation techniques. Although some approximate algorithms can solve significantly larger problems, 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 [43], load balancing for decentralized queues [10], coordinated helicopter flights [34, 41], multi-access broadcast channels [27] and sensor network management [26]. In all of the aforementioned problems, multiple decision makers are jointly controlling a process, but they cannot share all of their information every time step. Thus centralized models of decision making are not suitable for these problems. To illustrate this, three problems that have been widely used by researchers in this area are described below.

1.1.1 Multi-Access Broadcast Channel

The first example is an idealized model of a multi-access broadcast channel (adapted from Ooi and Wornell [27]). 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 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 certain beliefs about the outcome of their actions. Figure 1 illustrates the general problem setup. Experimental results can be found in [5].



- States: who has a message to send?
- Actions: send or don't send
- Reward: +1 for successful broadcast
  0 if collision or channel not used
- Observations: was there a collision? (noisy)

Figure 1: Multi-Access Broadcast Channel (Courtesy of Daniel Bernstein).

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] in 1998. A multi-agent version of the tiger problem was introduced by Nair et al. [25] in 2003. It involves two agents standing behind 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 instead of opening one of the doors, the agents can both 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 get 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 or penalty, the problem starts over again. The problem is illustrated in Figure 2.

Figure 2: Multi-Agent Tiger Problem.

1.1.3 Meeting Under Uncertainty

This more realistic example was first presented by Bernstein et al. [7]. It is a simplified version of the real-life problem of multi-robot planning [38]. 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 not sufficient 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 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 they meet as quickly as possible. Figure 3 gives an example position in this problem. Experimental results can be found in [20].

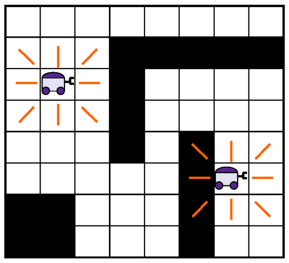


Figure 3: Meeting Under Uncertainty on a Grid (Courtesy of Daniel Bernstein).

1.2 Outline

All of the described problems are examples for decentralized control of multiple agents. In Sections 2.1 and 2.2, four different formal models for this class of problems are presented and their expressiveness and complexity is compared. Equivalence of all these models in terms of expressiveness and complexity is established. In Section 2.4, another formal model for describing decentralized problems is presented - an orthogonal approach to the other models. This framework differs from the ones presented earlier regarding expressiveness as well as computational complexity. Section 2.5 continues with interesting sub-classes of the general problem which lead to lower complexity classes. Some of these sub-problems are computationally tractable. But as not all interesting problems fall into one of the easier sub-classes, researchers have worked on non-trivial optimal algorithms for the complete problem, which are presented in Section 3, as well as approximation techniques, which are presented in Section 4. For some approaches, specific limitations are identified and discussed in detail. In Section 5, all presented algorithms are compared - those finding optimal solutions as well as those finding approximate solutions - and their advantages as well as drawbacks are discussed. Finally, in Section 6 conclusions are drawn and a short overview of open research questions is given.

2 Formal Models for Optimal Multi-Agent Planning

To formalize the problem of decentralized control of multiple agents, different formal models have been introduced in the last couple of years. In all of the following models, at each step, each agent takes an action, the state transitions and each agent gets a local observation. Then the environment generates a global reward which depends on the set of actions taken by all the agents. Figure 4 gives a schematic view of a decentralized process with two agents. It is important to note that each agent gets an individual observation, but the reward generated by the environment is the same for all agents. Thus, each agent wants to maximize the global reward, which is best for all agents. This is an important characteristic of cooperative multi-agent systems. In non-cooperative multi-agent systems such as partially observable stochastic games (POSGs), each agent has its own private reward function.

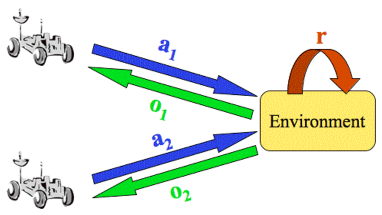


Figure 4: Schematic View of a Decentralized Process With 2 Agents, a Global Reward Function and Private Observation Functions (Courtesy of Christopher Amato).

One important aspect that differentiates the different formal models is the treatment of communication. Some frameworks explicitly model the communication actions of the agents and others subsume them under the general action sets. Either way has different advantages and disadvantages, depending on the focus of the analysis. In the next section, two formal models without explicit communication are presented. Following is the presentation of two frameworks that explicitly model communication actions. We prove that all four models are equivalent in terms of expressiveness as well as in terms of 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,...,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 $\vec{a} = \langle a_1, ..., a_n\rangle$ denotes a joint action.
- P is a Markovian transition probability table. $P(s'|s, \vec{a})$ denotes the probability that taking joint action $\vec{a}$ in state s results in a transition to state s'.

- 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 $\vec{o} = \langle o_1, ..., o_n \rangle$ denotes a joint observation.
- O is a table of observation probabilities. $O(\vec{o}|\vec{a}, s')$ is a rational representing the probability of observing joint observation $\vec{o}$ given that joint action $\vec{a}$ was taken and led to state s'. Here $s' \in S, \vec{a} \in \vec{A}, \vec{o} \in \vec{\Omega}$.
- $R : S \times \vec{A} \rightarrow \Re$ is a reward function. $R(\vec{a}, s')$ is a rational representing the reward obtained from transitioning to state s' after taking joint action $\vec{a}$.
- If the DEC-POMDP has a finite horizon, it is represented by a positive integer T.

This framework was first proposed by Bernstein et al. [6] in 2000. In 2002, they introduced another version of this model [7] that allows for a more general observation and reward function that differentiate between different originating states. But the more general functions can easily be simulated in the original model by growing the state space to the crossproduct of S with itself. Here, the earlier definition is used, because it simplifies showing the equivalence with the MTDP model (see Theorem 1 on page 10).

Note that we focus on finite-horizon problems, where the decentralized process ends after a finite number of steps. For infinite-horizon problems a discount factor to weight the rewards collected over time has to be introduced. For an in-depth coverage of infinite-horizon models and algorithms see Bernstein [5]. All models presented in this section generalize in a straightforward way to the infinite-horizon case. However, in Section 4, six different approximation techniques for DEC-POMDPs are presented, some for finite-horizon problems and some for the infinite-horizon case. It is important to note that it is not always straightforward to generalize these algorithms to the other cases.

Definition 2 (Local Policy for a DEC-POMDP) A local policy for agent i, δ_i, is a mapping from local histories of observations $\overline{o}_i = o_{i1} \cdots o_{it}$ over Ω_i, to actions in A_i.

Definition 3 (Joint Policy for a DEC-POMDP) A joint policy, $\delta = \langle \delta_1, ..., \delta_2 \rangle$, 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 can be over an infinite or a finite horizon. See Bernstein et al. [7] for more details on the model.

2.1.2 The MTDP Model

In 2002, Pynadath and Tambe [34] presented the MTDP framework which is very similar to the DEC-POMDP framework. The model as presented below uses one important assumption:

Definition 4 (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 5 (MTDP) A multiagent team decision problem (MTDP) is a tuple $\langle \alpha, S, \mathbf{A}_\alpha, P, \boldsymbol{\Omega}_\alpha, \boldsymbol{O}_\alpha, \boldsymbol{B}_\alpha, R, T \rangle$, where

- α is a finite set of agents, numbered 1,...,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, $\mathbf{A}_\alpha \equiv \Pi_{i \in \alpha} A_i$.
- $P : S \times \mathbf{A}_\alpha \times S \rightarrow [0, 1]$ is a probabilistic distribution over transitioning 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, $\boldsymbol{\Omega}_\alpha \equiv \Pi_{i \in \alpha} \Omega_i$.
- $\mathbf{O}_\alpha$ is a joint observation function modeling the probability of receiving a joint observation $\boldsymbol{\omega}$ after taking joint action $\mathbf{a}$ and transitioning to state s. I.e., $\mathbf{O}_\alpha(s, \boldsymbol{a}, \boldsymbol{\omega}) = Pr(\boldsymbol{\Omega}_\alpha^t = \boldsymbol{\omega} | S^t = s, \mathbf{A}_\alpha^{t-1})$.
- $\mathbf{B}_\alpha$ 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 so called **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.
- $R : S \times \mathbf{A}_\alpha \rightarrow \Re$ is a reward function representing a team's joint preferences.
- If the MTDP has a finite horizon, it 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 it does nothing but creating a list of observations. However, if this assumption is relaxed or somebody finds a way of mapping observations to a **compact** belief state, this additional element might become justified. But so far, there has been no specific proposal of a state estimator function not using the assumption of perfect recall and until now, no compact representation of belief states in multi-agent settings has been introduced.

Definition 6 (Domain-level Policy for an MTDP) The set of possible domain-level policies in an MTDP is defined as the possible mappings from belief states to actions, $\pi_{iA} : B_i \rightarrow A_i$.

Definition 7 (Joint Domain-level Policy for an MTDP) A joint domain-level policy for an MTDP, $\boldsymbol{\pi}_{\alpha A} = \langle \pi_{1A}, ..., \pi_{nA} \rangle$, is a tuple of domain-level policies, one for each agent.

Solving an MTDP means finding a joint policy such that the global reward is maximized.

2.1.3 Computational Complexity and Equivalence Results

Before the equivalence of the DEC-POMDP model and the MTDP model can be shown it has to be defined what **equivalence** means. All frameworks described in this paper are used to model decentralized decision problems. Thus, they can also be seen as *sets of problem instances*. In terms of complexity theory, a set of problem instances where each has a "yes/no" answer is a **complexity class**. Section 2.3.2 describes how a DEC-POMDP problem can be converted to a "yes/no" problem and analyzes the complexity of the resulting complexity class. The analysis shows that the first four models presented are NEXP-complete, i.e. they can be solved in **nondeterministic exponential time** and every other problem in NEXP can be efficiently reduced to these problems, where here efficiently means in polynomial time. For DEC-POMDPs this means: DEC-PODMP $\in$ NEXP and $\forall C \in$ NEXP: $C \leq_p$ DEC-POMDP. For more details on complexity theory see Papadimitriou [29].

Definition 8 (Equivalence of Models/Problems) Two models are called **equivalent** if their corresponding decision problems are complete for the same complexity class.

Thus, to show that two problems are equivalent, a completeness proof such as in Section 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 one class of problems A to another class of problems 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. This means, 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 performed. 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: To show: 1. DEC-POMDP $\leq_p$ MTDP and 2. MTDP $\leq_p$ DEC-POMDP.

1. DEC-POMDP $\leq_p$ MTDP:
   A DEC-POMDP is a tuple $\langle I, S, \{A_i\}, P, \{\Omega_i\}, O, R, T\rangle$ and an MTDP is a tuple $\langle \alpha, S', \mathbf{A}_\alpha, P', \boldsymbol{\Omega}_\alpha, \boldsymbol{O}_\alpha, \boldsymbol{B}_\alpha, R', T'\rangle$. 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 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, an optimal policy for the resulting MTDP then constitutes an optimal policy for the original DEC-POMDP, where the final policies are mappings from local histories of observations to actions.

2. MTDP $\leq_p$ DEC-POMDP:
For $\alpha, S', \mathbf{A}_\alpha, P', \mathbf{\Omega}_\alpha, \boldsymbol{O}_\alpha, R', T'$ 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. The assumption in the MTDP model is that the agents have perfect recall, i.e. they 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 which is a mapping from histories of observations to actions can be found. The solution to the DEC-POMDP is thus also a solution to the original MTDP.

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

It becomes apparent that the only syntactical difference between MTDPs and DEC-PODMPs is the additional state estimator function and the resulting belief state of the MTDP model. But as we have shown, 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 therewith implicitly existent in a DEC-POMDP, too. Thus, we consider a DEC-POMDP to be a somewhat cleaner and more compact model than an MTDP. Accordingly, from now on we use DEC-POMDPs to model decentralized decision making for multiple agents.

2.2 Models with Explicit Communication

Both models presented in Section 2.1 have been extended to models where the communication actions are modeled explicitly. In the following two models, the interaction among the agents is a process in which agents perform an action, then they observe their environment and then send a message that is instantaneously received by the other agents (no delays in the system). Both models allow for a general syntax and semantic of the 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. Goldman and Zilberstein [19]).

2.2.1 The DEC-POMDP-COM Model

The following model is equivalent to the one described in Goldman and Zilberstein [17]. The formal definition of some parts have been changed to make the DEC-POMDP-COM model presented here a straightforward extension of the afore described DEC-POMDP model.

Definition 9 (DEC-POMDP-COM) A decentralized partially observable Markov decision process with communication (DEC-POMDP-COM) is a tuple
$\langle I, S, \{A_i\}, P, \{\Omega_i\}, O, \Sigma, C_\Sigma, R, T \rangle$ 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. $\sigma_i \in \Sigma$ denotes an atomic message sent by agent i. $\vec{\sigma} = \langle \sigma_1, ..., \sigma_n \rangle$ denotes a joint message, i.e. a tuple of all messages sent by the agents in one time step. A special message that belongs to Σ is the null message, ε_σ. This message is sent by an agent that does not want to transmit anything to the other agents. The agents do not incur any cost in sending a null message.
- C_Σ is the cost of transmitting an atomic message. $C_\Sigma : \Sigma \rightarrow \Re, C_\Sigma(\varepsilon_\sigma) = 0$.
- R is the reward function. $R(\vec{a}, s', \vec{\sigma})$ represents the reward obtained by all agents together, when they execute the joint action $\vec{a}$, transition to the state s' and send the joint message $\vec{\sigma}$.

It is important that in this model, $\{A_i\}$ is a set of control actions which does not include actions to communicate between agents.

Definition 10 (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 $(j \neq i)$. $\delta_i^A : \Omega^* \times \Sigma^* \rightarrow A_i$.

Definition 11 (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 $\overline{o}_i$ and o, the last observation perceived after performing the last local action, over Ω_i and histories of messages σ_j received $(j \neq i)$. $\delta_i^\Sigma : \Omega^* o \times \Sigma^* \rightarrow \Sigma$.

Definition 12 (Joint Policy for a DEC-POMDP-COM) A joint policy $\delta = \langle \delta_1, ..., \delta_n \rangle$ is defined to be a tuple of local policies, one for each agent, where each δ_i is composed of the communication and the action policy for agent i.

Solving a DEC-POMDP-COM means finding a joint policy for a DEC-POMDP-COM that maximizes the expected total reward. This can be over an infinite or a finite horizon. See Goldman and Zilberstein [17] for more details on the model.

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

Proof: 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 where then the cost function for the communication messages becomes obsolete. Thus, $\Sigma = \emptyset$ and $C_\Sigma = \emptyset$. Obviously a solution for the resulting DEC-POMDP-COM is also a solution for the original DEC-POMDP.

2. DEC-POMDP-COM $\leq_p$ DEC-POMDP
   A DEC-POMDP-COM is a tuple $\langle I, S, \{A_i\}, P, \{\Omega_i\}, O, \Sigma, C_\Sigma, R, T\rangle$ and a DEC-POMDP is a tuple $\langle I', S', \{A_i\}', P', \{\Omega_i\}', O', R', T'\rangle$. The only difference between these models is the explicit modeling of the communication actions in the DEC-POMDP-COM model. For the second reduction these communication actions 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 **control state**. And $\langle s, 1\rangle$ represents a system state where only communication actions are possible, denoted **communication state**.
   - Each communication message is added to the action sets: $\{A_i\}' = \{A_i\} \cup \Sigma$. To differentiate control actions from communication actions, control actions are denoted as $\vec{a}$ and communication actions are denoted as $\vec{\sigma}$.
   - For each possible joint communication message, one new observation is added to the observation sets, representing a tuple of $n-1$ received messages by agent i: $\{\Omega_i\}' = \{\Omega_i\} \cup \Sigma^{n-1}$. Now, $\vec{o}$ denotes a normal observation and $\vec{\sigma}_{-i}$ denotes an observation representing $n-1$ communication messages, i.e. all messages sent expect the one from agent i.
   - The state transition table P' is changed in such a way that if the agents are in a control state, every joint control action leads to a communication state. The same way, if the agents are in a communication state, every communication action leads to a control state. In the former case, the probabilities are the same as for state transitions in the original DEC-POMDP-COM. In the later case, every communication action leads to a new control state. Thus:

     $$\forall s, s' \quad P'(\langle s', 1\rangle | \langle s, 0\rangle, \vec{a}) = P(s'|s, \vec{a})$$

     and

     $$\forall s \quad P'(\langle s, 0\rangle | \langle s, 1\rangle, \vec{\sigma}) = 1 \quad \text{and} \quad \forall s' \neq s \quad P'(\langle s', 0\rangle | \langle s, 1\rangle, \vec{\sigma}) = 0$$

     Furthermore, the probabilities for transitions that are not possible are set to 0 (e.g. a transition from a control state to another control state, or a transtion from a communication state to another communication state, or any transition after executing a communication action in a system state where only control actions are possible, or any transition after executing a control action in a system state where only communication actions are possible).

- The observation function O' has to be changed in such a way that after taking the joint communication action σ the probability of observing this particular joint communication message is 1 and the probability of observing any other communication message is 0. Thus:

$$O'(\vec{o}|\vec{a}, \langle s', 1\rangle) = O(\vec{o}|\vec{a}, s') \quad \text{and} \quad O'(\vec{\sigma_{-i}}|\vec{\sigma}, \langle s', 0\rangle) = \begin{cases} 1 & \text{if} \quad \vec{\sigma}_{-i} \cap \sigma_i = \vec{\sigma} \\ 0 & \text{else} \end{cases}$$

  Furthermore, the observation probabilities of observing any impossible observations are set to 0, analogously to the way described for the transition probabilities.

- The reward function R' has to be changed to account for the communication costs as defined by C_Σ. I.e., if s' is the system state after joint communication action $\vec{\sigma} = \langle \sigma_1, ...\sigma_n\rangle$ was taken, then $R'(\vec{\sigma}, s') = \sum_{i=1}^{n} C_\Sigma(\sigma_i)$. Obviously, the reward for a joint communication action is negative.

In the resulting DEC-POMDP, the agents start off taking a joint control action. Then they transition to a communication state. After taking a joint communication action, the agents transition to a control state. 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. Thus, an optimal policy for the resulting DEC-POMDP is also an optimal policy for the original DEC-POMDP-COM.

It remains to show that the presented mapping from a DEC-POMDP-COM to a DEC-POMDP does not increase the size of the problem more than polynomial in the problem description. This is indeed questionable: All possible joint messages have been added to the observation set, meaning that an agent observes a tuple of $n-1$ messages after taking a communication action. Thus the observation set's size is now exponential in the number of agents: $\{\Omega_i\} = \{\Omega_i\}' \cup \Sigma^{n-1}$. But for the complexity analysis, the size of the observation set is crucial. In the first step of the proof that DEC-POMDP $\in$ NEXP (see Theorem 4 on page 18), a joint policy is guessed and written down in exponential time. This is possible, because a joint policy is a mapping from local observation histories to actions, one mapping for each agent. Because the finite horizon T is bounded by $|S|$, all observation histories have at most length $|S|$ and thus the number of possible observation histories is $\{\Omega_i\}^{|S|}$, thus it is bounded exponentially by the problem description. But with the presented mapping, the new observation sets $\{\Omega_i\}$ have a size exponential in the number of agents, thus the number of possible histories of length T is now: $(|\Sigma|^{|I|})^{|S|}$. Fortunately, this number is still exponentially bounded in the size of the problem, because $(|\Sigma|^{|I|})^{|S|} = (2^{\log(|\Sigma|)\cdot|I|})^{|S|} = 2^{\log(|\Sigma|)\cdot|I|\cdot|S|}$. All other problem variables (state space S, actions sets $\{A_i\}$, transition probability table P, observation function O) also do not grow more than polynomially. Thus, the DEC-POMDP and the DEC-POMDP-COM model are equivalent. □

2.2.2 The COM-MTDP Model

Definition 13 (COM-MTDP[1]) A communicative multiagent team decision problem (COM-MTDP) is a tuple $\langle \alpha, S, \mathbf{A}_\alpha, P, \mathbf{\Omega}_\alpha, \boldsymbol{O}_\alpha, \mathbf{\Sigma}_\alpha, \boldsymbol{B}_\alpha, R \rangle$, where

- $\alpha, S, \mathbf{A}_\alpha, P, \mathbf{\Omega}_\alpha$ and $\boldsymbol{O}_\alpha$ remain defined as in an MTDP.
- $\mathbf{\Sigma}_\alpha$ is a set of combined communication messages, which is defined by $\mathbf{\Sigma}_\alpha \equiv \Pi_{i \in \alpha} \Sigma_i$ 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. Now, each agent $i \in \alpha$ forms a 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. Its is defined by $R : S \times \mathbf{A}_\alpha \times \mathbf{\Sigma}_\alpha \rightarrow \Re$.

Also for 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, temporal delays, etc. that might occur in the communication channel. Furthermore, it could possibly do some preprocessing with the observations and messages. But so far, nobody has exploited neither possibility. In fact, using the complete history of observations and messages always leads to the best possible policies.

Definition 14 (Communication Policy for a COM-MTDP) A communication policy for a COM-MTDP is defined as a mapping from the extended belief state space to communication messages, i.e. $\pi_{i\Sigma} : B_i \rightarrow \Sigma_i$.

Definition 15 (Joint Communication Policy for a COM-MTDP) A joint communication policy for an MTDP, $\boldsymbol{\pi}_{\alpha\Sigma} = \langle \pi_{1\Sigma}, ..., \pi_{n\Sigma} \rangle$, is a tuple of communication policies, one for each agent.

Definition 16 (Joint Policy for a COM-MTDP) A joint policy for a COM-MTDP is a pair 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 for a COM-MTDP that maximizes the expected total reward. This can be over an infinite or over a finite horizon.

[1]Adapted from Pynadath and Tambe [34].

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: To show: 1. DEC-POMDP-COM $\leq_p$ COM-MTDP and 2. COM-MTDP $\leq_p$ DEC-POMDP-COM.
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 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 costs of the communication actions C_Σ are not explicitly modeled in the COM-MTDP model, but instead part of the extended reward function.

   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, an optimal policy for the resulting COM-MTDP then constitutes an optimal policy 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 mapping as in part 1 is used. The only remaining element is the extended set of possible combined belief states, $\boldsymbol{B}_\alpha$, which again is implicitly existent 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. Now the **extended state estimator function** forms the belief state with the following three operations:

   a) At the beginning, it initializes the belief state with an empty history.
   b) It appends every new observation agent i receives to its belief state.
   c) It appends new messages agent i receives to is 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 does nothing but building up a complete history of observations and messages - the same that also happens in a DEC-POMDP-COM, where it is not merged into one belief state, but hold separate in two different histories. Thus, the history of observations and communication messages can simply be extracted from the belief state of the COM-MTDP and then the resulting DEC-POMDP-COM can be solved. This means finding a policy which is a mapping from histories of observations and communication messages to actions. Thus, the solution to the DEC-POMDP-COM is also a solution to the original COM-MTDP.

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

Corollary 1 *All of the aforementioned models, i.e. the DEC-POMDP model, the DEC-POMDP-COM model, the MTDP model and the COM-MTDP model are equivalent.*

Proof: The proof follows immediately from theorems 1,2 and 3. □

As explained in Section 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 vs. COM-MTDPs. Whether one uses the model with or without explicit communication messages depends on the specific purpose. The models are equivalent, but for a specific research analysis one of the models might be superior. 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 Goldman and Zilberstein [19]). If communication actions shall be considered as any other action, the DEC-POMDP model is more compact and thus preferable over the DEC-POMDP-COM model.

2.3.2 Complexity Results

The first complexity results for Markov decision processes go back to Papadimitriou and Tsitsiklis [31] (1987) 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 [30] could show that decentralized control of MDPs must be at least NP-hard. But a complete complexity analysis also giving tight upper bounds was still missing.

In 2000, Bernstein et al. [6] were the first to prove that DEC-POMDPs as well as DEC-MDPs 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 pratice. Moreover, it is strongly believed by most complexity theorists that EXP $\neq$ NEXP and that accordingly the problem needs double exponential time to be solved in the worst case. Note that this has **not** been proven formally but for the remainder of this paper it is assumed that it is true. Accordingly, when saying that any algorithm that solves finite-horizon DEC-POMDPs optimally needs double exponential time in the worst case, this is always under the assumption that EXP $\neq$ NEXP.

Below, a sketch of the original proof is presented. Obviously, due to the equivalence results from Corollary 1, all complexity results that can be shown for DEC-POMDPs are at the same time also true for DEC-POMDP-COMs, MTDPs and COM-MTDPs. The complexity proof only deals with two agents as this is already sufficient to show the NEXP-completeness. Let DEC-POMDP$_n$ denote a DEC-POMDP with n agents and accordingly DEC-POMDP$_2$ denotes a 2-agent problem.

As explained in Section 2.1.3, when analyzing the complexity of a model, the decision problem is considered. Thus, the original optimization problem (finding the best policy for a given DEC-POMDP) has to be turned into a decision problem with a "yes/no" answer. For all presented models this can be done by adding a parameter K to denote a threshold value. The decision problem for a DEC-POMDP$_n$ is then stated as follows: Given a DEC-POMDP$_n$ $\langle I, S, \{A_i\}, P, \{\Omega_i\}, O, R, T, K\rangle$, 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 (important for the hardness proof). Here, the finite-horizon version of the DEC-POMDP model with the convention that $T \leq |S|$ is analyzed. All following theorems and proofs in this section are taken from Bernstein et al. [7].

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

Proof: To show: 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. $\square$

Because the original proof of the DEC-POMDP hardness problem proves the NEXP-hardness for a sub-class of the DEC-POMDP model, namely the decentralized Markov decision process (DEC-MDP), first some more definitions are needed. Note that different synonyms for the same definitions are used in the DEC-POMDP and in the MTP framework. Thus, in the following definitions, both notations are listed, but only the first one listed in each definition is used.

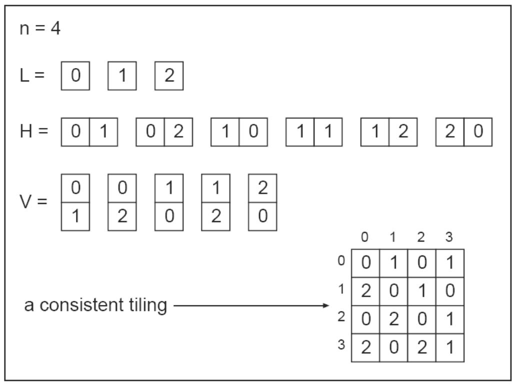
Definition 17 (Joint full observability $\equiv$ Collective observability) A DEC-POMDP is jointly fully observable, if the n-tuple of observations made by the agents together fully determine the current global state. That is, if $O(\vec{o}|\vec{a}, s') > 0$ then $P(s'|\vec{o}) = 1$.

Definition 18 (DEC-MDP) A decentralized Markov decision process (DEC-MDP) is a DEC-POMDP with joint full observability.

Theorem 5 *For all $n \geq 2$ 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. [7]. Here, only a proof sketch is presented to explain the general idea.

For this proof it is sufficient to reduce any NEXP-complete problem to DEC-MDP$_n$. For the following reduction, an NEXP-complete problem called TILING is used, which is described as follows: A board size n (represented compactly in binary) is given, a set of tile types $L = \{\text{tile-0}, ..., \text{tile-k}\}$, and a set of binary horizontal and vertical compatibility relations $H, V \subseteq L \times L$. A **tiling** is a mapping $f : \{0, ..., n-1\} \times \{0, ..., n-1\} \rightarrow L$. 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 5 shows an example of a tiling instance and a corresponding consistent tiling.



Given n, L, H and V, is there a consistent tiling?

Figure 5: The Tiling Problem (Courtesy of Daniel Bernstein).

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. Then a 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 each step. The environment then checks, if this leads to a consistent tiling.

A naive approach would lead to the idea of creating a state for every pair of tile positions for the DEC-MDP. Unfortunately, this would result in an exponential blow-up, but a polynomial time reduction is required. Thus, the environment itself cannot remember all information about the process. But it turns out that it is sufficient to only remember information about the relative position of two tiles to each other (the same?, horizontally adjacent?, vertically adjacent?). The agent's local policies are based on their observation histories, thus if they observe a tile position given from the environment, they can select an action in the next step based on this observation. Figure 6 illustrates how this process can be subdivided into the following five phases:

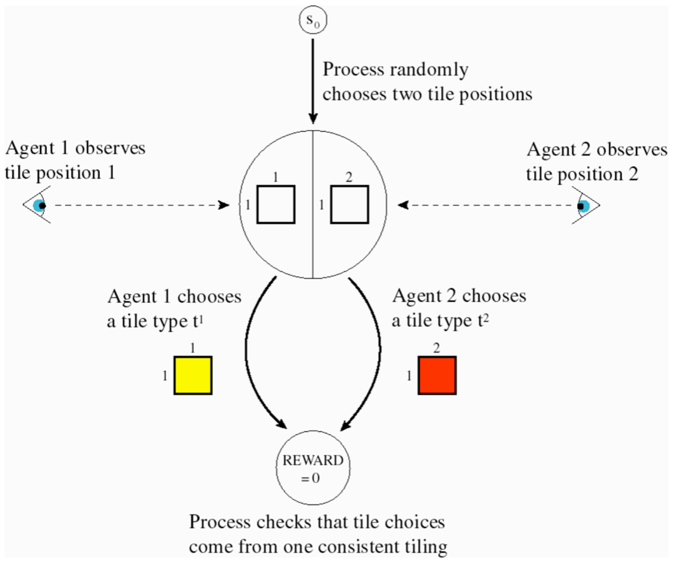
1. **Select Phase**: Select two bit indices and values, each identifying a bit position and the value at that position in the location given to one of the agents.

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.



s_0
Process randomly chooses two tile positions
Agent 1 observes tile position 1
Agent 2 observes tile position 2
1
2
1
1
Agent 1 chooses a tile type t^1
Agent 2 chooses a tile type t^2
1
1
2
1
REWARD = 0
Process checks that tile choices come from one consistent tiling

∃ policy with expected reward 0 ↔ ∃ consistent tiling

Figure 6: Reduction from the Tiling Problem (Courtesy of Daniel Bernstein).

Note that because the tiling problem is reduced to a **DEC-MDP**, joint-observability must be maintained throughout the whole process. I.e., all aspects off the DEC-MDP must be available to the agents through their observations. This is achieved by making all information observable to 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. Only if the tiling conditions hold, the agents receive a non-negative reward. 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-POMDP$_n$ at the same time constitutes a solution to the tiling problem. Accordingly, as the tiling problem is NEXP-complete, solving a finite-horizon DEC-POMDP$_n$ is NEXP-hard. □

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

Proof: Because a DEC-POMDP is a true super class of a DEC-MDP, and as has been shown in Theorem 5 that DEC-MDPs are NEXP-hard, the same follows immediately for DEC-POMDPs. □

Because all four formal models for optimal multi-agent planning are equivalent, the last complexity result also holds true for DEC-POMDP-COMs, MTDPs and COM-MTDPs. Thus, for following research work, either one of the models can be used for theoretical analysis purposes as well as for the development of algorithms. It will always apply to the other models as well. In the following section, a completely different approach to formalize the multi-agent planning problem is presented. It turns out that it is different from the models presented in this section in terms of expressiveness as well as in terms of complexity.

2.4 Models with Explicit Belief Representation

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-MDP/DEC-POMDP model there is probably no way of compactly representing a policy (see also Section 2.5.2 for a more detailed discussion of this topic). The agents have to remember their complete history of observations which results in exponentially large policies.

If there were a way to build up a compact belief state about the whole decentralized process, one would hope to get a lower complexity than NEXP. For one agent, this would include not only expressing the belief about its own local state but also its belief about the other agents (their states, their policies, etc.). Exactly this idea is central for the approach pursued by the I-POMDP model first introduced by Piotr Gmytrasiewicz and Prashant Doshi in 2004 and refined in 2005 [16]. They propose a framework for sequential planning in partially observable multi-agent settings. Their model is even more expressive than the DEC-POMDP model as it allows for cooperative as well as for non-cooperative settings. Thus, it is closely related to partially observable stochastic games (POSGs). But in contrast to classical game theory, this approach does not search for equilibria or care about stability. Instead it focuses on finding the best response action for one single agent with respect to the belief about the other agents. Thereby, it avoids the problems of equilibria-based approaches, namely the non-uniqueness (multiple equilibria) and that they only describe the agent's optimal action, if an equilibrium has been reached.

The formal I-POMDP model and corresponding solution techniques are presented below. Following some complexity results are discussed and unfortunately, as one expected (because the expressiveness of this model is even larger then the DEC-POMDP model), the new framework does not result in a lower worst case complexity than the DEC-POMDP model. In fact, a model that only allows for approximate solutions already needs double exponential time to be solved. Finally, the applicability of this model and its relationship to DEC-POMDPs is examined.

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 a so-called **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 believes which makes finding optimal solutions within this model problematic. This topic is discussed later in Sections 2.4.3 and 2.4.4.

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

Definition 19 (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 the 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 the reward function representing the 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 20 (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 $OPT(\theta_i)$.

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

Definition 21 (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 for 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 with types.

An I-POMDP is a generalization of POMDPs to handle 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' beliefs. For simplicity of notation, just two agents are considered, i.e. agent i interacting with one other agent j. But the formalism is easily extendable to any number of agents.

Definition 22 (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.

- 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 agents' autonomy, because agents' actions do not change the other agents' models directly. Actions can only change the physical state and thereby might indirectly change the other agent's belief after receiving an observation.

- Ω_i is the set of observations that agent i can make.

- O_i is an observation function which is defined by $O_i : S \times A \times \Omega_i \rightarrow [0, 1]$. This implies the **model non-observability assumption (MNO)** which ensures another aspect of agents' autonomy, namely the agents cannot observe each others 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, but usually only the physical state matters.

Definition 23 (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:

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

where:

- β is a normalizing constant
- $\sum_{is^{t-1}:\hat{m}_j^{t-1}=\hat{\theta}_j^t}$ indicates the summation over all possible interactive states whereby only those are considered, where the frame of agent j is fixed.
- 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 timestep 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 24 (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. But two new factors also have to be taken into account: At first, the change of the physical state now depends on the actions performed by both agents. Thus agent i has to take into account the other agent's model to infer the probabilities of its actions. At second, the changes in the model of j itself have to be included in the update, because the other agent also receives observations and updates its belief. The update of 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. Gmytrasiewicz and Doshi [16] state that "the update of the possibly infinitely nested belief over other's types is, in general, only asymptotically computable". This leads to the important question in how far optimality can be achieved with this model. This problem is discussed in more detail in Section 2.4.3 and 2.4.4.

Definition 25 (Value Function in I-POMDPs) Each belief state in I-POMDP has an associated value reflecting the maximum payoff the agent can expect in this belief state:

$$U(\theta_i) = \max_{a_i \in A_i} \{ \sum_{is} ER_i(is, a_i) b_i(is) + \gamma \sum_{o_i \in \Omega_i} Pr(o_i|a_i, b_i) \cdot U(\langle SE_{\theta_i}(b_i, a_i, o_i), \hat{\theta}_i \rangle) \} \tag{1}$$

where $ER_i(is, a_i)$ is the expected reward for state is taking action a_i, defined by: $ER_i(is, a_i) = \sum_{a_j} R_i(is, a_i, a_j) Pr(a_j|m_j)$. Equation (1) is a basis for value iteration in I-POMDPs.

Definition 26 (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:

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

Due to possibly infinitely nested beliefs, a step of value iteration and optimal actions is only asymptotically computable (them same as for belief updates).

2.4.2 Finitely Nested I-POMDPs

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

Definition 27 (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 which is called the **strategy level** of the finitely nested I-POMDP. Now, the belief update, value function, and the optimal actions can be computed because recursion is guaranteed to terminate at 0-th level of the belief nesting. See [16] for details on the inductive definition of levels in the interactive state space as well as for 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 a wrong model of the other agent. But to achieve unbounded optimality, an agent would have to model other agent's modeling the original agent. 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 application to their framework remains an open problem. In fact, we doubt that the finitely nested I-POMDP model allows for approximate optimality. We believe that for every value of the strategy level a corresponding example problem can be found such that the difference between the optimal solution (for example computed by the DEC-POMDP model) and the approximate solution can be arbitrarily large. But this remains an interesting open research question.

2.4.3 Complexity Analysis for Finitely Nested I-POMDPs

As has been mentioned, the general I-POMDP model is not computable. But for finitely nested I-POMDPs, it is possible, to compute the solutions defined by the model. At the 0-th level of the belief nesting, the resulting types are POMDPs, were the other agent's actions are folded into the T, O and R functions as noise. An agent's first level belief are probability distributions over S and over 0-level models of the other agents. Given these probability distributions and the solved POMDPs of the 0-level beliefs, the level-1 beliefs can also be solved as POMDPs. This solution then provides probability distributions for the next level model, and so on. It is assumed that the number of models considered at each level is bounded by a finite number, M. Then, solving an I-POMDP$_{i,l}$ is equivalent to solving $O(M^l)$ POMDPs. Gmytrasiewicz and Doshi then conclude that the complexity of solving an I-POMDP$_{i,l}$ is PSPACE-hard for finite time horizons, and undecidable for infinite horizons, just like for POMDPs. Furthermore, they assert PSPACE-completeness for the case where the number of states in the I-POMDP is larger than the time horizon.

Unfortunately, this complexity analysis is only correct if the strategy level is considered to be fixed. Otherwise, solving an I-POMDP$_{i,l}$ with the strategy level parameter l as part of the problem description, is at least EXP-hard. Because the integer l is given in binary encoding, its size is $log(l)$. Thus, solving $O(M^l)$ POMDPs obviously takes double exponential time in the size of l. Thereby, it cannot be in PSPACE anymore. In fact, it is likely to be EXPSPACE-hard, as it uses the full power (with regard to time) of this complexity class, namely double exponential time in contrast to DEC-POMDPs which only use non-deterministic exponential time. Obviously, this is a preliminary evaluation of the complexity of finitely nested I-POMDPs. A complete analysis asserting the correct complexity has to include a formal reduction proof. For now, we simply see now way how to guess and then evaluate a solution for finitely nested I-POMDPs in less than double exponential time. However, in practice algorithms for I-POMDPs and DEC-POMDPs both use double exponential time.

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

As mentioned in the beginning of Section 2.4, the I-POMDP model is even more expressive than the DEC-POMDP model, because I-POMDPs represent an individual agent's point of view on the environment and the other agents. Thereby it also allows for non-cooperative settings. For cooperative settings like in 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 agents are cooperative to build up an adequate belief. But an adequate belief cannot be built up, because it is infinitely nested. Reduced to the cooperative case and compared to the DEC-POMDP model, some drawbacks of the I-POMDP model become apparent:

- Unlike the DEC-POMDP model, the I-POMDP model does not allow for computing the optimal solution. The infinite nesting of beliefs leads to non-computable agent functions.
- Due to self-reference, finitely nested I-POMDPs can only be boundedly optimal, as discussed in Section 2.4.2. Thus, the solution of a finitely nested I-POMDP is always only a suboptimal approximation.

- Even for the finitely nested I-POMDPs, there is an obstacle to solving this problem. 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 it with the corresponding probability. In their complexity analysis, Gmytrasiewicz and Doshi make the assumption that the number of models considered at each level is bounded by a finite number which is another approximation. No bounds are given, how far from optimal the solution could be. It seems that for the example given, it is possible to represent the belief over infinitely many models compactly and that also an evaluation is possible. But the authors do not show how this is possible in general, i.e. why a finite number of models is generally sufficient.

- Even with both approximations (i.e. the finite nesting and the bounded number of models), the worst case complexity of an I-POMDP$_{i,l}$ is probably EXPSPACE for the finite-horizon case. As NEXP $\subseteq$ EXPSPACE, the complexity of I-POMDP$_{i,l}$ is at least as high as the one of DEC-POMDPs.

Despite all these drawbacks, the I-POMDP model is still a useful contribution to the group of formal models for decentralized control of multiple agents. Obviously, it does not allow for optimal solutions of decentralized processes. But optimal solutions in the DEC-POMDP framework are also only computable for very small problems and for a very short horizon due to the complexity barriers. To solve any but toy problems, approximation techniques for DEC-POMDPs have to be used (see Section 4). Most solutions computed with these approximate algorithms can also be arbitrarily far away from optimal, just like solutions for I-POMDPs. Thus, it remains an open research question, how the approximate version of the I-POMDP relates to the approximations of the DEC-POMDP, what can be said about theoretical bounds on the algorithms and how they prove oneself in experimental settings. Recently, Doshi and Gmytrasiewicz have followed up their work with approximation techniques for I-POMDPs. To better deal with the high belief space complexity, they use an approach based on particle filtering that increases the scalability of their algorithm [12, 13].

2.5 Sub-Classes of DEC-POMDPs

2.5.1 Motivation for Analyzing Sub-Classes

As shown in Section 2, 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. [35] 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 Section 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, the agents are very dependent on each other, meaning that their actions have much influence on each others next state (e.g. in the multi-access broadcast channel problem). In other problems, the agents are solving (mostly) independent local problems and only interact with each other very rarely, or their actions only slightly influence each other's state (e.g. meeting on the grid, mars rover navigation).

If one phrases 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 others observation or local state. Moreover, the agents cannot communicate with each other and they cannot observe each others observation or state. The only way the agents interact is through a global value function that 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 this problem is the one of controlling the operation of multiple planetary exploration rovers, such as the ones used by NASA to explore the surface of Mars [42] (illustrated in Figure 7). Each one of the rovers has its own part to explore and to collect information about. Periodically, the rovers can communicate with the ground control center but continuous communication is not possible. Some of the exploration sites of the two rovers are overlapping. Thus, if both rovers collect information about these sites (e.g. take pictures) the reward for this 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.

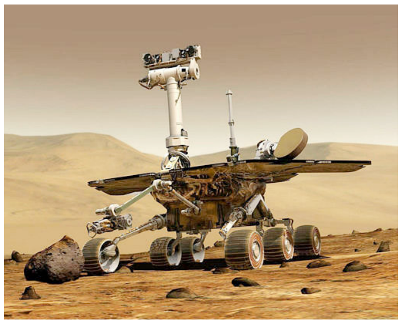


Figure 7: Mars Exploration Rover.

2.5.2 Formal Models

As mentioned in Section 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 28 (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, $S = S_0 \times S_1 \times ... \times S_n$.

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 the external features which all agents observe and are affected by, but which are not changed by the agent's 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 is dependent on its private state features and on the external features.

Definition 29 (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 structured state space on hand, the transition independence of the agents can be formalized.

Definition 30 (DEC-MDP with Independent Transitions) A factored, n-agent DEC-MDP is said to be transition independent if there exists P_0 through P_n such that

$$Pr(s'_i|(s_0,...,s_n),\vec{a},(s'_1,...,s'_{i-1},s'_{i+1},...,s'_n)) = \begin{cases} P_0(s'_0|s_0) & i=0 \\ P_i(s'_i|\hat{s}_i,a_i,s'_0) & 1 \leq i \leq n \end{cases}$$

That means, the external features change based only on the previous external features, and the new local state of each agent dependes 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 31 (DEC-MDP with Independent Observations) A factored, n-agent DEC-MDP is said to be observation independent if there exists O_1 through O_n such that:

$$\forall o_i \in \Omega_i : Pr(o_i|(s_0,...s_n),\vec{a},(s'_0,...,s'_n),(o_1,...,o_{i-1},o_{i+1},...o_n)) = Pr(o_i,\hat{s}_i,a_i,\hat{s}_i{}').$$

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

Definition 32 ("Local" Full Observability) A factored, n-agent DEC-MDP is said to be 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 are true then the definition of 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 should be intuitively clear that if an agent's transitions and observations in a DEC-MDP are independent from the other agents, its next local state also shouldn't depend on the other's actions or observations. This is framed more formally by the following theorem.

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

Proof: See Goldman and Zilberstein [19] for the detailed proof. □

Definition 33 (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

$$\begin{array}{rcl} R((s_0,...,s_n),\vec{a},(s'_0,...,s'_n)) & = & f(R_1(\hat{s}_1,a_1,\hat{s}'_1),...,R_n(\hat{s}_n,a_n,\hat{s}'_n)) \\ & \text{and} & \\ R_i(\hat{s}_i,a_i,\hat{s}'_i) \leq R_i(\hat{s}_i,a'_i,\hat{s}''_i) & \Leftrightarrow & f(R_1...R_i(\hat{s}_i,a_i,\hat{s}'_i)...R_n) \leq f(R_1...R_i(\hat{s}_i,a'_i,\hat{s}''_i)...R_n) \end{array}$$

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. This function is such that maximizing each of the local reward functions individually maximizes the function itself.

It is important to notice that a factored DEC-MPD with transition independence, observation independence, reward independence and local full observability decomposes into n independent local MPDs (which are P-complete). Thus, the problem becomes trivial. However, if just one of these independence relations is missing, then the problem remains an interesting sub-class of DEC-MPDs.

The following section concentrates on the class of factored DEC-MDPs that are transition and observation independent and have local full observability but are not reward independent. Instead, the reward function consists of two different components: The first component is a set of local reward functions that are reward independent, just as having multiple independent MDPs. But the second component depends on the actions of multiple agents and gives reward to the whole system, thus is not reward independent. It is formally defined by the **joint reward structure**, denoted $\boldsymbol{\rho}$. This allows for defining interdependent rewards for cases where the whole system gets a reward after multiple agents have successfully completed a task, i.e. finished a specific order of actions. The formal definition of a joint reward structure is beyond this study. See Becker et al. [4] for a detailed definition and further information. Here, it is only necessary for the following important theorem in the next section.

2.5.3 Complexity Results

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

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

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

The Mars rover example presented earlier qualifies for this class of problems. The rovers have independent transition and observations and local full observability. They mainly solve their tasks independently, but there might be tasks where they have to work together to get a reward, e.g. make different pictures from the same rock from different angles, to finally be able to construct a 3D model. A more detailed example and an optimal algorithm (Coverage Set Algorithm) for this model can also be found in Becker et al. [4]. Furthermore, a more elaborate discussion about the complexity gap between problems that are NEXP-complete and problems that are NP-complete can be found in [37].

By reducing the expressiveness of the original DEC-POMDP model and imposing a specific structure on it, the complexity was reduced from NEXP to NP. But even for this case, algorithms that work in practice with a lot of states are still missing. This gives rise to another class of problems, which is even more specialized, but where the complexity can be reduced to P-completeness.

Definition 34 (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 $R(s, \vec{a}, s') = C(a_1) + ... + C(a_n)$.
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 35 (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 drop in complexity to NP and P respectively in the last two models is explainable by the following theorem.

[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, 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 8 shows schematically the differences in policy representation size which lead to the aforementioned differences in complexity.

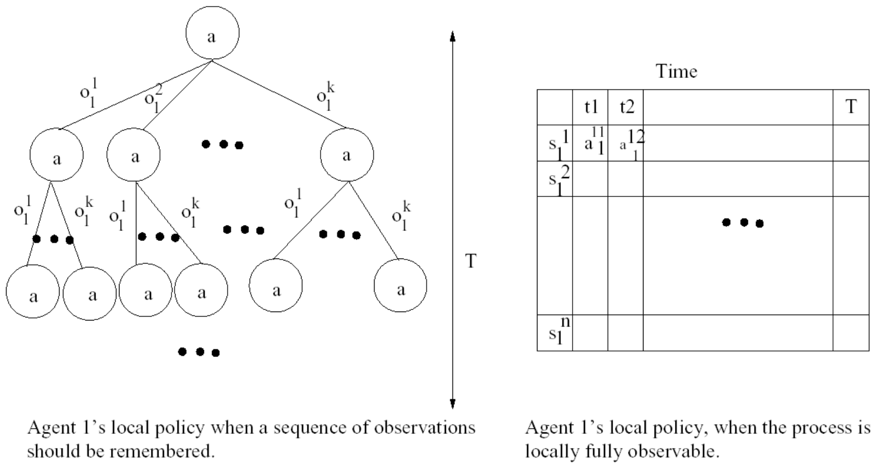

a
o_1^1
o_1^2
o_1^k
T
Time
t1
t2
T
s_1^1
a_1^{11}
a_1^{12}
s_1^2
s_1^n
Agent 1's local policy when a sequence of observations should be remembered.
Agent 1's local policy, when the process is locally fully observable.

Figure 8: Exponential vs. Polynomial Size Policies (Courtesy of Claudia V. Goldman).

This section finishes with an overview of the different problems that emerge depending on the restrictions one imposes on the way the agents interact and the way they observe their environment. Therefore some more definitions regarding observability and communication are needed.

Definition 36 (Partial Observability $\equiv$ Collective Partial Observability) A DEC-POMDP is partially observable if the observations of all agents together still do not determine the global system state.

Definition 37 (Full Observability $\equiv$ 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, the agent's partial view of the problem, i.e. its own local observation already determines the global system state.

Definition 38 (MMDP) A multi-agent Markov decision process is a DEC-POMDP with full observability (Boutilier [9]).

An MMDP is a straightforward extension to 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 was on the complexity due to different levels of observability. But in a multi-agent setting, the way agents interact through communication is also of high interest, from a practical point of view as well as for complexity analyses. See [34],[19] and [18] for a broader coverage on this subject.

Definition 39 (Free Communication) If the agents have free communication in a decentralized process, they can share all their information with each other at every step with no costs incurring.

Definition 40 (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, where the costs 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 and on the level of communication between the agents. A summarizing overview is given in Table 1.

	General Communication	Free Communication
Full Observability	MMDP (P-c)	MMDP (P-c)
Joint Full Observability	DEC-MDP (NEXP-c)	MMDP (P-c)
Partial Observability	DEC-POMDP (NEXP-c)	MPOMDP (PSPACE-c)

Table 1: Complexity Results for Finite-Horizon Problems Depending on Observability and Communication.

Obviously, free communication does not lower the complexity in the case of full observability, because every single agent already observes all information that is available. But when full observability is not given, the agents benefit a lot from free communication in that they can share all information available with no cost, thus resulting in lower complexity classes. Note that this assumes that a) the agent's communication language is expressive enough to share all their observations and b) the agent's have agreed beforehand on how to interpret their messages. If this is not the case, free communication does not automatically lead to full observability, but instead a problem of learning the semantics of a communication language arises. See Allen et al. [1] for a more detailed coverage on this topic.

Looking at the different levels of observability, the big jump in complexity occurs, when going from full observability to joint full observability or partial observability. In these cases, the agents have to remember their histories of observations leading to much higher and provably intractable complexity classes.

3 Algorithms for Optimal Multi-Agent Planning

The complexity results from Section 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. A naive algorithm which simply performs a complete search in policy space thus quickly becomes infeasible which is illustrated below.

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 $\delta^t = (q_1^t, q_2^t, ..., q_n^t)$, one for each agent, where $q_i^t \in Q_i^t$. (For this section, it is assumed that the initial state of the DEC-POMDP is known. If it is not known, but a probability distribution over states is given, $|S|$-dimensional vectors of policy vectors are considered, one for every possible initial state, and then the overall value according to the probability distribution is computed.) Figure 9 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.

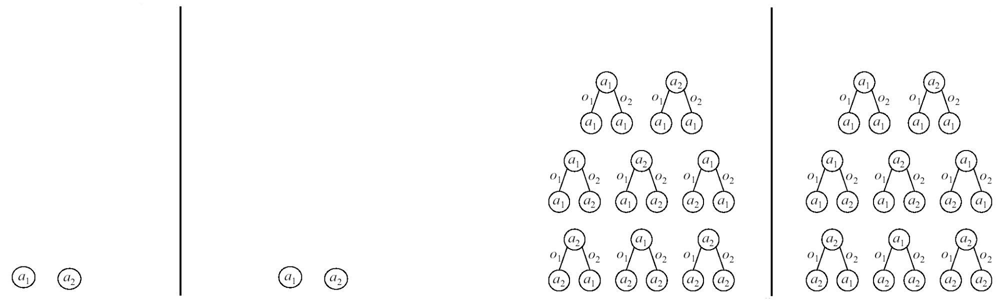


Figure 9: Number of Possible Policies for Horizon 1 and 2 (Courtesy of Daniel Bernstein).

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 possibilities to assign different combinations of actions. For the horizon 1, only one action has to be picked, thus 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 $|A|^{(1+|O|+|O|^2+...+|O|^{t-1})}$. The exponent is a geometric series which can be transformed in the following way:

$$
\begin{aligned}
S_{t-1} = \sum_{i=0}^{t-1} |O|^i &= 1 + |O| + |O|^2 + ... + |O|^{t-1} \\
& \text{multiplying both sides with } |O| \\
|O|S_{t-1} &= |O| + |O|^2 + ... + |O|^{t-1} + |O|^t \\
& \text{adding } S \text{ and subtracting } 2|O|S_{t-1} \\
(1 - |O|)S_{t-1} &= (1 + |O| + |O|^2 + ... + |O|^{t-1}) - (|O| + |O|^2 + ... + |O|^{t-1} + |O|^t) \\
&= 1 - |O|^t \\
&\Rightarrow \; S_{t-1} = \frac{|O|^t - 1}{|O| - 1}
\end{aligned}
$$

Thus the number of possible policy trees for horizon t is $|A|^{\frac{|O|^t-1}{|O|-1}} \in O(|A|^{(|O|^t)})$, thus 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 2 illustrates the infeasibility of a complete search, which becomes most obvious looking at the number of possible joint-policies.

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	$4.6 \cdot 10^{18}$

Table 2: Possible Joint-Policies for 2 Actions, 2 Observations and 2 Agents.

Despite the fact that every optimal solution for a DEC-POMDP will be double exponential in the worst case, researchers have introduced two non-trivial algorithms that also solve the problem optimally but achieve a much better performance than the complete search. They use two very different programming paradigms: One uses dynamic programming and the other one uses heuristic search. In the following sections, both algorithms are presented.

3.1 Dynamic Programming for DEC-POMDPs and POSGs

The first non-trivial algorithm for finding optimal solutions in DEC-POMDPs has been presented by Hansen et al. [22] in 2004. Their algorithm finds sets of policies for partially observable stochastic games (POSGs), which are a special kind of an extensive game with imperfect information [28]. POSGs only differ from DEC-POMDPs in that they allow for one private reward function per agent, thus also allowing non-cooperative settings. But for DEC-POMDPs, the same algorithm finds at least one optimal policy. Their algorithm generalizes two algorithms to dynamic programming for POSGs: dynamic programming for POMDPs and elimination of dominated strategies in solving normal form games.

For a detailed presentation of dynamic programming for POMDPs, see Hansen [21]. The main idea is to incrementally search through the space of possible policies, with 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 $V = \{v_1, v_2, ..., v_k\}$, where

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

It has been shown that 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 41 (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 $v_k \in V^t \setminus v_j$ such that $b \cdot v_k \geq b \cdot v_j$. This test for dominance is performed using linear programming.

In every iteration of the 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, i.e. pruned from the set Q^{t+1}. 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 but 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, which resembles some ideas from 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^t_{-i} denote the set of horizon-t policy tress 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 42 (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 there exists a distribution p over policy trees in $Q^t_i \setminus q_j$ such that

$$\sum_{k \neq j} p(k) v_k(s, q_i) \geq v_j(s, q_i) \text{ for all } s \in S \text{ and } q_i \in Q^k_i.$$

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, taking the 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.

But the DP operator for POMDPs can be generalized to the multi-agent case, i.e. interleaving exhaustive backups with pruning of dominated policies. In the first step of an iteration, the multi-agent DP operator is given Q_i^t and generates Q_i^{t+1}. 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 finally 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 the 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 double exponential in the time horizon t in the worst case. The improvement compared to the 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 as described in Section 1.1.1. Table 3 illustrates the possible improvements in practice. For the horizon 4, the dynamic programming algorithm produces less than 1% of the number of policy trees the brute force algorithm would produce. Note that the brute force algorithm could not compute iteration 4.

Horizon	Brute Force Search	Dynamic Programming
1	(2,2)	(2,2)
2	(8,8)	(6,6)
3	(128,128)	(20,20)
4	(32,768, 32,768)	(300,300)

Table 3: Performance Comparison: Number of Policies for each Agent.

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

3.2 Heuristic Search for DEC-POMDPs

3.2.1 MAA*: A Heuristic Search Algorithm for DEC-POMPs

In 2005, Szer et al. [40] presented an approach orthogonal to dynamic programming for DEC-POMDPs, based on heuristic search. It has some advantages and some disadvantages over the dynamic programming algorithm which are pointed out in Section 3.2.2. The algorithm is based on the idea of the popular A* algorithm and extends it to heuristic best first search in the space of possible joint policies.

The algorithm is based on the same representation for joint policies as presented in Section 3. q_i^t is a depth-t policy tree for agent i and $\delta^t = (q_1^t, ..., q_n^t)$ 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 can thus be stated as finding $\delta^{*T} = \text{argmax}_{\delta^T} V(s_0, \delta)$. As in the brute force search, the algorithm performs a search 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 in the 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. Then the node with the highest heuristic estimate is selected and expanded, going one step further in the search tree. Figure 10 shows a section of such a multi-agent search tree.

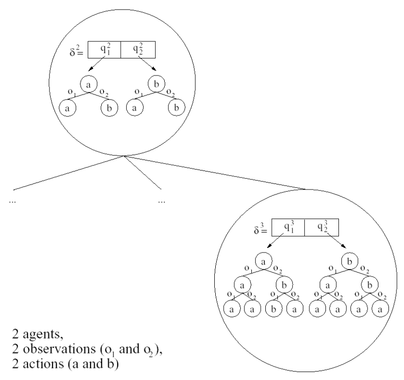

$\delta^2=$ q_1^2 q_2^2
$\delta^3=$ q_1^3 q_2^3
a
b
o_1
o_2
...
2 agents,
2 observations (o_1 and o_2),
2 actions (a and b)

Figure 10: 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).

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 43 (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 than can be attached at the leaf nodes of a policy vector δ^t such that $\{\delta^t, \Delta^{T-t}\}$ consitutes a complete policy vector of depth T.

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

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

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, it is estimated efficiently which leads to:

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

For the heuristic search to be optimal and complete the function H must be **admissible**, i.e. overestimating the exact value of the 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 now crucial for the whole algorithm to define an admissible heuristic function that is efficiently computable but that also is as close as possible to the true value of the state, to perform as much pruning as possible. To define a whole class of admissible heuristics some more definitions are necessary:

- $P(s|s_0, \delta)$ denotes the probability of being in state s after executing the vector of policy trees 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 [40]). 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, provided that the chosen heuristic is good enough, i.e. its estimate is close enough to the exact value function. Then only a few nodes have to be expanded at every level to finally find the optimal solution. Szer et al. [40] 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 $T - t : h^{T-t}(s) = V^{MDP}_{T-t}(s)$.
   As 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*:** For this class of heuristics, the revelation of the underlying state at time t is simulated, the tightest heuristic possible is 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 compared to a complete search which is double exponential in t.

Obviously, there exists a tradeoff between the tightness of a heuristic function (implying the possible pruning) on the one side and the complexity on the other side. 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 that was shown to be able to solve the multi-agent tiger problem as presented in Section 1.1.2 with 2 states, 2 agents, 3 actions and 2 observations up to horizon 4. But it runs out of time when computing horizon 5. Until now, no comparison of the dynamic programming algorithm with the MAA* algorithm on the same machine with a comparable implementation has been performed. Accordingly, no conclusion about which algorithm is superior can be drawn. But as a matter of fact this is not necessary, because both algorithms can only solve small toy problems and to improve scalability approximation techniques such as presented in Section 4 have to be used. Recently, Szer and Charpillet [39] have extended their work to the infinite-horizon case. They present an optimal best-first search algorithm for solving infinite-horizon DEC-POMDPs. In this work, policies are represented as finite-state controllers with limited memory as presented in more detail in Section 4.1. The limitations of the infinite-horizon approach is briefly discussed in Section 4.2.

3.2.2 Limitations of MAA*, Weighted Heuristics and ε-Approximations

Szer et al. [40] 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 concentrated in the last level of the policy tree. Completely expanding a node at level T-1 thus creating all its child nodes for level T takes a lot of time. But fortunately, if the value of one of these children is as high as the heuristic estimate of its parent (which is called a **tie**), the other children do not have to be considered anymore. Thus, in the implementation of the MAA* algorithm, the authors have used a so-called **incremental node expansion**. This means that only one child assignment of actions is constructed in each iteration step and then the value of this child is computed. As long as not all children have been constructed, the parent remains in an **open list**. The authors argue that this might lead to considerable savings in both memory and runtime.

Unfortunately, the incremental node expansion only saves time in the case where the heuristic estimate of a node actually coincides with the value of the best child: $F^T(s_0, \delta^{t-1}) = F^T(s_0, \delta^{*t})$. But even if this case happens, still all children would have to be enumerated until the best one is picked. So far, it is not known if there is a faster way to find the best child. Heuristics might be used to find the best child faster than by simple enumeration. However, we do not believe that heuristics can be found that work well for multiple problems. Good heuristics are likely to be problem-dependent and for most problems, no reasonable heuristics might exist at all. Furthermore, if this case actually would happen this would imply that revealing the true system state to all agents at time t-1 would always lead to the same value, independent of which state it would be. This scenario seems very special and highly unlikely. 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 at time t-1 the agents know the location of the tiger and thus they can open the door with the treasure behind, because they know its location with probability 1. In most problems, these ties will not occur and thus the incremental node expansion will not help a lot, but it will also do no harm. Obviously, heuristics that do not reveal the underlying system state do not suffer from this problem. But until now, no heuristics are known to the authors that can estimate the value of a completion better than the recursive MAA* heuristic. In fact, the heuristic has to estimate the value of a completion without knowledge about the exact start state for the remaining problem. It has to deal with a distribution over possible states. It is unclear, whether any heuristic can do better than revealing the underlying system state at this point, because efficient representations of a multi-agent belief state (after execution of a partial policy) are not known. Even if a compact multi-agent belief state could be computed, it is not obvious how this could be incorporated into the MAA* algorithm.

The low likelihood of ties is a significant drawback that limits the applicability of this heuristic search algorithm. Only if a tie happens for all expanded parent nodes of horizon T-1 the incremental node expansion would save considerable time. If there is at least one node expanded at level T-1 where no such tie happens, all of its children eventually have to be generated to find the best child constituting the optimal solution for horizon T. Unfortunately, completely expanding just one of the last search nodes, thus going from level T-1 to level T, creates $(|A|^{(|\Omega|^{T-1})})^n$ new child nodes. These are more nodes whose value has to be computed exactly, than a complete brute force search would evaluate for a problem with horizon T-1. Thus, in general, the MAA* algorithm can at best solve problems whose horizon is 1 greater than the problems that can already be solved by the naive brute force search algorithm. This conclusion is also illustrated by the empirical results as shown in Table 4.

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	$3.82 \cdot 10^{29}$	$> 2.06 \cdot 10^{14}$

Table 4: Performance Comparison: Number of Evaluated Policy Pairs.

The MAA* algorithm could solve the multi-agent tiger problem with horizon 4, where the brute force search could only solve horizon 3. But, as explained before, the MAA* algorithm evaluated more nodes for the horizon 4 problem than the brute force algorithm does for the horizon 3 problem. The same would be true, comparing horizon 4 with horizon 5. This would mean, MAA* would have to evaluate more than $2.06 \cdot 10^{14}$ nodes to solve the horizon 5 problem, independent of the heuristic used.

Note that this result is vitally important for future research. The work done by the algorithm in the last step (on the lowest level of the search tree) is independent of the heuristic function used. As this work is inevitable and at the same time the largest part of the whole work, no more effort has to be put into looking for tighter heuristics. Furthermore, finding better ways to enumerate the children during expansion of a parent node will also not lead to savings in runtime, because even in the case that only one parent node at level T-1 does not have a tie, all its children will have to be evaluated which is already unfeasible for any problem with horizon ≥ 5.

Another direction of research has recently 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 it would find suboptimal solutions quicker, as it goes deep into the search tree very quickly. The idea is that after suboptimal solutions have been found, large chunks of the search tree can be pruned, thus reducing the size of the open list enormously. If at the end, the open list is empty, the last solution generated is optimal. In fact, if keeping a large open list in memory has a huge impact on the search process, this idea can lead to a considerable speedup. Unfortunately, with double exponential dependence on the time horizon, this approach does not yield considerable savings because even with a weighted heuristic, at least one node at level T-1 has to 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 unfeasible.

Note however that this analysis does not rule out the usefulness of heuristic search for solving DEC-POMDPs in general. The complexity results shown in Section 2.3.2 suggest that there is no efficient algorithm that finds the optimal solution for DEC-POMDPs. MAA* could still serve as a good framework for future research on approximation techniques. For example some researchers are looking at ε-optimal MAA* algorithms, thus algorithms that find an approximate solution for a DEC-POMDP such that: $\frac{\text{Value}(\varepsilon\text{-Approx.})}{\text{Value(Optimal Solution)}} \geq 1 - \varepsilon$. One obvious way of saving computation time is to stop enumerating child nodes once a child has been found whose value is **close enough** to the heuristic estimate of its parent. This was not sufficient for the optimal algorithm but is sufficient for the ε-approximation. Thus, the focus has shifted from ties to "ε-ties". Intuitively one might want to save everything for the last level of the horizon, because that is where the savings due to the ε-tolerance are most beneficial. Unfortunately, this would mean that all possible savings would be wasted for the last level and the same work as before would have to be done for the other levels. Thus, this approach would at best result in one additional step in the horizon. Accordingly, one has to use the ε-tolerance in every layer of the search tree, i.e. split it up such that savings are possible for every step in the horizon. Then again, everything is dependent on the quality of the heuristic, i.e. if revealing the underlying system state results in a heuristic estimate that is within the ε-tolerance. If this is not the case for even one node after horizon-level 4, even finding an ε-optimal solution remains infeasible. As discussed before, a heuristic revealing the underlying system state is likely to significantly overestimate the value. Thus, even ε-ties are unlikely to happen which makes this approach appear to be unpromising.

Note that the MAA* algorithm runs out of time before it runs out of memory in opposition 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. It seems to be a more promising approach to combine the heuristic search algorithm with the dynamic programming algorithm yielding a better and more sophisticated algorithm for finding approximate solutions for finite-horizon DEC-POMDPs. This topic is still part of our ongoing research.

4 Approximating the Optimal Solution for Decentralized Systems

One way of overcoming the high complexity barriers of DEC-POMDPs are algorithms that are aiming at approximate solutions. Unfortunately, is has been shown by Rabinovich et al. [35] in 2003 that even ε-optimal history-dependent joint policies are still NEXP-hard to find in 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 finally handle larger problems. At this point, a small ambiguity of the term "approximation" has to be discussed. In the field of theoretical computer science, an "approximation algorithm" 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 c times the optimal solution) or absolute guarantees (the approximation will be within ε of the optimal solution). Furthermore, as approximation algorithms are normally used to find solutions for NP-hard optimization problems, a polynomial running time of the algorithms 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, six approximate algorithms are presented. All of these algorithm 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 acceptable in light of the NEXP-completeness result. But due to these missing guarantees, it is difficult to compare the different approximation techniques. In the long run, certain problems/experiments have to be established such that comparable benchmark tests can be performed. But a real fair comparison is only possible with similar implementations and on the same machines. Comparable competitions already exist for the field of centralized deterministic planning algorithms but they are still to be established for the field of decentralized planning under uncertainty.

One approach to finding approximate algorithms for decentralized problems is to generalize existing approaches to approximate POMDPs. Unfortunately, most algorithms for POMDPs work with a compact representation of the belief state space. So far, nobody has found a compact way of formalizing the notion of a belief state in a DEC-POMDP, 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 in a straightforward way does not work. Nevertheless, some non-trivial algorithms have been presented in the last few years.

Nair et al. [25] present 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. Their best algorithm DP-JESP incorporates three different ideas. At first, they modify the policy of one single agent while keeping the policies of the others fixed (a similar idea is presented in Section 4.1.3). Second, they use dynamic programming to construct policies iteratively, step by step starting at the end of a finite horizon. This is a similar idea as presented in Section 3.1. Third, and most notably, they only consider the reachable belief states of the DEC-POMDP 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 (up to horizon 4), but the algorithm scales badly with increasing problem size.

Emery-Montemerlo et al. [14] take a game-theoretic approach modeling the problem as a POSG with common payoffs. They approximate the POSG as a series of smaller Bayesian games. In each of these simple games, the agents just have to make the decision about the next action to take. The future reward after taking one action is estimated using heuristics such as Q_{MDP} that turns the remaining problem into a fully-observable problem. Similar to the dynamic programming algorithm proposed by Nair et al. [25] 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. To limit memory-requirements for remembering the history of actions and observations, low-probability histories are pruned. It is particularly appealing that large parts of the algorithm can run in parallel, opening up the possibility for "truly" distributed planning in the sense that each agent does some small part of the planning process as opposed to centralized planning. Unfortunately, the several approximation steps (1. the simplification of the problem as a series of smaller games, 2. the computation of future rewards using a heuristic and 3. the usage of the alternating-maximization algorithm) lead to significant value loss. On the other hand, this algorithm is able to solve substantially larger problems as the one proposed by Nair et al. [25].

Within the scope of this study it is not possible to cover all recently developed algorithms that find approximate solutions for DEC-POMDPs in detail. Four approaches have been chosen for a more detailed presentation, because they either have some very appealing properties or they introduce an interesting novel idea. Note that the approaches by Nair et al. [25], Emery-Montemerlo et al. [14] and Goldman and Zilberstein [20] are suitable for finite-horizon problems and the approaches by Bernstein et al. [8], Amato et al. [2] and Cogill et al. [10] are suitable for infinite-horizon problems. Some of the ideas carry over to the other case, but in general it is not straightforward to find similar algorithms, because some of the methods are limited to either the finite or infinite-horizon problem. Accordingly, it is also not reasonable to contrast the advantages and tradeoffs of the algorithms, because they simply solve two different problems.

4.1 Bounded Policy Iteration for Decentralized POMDPs

The dynamic programming algorithm presented in Section 3.1 suffered from the problem that the algorithm runs out of memory very quickly, because the number of evaluated policies grows double exponentially, even when pruning is used. The approach taken by Bernstein et al. [8] in 2005 uses finite-state controllers which provide a way to represent policies using only a fixed amount of memory. They consider infinite-horizon DEC-POMDPs with a discount factor γ, where $0 \leq \gamma < 1$.

Definition 44 (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.

The state of the controller depends on the observation sequence and then determines the action taken by the agent. As there are infinitely many possible observation sequences for the infinite-horizon case, but only a finite set of states, this representation of agent policies obviously leads to an approximation of the optimal solution: The agent will be in the same controller state for different observation sequences. But the great advantage is the memory-bounded way of representing the policy which allows for solving larger (and also infinite) problems.

Definition 45 (Independent Joint Controller) The local finite-state controllers of multiple agents determine the conditional distribution $P(\vec{a}, \vec{q}'|\vec{q}, \vec{o})$, which is denoted an independent joint controller.

This way of representing a joint policy is called **independent** joint controller, because there is no direct correlation between the agents. In particular, there is no correlated randomness although this is necessary for the optimal bounded-memory joint policy for some DEC-POMDPs.

4.1.1 The Utility of Correlation

Figure 11 shows a DEC-POMDP that illustrates the importance of correlated randomness in joint controllers for some problems.

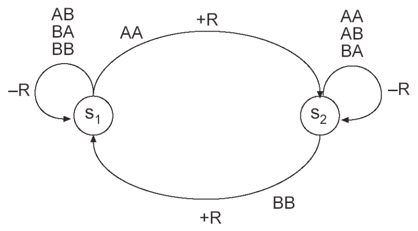


Figure 11: A DEC-POMDP Whose Optimal Memory-Bounded Joint Policy Requires Correlation (Courtesy of Daniel Bernstein).

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 choosing either action A or B according to a uniform distribution in each step. With an expected reward of $-\frac{R}{2}$ per time step this yields an expected long-term reward of $-\frac{R}{2(1-\gamma)}$.
- If the agents can correlate, thus use a common source of randomness, the best policy would be to execute the joint action AA with probability $\frac{1}{2}$ and BB with probability $\frac{1}{2}$. This leads to an expected long term reward of 0. Thus, the difference to the independent policy can be made arbitrarily large by increasing R.

This example shows the need for correlation when representing joint policies with bounded memory. Thus, the definition of a joint controller has to be extended to take this into account.

The importance of correlated randomness has already been mentioned by Peshkin et al. [32] in 2000. They presented an algorithm that uses gradient descent in policy space to find locally optimal solutions for multi-agent problems. Their approach is not presented in more detail in this study, because it considers the problem of multi-agent learning as opposed to multi-agent planning. However, the ideas presented in their paper had significant impact on the further development of multi-agent planning algorithms in last 5 years. But they did not provide a solution for the problem of correlated randomness with bounded memory.

4.1.2 Correlated Joint Controllers

To allow for correlation, Bernstein et al. [8] introduce a so-called **correlation device** which is an additional finite-state machine. All agents have access to extra signals from the device in each time step, but they cannot exchange information about each other through the correlation device.

Definition 46 (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 also consider the correlation device. The action taken by the agent and the state the controller transitions to is now additionally 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 47 (Correlated Joint Controller) A correlation device together with the local controllers for each agent form a joint conditional distribution $P(c', \vec{a}, \vec{q}' | c, \vec{q}, \vec{o})$, which is denoted 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 $c \in C$:

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

4.1.3 Bounded Policy Iteration

Every representation of a joint policy using finite-state controllers only uses finite memory. But the number of nodes used for each controller is still 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. Notice that the correlation device can only be used for correlated randomness if it has at least two nodes.

Initially an arbitrary distribution for the controller nodes is chosen. Obviously this corresponds to a possibly arbitrarily bad policy. Now the correlated joint controller is improved iteratively via a **bounded backup** until a local maximum is reached. In each iteration of the algorithm, either one agent i, along with a node q_i is chosen or one node c of the correlation device is chosen. Here, the improvement for a local controller is described: For each $o_i \in O_i$ new parameters for the conditional distribution $P(a_i, q_i'|c, q_i, o_i)$ are searched. This search assumes that the new controller is used from the second step on. It can be performed using a linear program that makes sure 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. See [8] for the details of the improvement step including the linear programs.

Bernstein et al. [8] prove that the 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 every 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 theoretical idea that larger numbers of controller nodes lead to higher values. As explained before, with a fixed number of controller nodes this algorithm can only compute approximate solutions. The interesting features are the usage 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.

4.2 From Decentralized BPI to Quadratically Constrained Linear Programming

4.2.1 Limitations of Bounded Policy Iteration

The BPI algorithm suffers from the same problem as most of the other approximate algorithms: They get stuck in local optima and thus their bound on value-loss is very weak. In particular, the BPI algorithm improves only one controller node at a time with a one-step look-ahead which leads to short-term improvements but generally does not lead to optimal controllers. Even though heuristics may be used to get unstuck from local maxima, this approach is limited to suboptimal controllers.

The limitations of BPI haven recently been illustrated by Amato et al. [2] using a simple example. As BPI for DEC-POMDPs is an extension of BPI for POMDPs, the example given in Figure 12 showing a simple POMDP for which BPI fails to find an optimal controller also illustrates the limitations of the multi-agent approach. 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, otherwise the state 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 an initial state distribution of being in either state with equal probability 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.

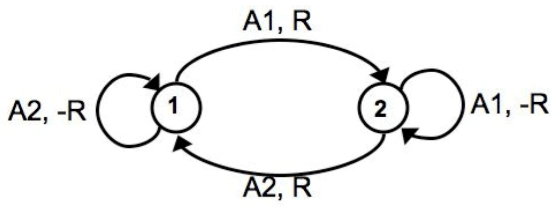


Figure 12: Simple POMDP for Which BPI Fails to Find an Optimal Controller (Courtesy of Christopher Amato).

The BPI algorithm starts with an arbitrary initial controller. If this controller is deterministic and chooses either action, say action 1, BPI will not converge to the optimal controller. The value of this initial controller is $R - \frac{\gamma R}{1-\gamma}$ in state 1 and $\frac{-R}{1-\gamma}$ 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 probability > 0 to action 2 raises the value for state 2 but lowers it for state 1. Because in every iteration the algorithm only accepts a new controller if the value increased or remained 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.

4.2.2 Optimal Fixed-Size Controller for POMDPs/DEC-POMDPs

In 2006, Amato et al. [2] have 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 to optimize 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. QCLPs must have a linear objective function, but may contain quadratic terms in the constraints. They are more difficult than a linear program, but simpler than a general non-linear program, because a large number of algorithms have been developed that can exploit the additional structure. The full description of the QCLP is given in Table 5.

Fortunately, an optimal solution of the QCLP results in an optimal stochastic controller for the given size and initial state distribution. This follows from the setup of the QCLP, i.e maximizing a given node at the initial state distribution while obeying the Bellman equation constraints. In this respect, the formulation of the optimization problem as a quadratically constrained linear program is a huge improvement over the original BPI algorithm which generally got stuck in local maxima. Additionally, the possibility to exploit an initial state distribution leads to further value improvement.

Unfortunately, the resulting problem is nonconvex, thus it may have multiple local as well as 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, the globally optimal solution can 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.

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:

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

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

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

Table 5: The quadratically constrained linear program for finding the optimal fixed-size controller. 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.

[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 compared to BPI, the hitherto state-of-the-art solution technique. The authors note that there is an interesting relationship between the nonlinear formulation of the problem and the performance of the used optimization algorithms. The detailed analysis of this relationship is subject to ongoing research.

Recently, Amato et al. [2] have extended their work and applied the same approach to DEC-POMDPs comparing the performance to the decentralized BPI algorithm as described in Section 4.1.3. Now, multiple controllers have to be taken into consideration at a time for the optimization. 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. The resulting nonlinear program can be converted to a quadratically constrained linear program. Now, the optimal solution of the QCLP provides optimal fixed-size controllers for a given DEC-POMDP. Obviously, trying to find the optimal solution for the QCLP one runs into the same problems as described before. Nevertheless, experimental results show that using nonlinear optimization techniques leads to higher valued controllers than those produced by BPI. Thus, the QCLP 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 [39] in 2005. 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 stochastic controllers used by Bernstein et al. [8] and Amato et al. [2]. As has been shown before, stochasticity is crucial when using finite memory control. Moreover, as shown in Section 4.1.1, even correlated randomness improves the performance of finite state controllers. Thus, it is obvious that an approach that only considers deterministic controllers leads to suboptimal solutions with respect to the whole space of possible (stochastic) controllers. Accordingly, the QCLP approach is significantly more sophisticated as it defines the optimal stochastic controller for a DEC-POMDP.

4.3 Approximate Dynamic Programming for Decentralized Systems

In 2003, de Farias and van Roy [11] 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. [10] in 2004 generalizes this algorithm by extending it to decentralized problems. Their approach is the first that uses 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, but 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. At first, the necessary definitions to introduce the linear program for centralized problems are introduced in Section 4.3.1. Section 4.3.2 then presents the approximate linear programming algorithm for decentralized problems. For the presentation of the formal models and algorithms some notation compared to [10] has been changed to facilitate a comparison with the DEC-POMDP model.

4.3.1 Dynamic Programming using Q Functions 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, the state transitions and the next state in the next time step is $y \in S$ 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

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

where α is a discount factor with $0 \leq \alpha < 1$. There is a single policy that minimizes the expected total discounted reward. Thus, the minimum cost-to-go J^* is unique and satisfies Bellmans's equation

$$J^*(s) = \min_{a \in A}[g(s,a) + \alpha \sum_{y \in S} P(y|s,a) J^*(s)] \tag{2}$$

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

$$Q^*(s,a) = g(s,a) + \alpha \sum_{y \in S} P(y|s,a)(\min_{w} Q^*(y,w)) \tag{3}$$

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 equations 2 and 3, i.e. finding the policy that hast minimal costs for all state-action pairs. This can be done by the linear program shown in Table 6.

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

Table 6: Linear Program for the Centralized Problem.

4.3.2 Approximate Dynamic Programming using Q Functions for Decentralized Problems

For the decentralized problem a specific structure is imposed onto the Q function of the problem. 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 $\mathbf{A} = A_1 \times ... \times A_n$, 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 $\mathbf{S} = S_1 \times ... \times S_m$.

For a centralized policy each action a_i would be based on the entire state $s = (s_1, ..., s_m)$. For a decentralized policy, each action a_i only depends on some specified subset of the state variables $s_1, ..., s_m$. This makes the problem decentralized, because one agent does not have all information available for its decision making. This corresponds to the notion of a mapping from observation sequences to actions in a DEC-POMDP where one agent also has to make a decision based on incomplete information.

If the problem is decentralized, the dependencies of the actions on the state variables are described by a so called **information structure**. To formalize this, some more definitions are needed:

- The set of variables that may be 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 $\mathbf{S}_i = \times_{j \in I_i} S_j$.
- A decentralized policy is a set of functions $\mu_i : \mathbf{S}_i \to A_i$ for $i = 1, ..., n$.

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 now 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 $\hat{Q}$ that shall approximate Q^* leads to a decentralized problem which can then be solved by the approximate linear programming algorithm shown in Table 7.

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

$$\begin{aligned} \text{maximize:} \quad & \sum_{s \in S} \sum_{a \in A} \omega(s) \hat{Q}(s,a) \\ \text{subject to:} \quad & \hat{Q}(s,a) \le g(s,a) + \alpha \sum_{y \in S} P(y|s,a) \hat{J}(y) \quad \text{for all } s \in S, a \in A \\ & \hat{J}(s) \le \hat{Q}(s,a) \quad \text{for all } s \in S, a \in A \end{aligned}$$

3. Let $\mu(s) = \text{argmin}_a \hat{Q}(s,a)$. This policy is decentralized with the desired information structure.

Table 7: Approximate Linear Programming Algorithm for the Decentralized Problem.

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 just 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. [10] 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 possibly. Unfortunately, bounds on the value loss due to the approximation in algorithm 2 are missing. The authors note that the weights $\omega(s)$ in the linear program have a huge impact on the quality of the policy obtained. But it remains an open question, how good weights (or how the best weights) can be found in general, and which bounds on the difference in value can be given with respect to 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. Unfortunately, the authors do not compare the empirical results of their algorithm with any optimal decentralized algorithm. Thus, a comparison between the bounded policy iteration algorithm and this algorithm could be useful to evaluate these two different approaches for approximating the optimal solution of decentralized systems.

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 are arriving 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 queue in every 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 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 is only dependent on its immediate neighbors. This is already an approximation, as an optimal decentralized solution would have to consider histories of observations. This can be illustrated by an easy example with five queues shown in Figure 13:

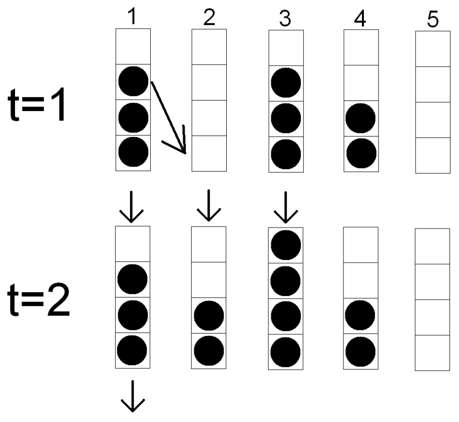


Figure 13: An Example of the Load Balancing Problem, Showing the Importance of an Observation History.

The arcs shall 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. Only considering 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. But 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 thus, it is obviously best to pass the job to queue number 4 instead of queue number 2. This lack of a history illustrates one weakness of the algorithm.

But actually, it would be possible to incorporate history into the algorithm, by adding states to the state space that serve as history states. A more important problem is the question, if approximating a decentralized problem by trying to approximate the optimal centralized solution is a reasonable approach at all. The problem is that the imposed information structure, i.e. the new Q_i's, might not make any sense for certain problems. In these cases, running the linear programming with those Q_i's actually might go into a completely wrong direction compared to an optimal decentralized solution. Indeed, for the load balancing problem, one can imagine that 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 attained. But obviously, this problem is baring a special structure in that the coupling between queues that are not immediate neighbors is weak. Thus, the decentralized version of the problem is very close to the centralized one and thus, it makes sense trying to approximate the centralized solution. For other problems, where the local view of an agent does not say anything about the global state of the problem, this approach might not be suitable. In this case, the value achieved by the algorithm could be arbitrarily bad.

4.4 Decomposition Techniques

The approximate algorithms presented in Section 4 scale better than the algorithms that find the optimal solution, but they still can't solve any but toy problems. As shown in Section 2.3.2 even problems with joint full observability are NEXP-complete as soon as two or more agents control a decentralized process together. The unfortunate jump in complexity is due to the various ways multiple agents might influence and are dependent on each other's states, observations and rewards. The last approximation technique presented in Section 4.3 exploits a specific structure of the decentralized process and lowers the running time of a corresponding algorithm by reducing/ignoring some of the interdependencies of the considered problem. A similar approach was taken by Goldman and Zilberstein [20] in 2005. The important observation is that a lot of decentralized problems bare some structure that has influence on the "level of decentralization" as discussed in Section 2.5.1. For example, as shown in Goldman and Zilberstein [19], 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. Obviously, not all decentralized problems fall into that special class, but a lot of them are only violating the 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 do justice to the decentralized part of the problem, Goldman and Zilberstein [20] introduce the notion of **communication-based** decomposition mechanisms, i.e. they compute a communication policy to let the single-agent MDPs be able to synchronize from time to time.

The formal models and algorithmic solutions of their paper are quite involved and their presentation is beyond this study. In the following section, the general idea is presented. It can be seen as another approach to approximate decentralized control problems. In Section 4.4.1, communication-based decomposition mechanisms are introduced. Section 4.4.2 presents the formal framework of decentralized semi-markov decision problems that serves as a basis for all algorithms. As it turns out, finding an optimal decomposition mechanism for the general case is still very hard. Therefore, the focus turns toward more specific sub-classes of decomposition mechanisms in Section 4.4.3 which also leads to more efficient algorithms.

4.4.1 Communication-based Decomposition Mechanisms

As discussed in Section 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, goal-oriented DEC-MPDs - which have a lower complexity - have been discussed in Section 2.5.2. Unfortunately, a goal-oriented behavior of multiple agents does not necessarily decompose nicely into multiple local goals. But only in this case, the complexity would automatically drop down to P. In their paper, 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, this sacrifices the overall attained value. But the profit is a lower **computational complexity** for the single agent problems. Obviously, this is only beneficial if the whole decomposition algorithm also has a low complexity, i.e. it is practical to apply it to hard decentralized problems. Furthermore, a decomposition mechanism is requested to be **complete**, that is, there exists a communication policy that guarantees that the agents reach one of the global goals whenever it is possible.

To formally phrase the separated phases of execution of the agents, temporally abstracted sequences of actions are used which are similar to options, including a deterministic single-agent policy, terminal actions which are communication messages and a set of initiation states:

Definition 48 (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 are the local states of agent i.
- τ is the number of time steps since the process was started.
- A_i are agent i's actions.
- Σ are the communication messages.
- I is the initiation set for this option, i.e. the set of states in which the agent can start this option.

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

Definition 49 (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. $CDM : S \rightarrow (Opt_1, Opt_2, ..., Opt_n)$.

Obviously, every mechanism is an approximation for the optimal joint policy of the decentralized problem. To find the best mechanism, one needs to search over all possible mechanisms (for two agents that is a 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. Unfortunately, as Section 4.4.2 shows, computing optimal mechanisms themselves is a very complex task. Thus, Section 4.4.3 turns towards an easier sub-class of mechanisms which are computationally tractable.

4.4.2 Decentralized Semi-Markov Decision Problems

Working with options instead of basic actions for the agents, the formal framework has to be modified slightly. Assuming an underlying decentralized MDP with direct communication, now a goal-oriented decentralized SMDP with direct communication ca 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 50 (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 $\langle DMC, Opt_1, Opt_2, ..., Opt_n, P^N, R^N \rangle$, where:

- DMC is a standard Dec-MDP-COM with its 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 that agent i can choose to perform, which terminate necessarily with a communication act: $opt_i = \langle \pi : S_i \times \tau \rightarrow A_i \cup \Sigma, I \subseteq S_i \rangle$.
- $P^N(s', t+N|s, t, opt_1, ..., opt_n)$ is the probability of the system reaching state s' after exactly N time units ($t + N \leq T$), when at least one option terminates (necessarily with a communication act).
- $R^N(s, t, opt_1, opt_2, ..., opt_n, s', t+N)$ is the expected reward obtained by the system N time steps after the agents started options $opt_1, ..., opt_n$ in state s at time t, when at least one of them has terminated its option with a communication act (resulting in the 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 (as opposed to joint fully observability) of the state. Their local policy is a mapping from global states to options. After the first option of one of the agents terminates, all agents communicate which leads to joint exchange of messages and thus full observability of the global state is again achieved. As the global state is revealed every time the agents have to choose a new option to execute, the choice of the local policies are no longer dependent on histories of observations. In fact, only the global state and the system time are important: $\delta_i : S \times \tau \rightarrow Opt_i$. A joint policy is now a tuple of local policies, one for each agent. Intuitively, solving optimally a decentralized semi-Markov decision problem with temporally abstracted actions (which is solving for an optimal mechanism) is equivalent to solving a multi-agent MDP.

Theorem 10 A GO-Dec-SMDP-COM is equivalent to a multi-agent MDP.

Proof: See Goldman and Zilberstein [20] for the detailed proof. □

At first sight, this result might be great news regarding the complexity of the resulting problem: As MMDPs are P-complete, these problems seem computationally tractable, as opposed to the original DEC-MDP, which is NEXP-complete. But unfortunately, 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. As one part of solving the problem is to compute the best option for each agent for each global state, this results in the same double exponential complexity of the problem as before. Naturally, any algorithm solving this problem optimally needs double exponential time in the worst case (c.f. the multi-step backup policy-iteration algorithm in [20]). As the number of possible options leads to the high complexity, from now on a restricted class of options is considered. Using these **goal-oriented options** significantly reduces the size of the search space and lowers the overall complexity.

4.4.3 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, considering the complete set of possible options for each agent is avoided. Unfortunately, not every goal-oriented DEC-MDP automatically decomposes into local goal-oriented behavior for each agent. As the question of how to generate local goals from a given global goal is beyond this study, it is assumed 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:

Definition 51 (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 $\delta : S_i \rightarrow A_i$ 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 when they are in some goal state component g_i, which incurs zero cost and has no transition effect, i.e., $\delta(g_i) = NOP$.

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 executed their options for k time steps they re-synchronize, even if they did not reach 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 by the agent independently by solving optimally its local process $MDP_i = (S_i, P_i, R_i, \hat{G}_i, T)$, which can be done in polynomial time. Intuitively this opens the possibility for polynomial time decomposition mechanisms. No longer all possible options have to be considered - instead the mechanism can focus on the possible local goal states. And 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 which 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 that strong (possibly only through the reward function) that every local goal assignment could lead to arbitrarily bad solutions. Note that this is the same problem as with the approximate dynamic programming approach discussed in Section 4.3. But for other problems, these techniques make a lot of sense and actually make solving some larger problems possible. In their paper, Goldman and Zilberstein [20] choose the meeting under uncertainty example to illustrate their approach. Obviously, in this domain it makes sense to assign local goals (which are meeting points on a grid) and to communicate from time to time (to get updated information where the other agent is). For these kind of problems, their algorithms prove to be suitable.

Furthermore, the authors present algorithms for the case, where the mapping from global states to single agent behaviors is already given and where the problem reduces to computing an optimal communication policy. This topic is also interesting with regard to the complexity of approximating decentralized control problems but beyond this study.

5 Summary of Algorithms for Optimal and Approximate Multi-Agent Planning

This section provides a summarizing presentation of all algorithms for multi-agent planning discussed in this paper. Their applicability, their computational complexity and their solution quality is presented. As explained before, it is not always possible to compare two different algorithms if they tackle two slightly different problems such as finite or infinite-horizon DEC-POMDPs. Most solution techniques do not generalize in a straightforward way to the other cases. Furthermore, for algorithms seeking an approximate solution, it is difficult to evaluate their quality without extensive experimental testing. Thus, in this section we do not try to identify a single best algorithm that is superior to all the others. Instead we want to illustrate the different approaches taken over the last 5 years and point out their advantages and disadvantages. At this point it is not possible to say which approach will lead to a breakthrough for solving DEC-POMDPs. As discussed in Section 2.4.4 the I-POMDP model [16] is even more expressive than the DEC-POMDP model. For algorithms developed for this model see [16], [12] and [13]. These algorithms are not discussed in this paper, because of the I-POMDP model's limitations due to complexity and applicability and its non-equivalence to the other models.

The following presentation of the algorithms is divided into two sections. At first, the algorithms solving finite-horizon DEC-POMDPs are presented. Subsequently, Section 5.2 presents all algorithms solving infinite-horizon DEC-POMDPs. Note that the **Maximum Problem Size** shown is the size of the largest problem that has been solved with the algorithm so far. Spmetimes, this does not necessarily mean that this is the limit of the algorithm. There are simply no other/better experimental results known.

5.1 Algorithms for Finite-Horizon DEC-POMDPs

Algorithm	Dynamic Programming for DEC-POMDPs
Authors	Bernstein et al. [6]
Solution Quality	Optimal
Solution Technique	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
Max. Problem Size	2 agents, 2 states, 2 actions, 2 observations up to horizon 4

Algorithm	Multi-Agent A*: Heuristic Search for DEC-PODMPs
Authors	Szer et al. [40]
Solution Quality	Optimal
Solution Technique	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 of the problem, runs out of time quickly (independent of the heuristic used) due to the double exponential dependence on the time horizon
Max. Problem Size	2 agents, 2 states, 3 actions, 2 observations up to horizon 4

Algorithm	Joint Equilibrium-based Search for Policies
Authors	Nair et al. [25]
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: The algorithm gets stuck in local optima
Max. Problem Size	2 agents, 2 states, 3 actions, 2 observations up to horizon 7

Algorithm	Approximate Solutions for POSGs with Common Payoffs
Authors	Emery-Montemerlo et al. [14]
Solution Quality	Approximate
Solution Technique	Usage of a series of smaller Bayesian games to approximate the POSG, usage of heuristics to estimate future reward, improving policy of one agent while holding the others fixed
Advantages	Large parts of the algorithm can run in parallel, scales moderately well with problem size
Disadvantages	Suboptimal Solutions: The algorithm gets stuck in local optima, high value loss to multiple steps of approximation
Max. Problem Size	2 agents, $>$20000 states, 5 actions, $>$ 30 observations , horizon $>$ 100

Algorithm	Communication-based Decomposition Mechanism
Authors	Goldman and Zilberstein [20]
Solution Quality	Approximate
Solution Technique	Decomposition into multiple MDPs and computation of a comuinaction policy
Advantages	Exploits structure of the process and takes into account the ”level of decentralization” of the problem, scales moderately well for some problems
Disadvantages	Limited to those problems that bear special structure

5.2 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 of controllers
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. [8]
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, guarantees monotonic value improvement for all initial state distributions at each iteration
Disadvantages	Suboptimal solutions: Gets stuck in local optima
Max. Problem Size	2 agents, 16 states, 5 actions, 4 observations

Algorithm	An Optimal Best-first Search Algorithm for Solving Infinite-Horizon DEC-POMDPs
Authors	Szer and Charpillet [39]
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 have significant drawbacks over stochastic finite state controllers
Max. Problem Size	2 agents, 16 states, 5 actions, 4 observations

Algorithm	Optimal Fixed-size Control of Decentralized POMDPs
Authors	Not published yet, cf. Amato et al. [2]
Solution Quality	Approximate
Solution Technique	Quadratically constrained linear programming, representation of policies using stochastic finite state controllers
Advantages	Defines optimal stochastic finite state controller for a given size, optimal solution can be found for some problems
Disadvantages	Suboptimal solutions: solution quality limited to the size of the controller, QCLPs are more complex than normal LPs, optimal solution cannot always be found
Max. Problem Size	2 agents, 16 states, 5 actions, 2 observations

Algorithm	Approximate Dynamic Programming for Decentralized Systems
Authors	Cogill et al. [10]
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 into the wrong direction with regard to the optimal decentralized solution
Max. Problem Size	2 agents, $>$ 100 states, 3 actions, 4 observations

6 Conclusion and Open Research Questions

We have examined the problem of decentralized control of multiple cooperative agents. Five different formal frameworks to model this problem were presented: DEC-POMDPs, MTDPs, DEC-POMDP-COMs, COM-MTDPs and I-POMDPs. We showed that the former 4 are all equivalent - they are all NEXP-complete. The orthogonal approach of I-POMDPs is even more expressive as it also allows for non-cooperative agents. But unfortunately, the general model is not computable due to a recursive structure of the agents' beliefs. Approximate algorithms for I-POMDPs, solving finitely nested I-POMDPs, remain intractable as they require double exponential time.

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

Both of the optimal algorithms introduced in Section 3 quickly run out of time or memory, a problem that is addressed by the latest approximation techniques presented in Section 4. All the algorithms establish certain performance bounds, but generally only local optimality can be guaranteed. Thus, the value loss could be arbitrarily large depending on the specific problem. For the Approximate Dynamic Programming approach, we discussed the problems for which this approach might be suitable and why it fails for other classes of problems. The same holds true for the decomposition techniques presented in Section 4.4. Here, the main idea was to decompose a DEC-MDP into multiple single agent behaviors, thus resulting in a lower complexity, and at the same time computing a communication policy, to let the single agents synchronize from time to time. This approach is limited to certain classes of problems, where local (goal-oriented) behavior of the agents is suitable.

So far, the lowest computational complexity applies to a specific sub-class of DEC-POMDPs: Goal-oriented DEC-MDPs with independent transitions and observations, with a single global goal and with uniform costs. As explained in Section 2.5.3, such problems are P-complete. Thus, if the considered problem posses this special structure, even larger instances can be solved efficiently. For less restricted classes, there are still many open research questions. For example, the complexity of goal-oriented behavior for DEC-POMDPs with independent transitions and observations is unknown. Another open question relates to the impact of correlated randomness in DEC-MDPs with independent transitions and observations. As shown in Section 4.1.1, correlation can have a huge impact on the value achieved by memory-bounded algorithms in decentralized settings. But a complete examination of its impact remains to be done.

The recent results in the field of algorithms aiming for approximate solutions for DEC-PODMPs have shown different potential for future improvements in scalability. For some approaches, it has been shown that they are still limited to toy problems. But for some algorithms, experimental results have shown that good solutions can be found for significantly large problems. The challenge for future research is to fully understand where complexity can be lowered without sacrificing value too much. On the one hand, history-dependent approaches generally run out of time or memory very quickly. On the other hand, memory-bounded algorithms that have better running-time guarantees, are able to find good solutions for some problems but they fail for other classes of problems. How to use the available memory most efficiently remains one of the main research challenges. Recently, researchers have started to combine approaches taken from different research areas such as planning, reinforcement learning, game theory, and operations research to tackle this problem. Thus, synergetic effects are very likely; we foresee the development of promising new approaches in the near future.

Acknowledgments

We thank Daniel Bernstein, Christopher Amato, Marek Petrik, Martin Allen, Raphen Becker, Daniel Szer, Victor Lesser, Claudia Goldman, Randy Cogill, Zinovi Rabinovich and Prashant Doshi for helpful discussions. Furthermore, we thank the authors of all 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] Allen, Martin; Goldman, Claudia V. & Zilberstein, Shlomo: Learning to Communicate in Decentralized Systems. In *Proceedings of the Wokshop on Multiagent Learning, Twentieth National Conference on Artificial Intelligence (AAAI)*, 2005.

[2] Amato, Christopher; Bernstein, Daniel S. & Zilberstein, Shlomo: Solving POMDPs Using Quadratically Constrained Linear Programs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, January 2006.

[3] Astrom, Karl J.: Optimal Control of Markov Decision Processes with Incomplete State Estimation. *Journal of Mathematical Analysis and Applications* 10:174–205, 1965.

[4] Becker, Raphen; Zilberstein, Shlomo; Lesser, Victor & Goldman, Claudia V.: Solving Transition Independent Decentralized Markov Decision Processes. *Journal of Artificial Intelligence Research (JAIR)* 22:423–255, 2004.

[5] Bernstein, Daniel S.: *Complexity Analysis and Optimal Algorithms for Decentralized Decision Making.* PhD thesis Department of Computer Science, University of Massachusetts Amherst, September 2005.

[6] Bernstein, Daniel S.; Zilberstein, Shlomo & Immerman, Neil: The Complexity of Decentralized Control of Markov Decision Processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, Stanford, California, June 2000.

[7] Bernstein, Daniel S.; Givan, Robert; Immerman, Neil & Zilberstein, Shlomo: The Complexity of Decentralized Control of Markov Decision Processes. *Mathematics of Operations Research* 27(4):819–840, November 2002.

[8] Bernstein, Daniel S.; Hansen, Eric A. & Zilberstein, Shlomo: Bounded Policy Iteration for Decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Edinburgh, Scotland, July 2005.

[9] Boutilier, Craig: Planning, Learning and Coordination in Multiagent Decision Processes. In *Proceedings of the Conference on Theoretical Aspects of Rationality and Knowledge*, pages 195–210, 1996.

[10] Cogill, Randy; Rotkowitz, Michael; van Roy, Benjamin & Lall, Sanjay: An Approximate Dynamic Programming Approach to Decentralized Control of Stochastic Systems. In *Proceedings of the Allerton Conference on Communication, Control, and Computing*, 2004.

[11] de Farias, Daniela Pucci & van Roy, Benjamin: The Linear Programming Approach to Approximate Dynamic Programming. *Operations Research* 51(6):850–865, 2003.

[12] Doshi, Prashant & Gmytrasiewicz, Piotr J.: A Particle Filtering Based Approach to Approximating Interactive POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, pages 969–974, Pittsburg, Philadelphia, July 2005.

[13] Doshi, Prashant & Gmytrasiewicz, Piotr J.: Approximating State Estimation in Multiagent Settings Using Particle Filters. In *Proceedings of the Fourth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Utrecht, Netherlands, July 2005.

[14] Emery-Montemerlo, Rosemary; Gordon, Geoff; Schneider, Jeff & Thrun, Sebastian: Approximate Solutions for Partially Observable Stochastic Games with Common Payoffs. In *Proceedings of the Third International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2004.

[15] Feng, Zhengzhu & Zilberstein, Shlomo: Region-Based Incremental Pruning for POMDPs. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI)*, Banff, Canada, 2004.

[16] Gmytrasiewicz, Piotr J. & Doshi, Prashant: A Framework for Sequential Planning in Multiagent Settings. *Journal of Artificial Intelligence Research (JAIR)* 24:49–79, 2005.

[17] Goldman, Claudia V. & Zilberstein, Shlomo: Mechanism Design for Communication in Cooperative Systems. In *Proceedings of the Fifth Workshop on Game Theoretic and Decision Theoretic Agents, Second International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Melbourne, Australia, July 2003.

[18] Goldman, Claudia V. & Zilberstein, Shlomo: Optimizing Information Exchange in Cooperative Multi Agent Systems. In *Second International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Melbourne, Australia, 2003.

[19] Goldman, Claudia V. & Zilberstein, Shlomo: Decentralized Control of Cooperative Systems: Categorization and Complexity Analysis. *Journal of Artificial Intelligence Research (JAIR)* 22:143–174, 2004.

[20] Goldman, Claudia V. & Zilberstein, Shlomo: Goal-Oriented Dec-MDPs with Direct Communication. Technical Report 04-44 Department of Computer Science, University of Massachusetts Amherst, 2005.

[21] Hansen, Eric A.: *Finite-Memory Control of Partially Observable Systems.* PhD thesis Department of Computer Science, University of Massachuetts Amherst, 1998.

[22] Hansen, Eric A.; Bernstein, Daniel S. & Zilberstein, Shlomo: Dynamic Programming for Partially Observable Stochastic Games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI)*, pages 709–715, San Jose, California, July 2004.

[23] Kaelbling, Leslie P.; Littmann, Michael L. & Cassandra, Anthony R.: Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence* 101(2):99–134, 1998.

[24] Kalai, Ehud & Lehrer, Ehud: Rational Learning Leads to Nash Equilibrium. *Econometrica* 1:1231–1240, 1993.

[25] Nair, Ranjit; Pynadath, David; Yokoo, Makoto; Tambe, Milind & Marsella, Stacy: Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings.

In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[26] Nair, Ranjit; Varakantham, Pradeep; Tambe, Milind & Yokoo, Makoto: Networked Distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, pages 133–139, Pittsburgh, Pennsylvania, July 2005.

[27] Ooi, James M. & Wornell, Gregory W.: Decentralized Control of a Multiple Access Broadcast Channel: Performance Bounds. In *Proceedings of the Thirty-Fifth Conference on Decision and Control*, pages 293–298, 1996.

[28] Osborne, Martin J. & Rubinstein, Ariel: *A Course in Game Theory.* The MIT Press, 1994.

[29] Papadimitriou, Christos H.: *Computational Complexity.* Addison Wesley, 1994.

[30] Papadimitriou, Christos H. & Tsitsiklis, John N.: Intractable Problems in Control Theory. *SIAM Journal on Control and Optimization* 24(4):639–654, 1986.

[31] Papadimitriou, Christos H. & Tsitsiklis, John N.: The Complexity of Markov Decision Processes. *Mathematics of Operations Research* 12(3):441–450, 1987.

[32] Peshkin, Leonid; Kim, Kee-Eung; Meuleau, Nicolas & Kaelbling, Leslie P.: Learning to Cooperate via Policy Search. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000.

[33] Puterman, Martin L.: *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., 1994.

[34] Pynadath, David V. & Tambe, Milind: The Communicative Multiagent Team Decision Problem: Anaylzing Teamwork Theories and Models. *Journal of Artificial Intelligence Research (JAIR)* 16:389–423, June 2002.

[35] Rabinovich, Zinovi; Goldman, Claudia V. & Rosenschein, Jeffrey S.: The Complexity of Multiagent Systems: The Price of Silence. In *Proceedings of the Second International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1102–1103, Melbourne, Australia, 2003.

[36] Russell, Stuart & Norvig, Peter: *Artificial Intelligence A Modern Approach.* Prentice Hall, 2001.

[37] Shen, Jiaying; Becker, Raphen & Lesser, Victor: Agent Interaction in Distributed MDPs and its Implications on Complexity. In *Proceedings of the Fifth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2006.

[38] Suzuki, Ichiro & Yamashita, Masafumi: Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *SIAM Journal on Computing* 28(4):1347–1363, 1999.

[39] Szer, Daniel & Charpillet, Francois: An Optimal Best-first Search Algorithm for Solving Infinite Horizon DEC-POMDPs. In *Proceedings of the Sixteenth European Conference on Machine Learning (ECML)*, Porto, Portugal, 2005.

[40] Szer, Daniel; Charpillet, Francois & Zilberstein, Shlomo: MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, Edinburgh, Scotland, July 2005.

[41] Tambe, Milind: Towards Flexible Teamwork. *Journal of Artificial Intelligence Research (JAIR)* 7:83–124, September 1997.

[42] Washington, Richard; Golden, Keith; Bresina, John; Smith, David E.; Anderson, Corin & Smith, Trey: Autonomous Rovers for Mars Exploration. In *Proceedings of the IEEE Aerospace Conference*, 1999.

[43] Zilberstein, Shlomo; Washington, Richard; Bernstein, Daniel S. & Mouaddib, Abdel-Illah: Decision-Theoretic Control of Planetary Rovers. In: *Plan-Based Control of Robotic Agents*, Michael Beetz. 2466:270–289, 2002.