Reinforcement Learning for DEC-MDPs with Changing Action Sets and Partially Ordered Dependencies

Thomas Gabel and Martin Riedmiller
Neuroinformatics Group, Department of Computer Science, Institute of Cognitive Science
University of Osnabrueck, 49069 Osnabrueck, Germany
{thomas.gabel|martin.riedmiller@uos.de}

ABSTRACT

Decentralized Markov decision processes are frequently used to model cooperative multi-agent systems. In this paper, we identify a subclass of general DEC-MDPs that features regularities in the way agents interact with one another. This class is of high relevance for many real-world applications and features provably reduced complexity (NP-complete) compared to the general problem (NEXP-complete). While this is a positive result, optimally solving larger-sized NP-hard problems is intractable. We propose to keep the learning as much decentralized as possible and employ a multi-agent reinforcement learning method to improve the agents' behavior online. Moreover, we suggest a restricted message passing scheme that notifies other agents about forthcoming effects on their state transitions. Although optimal solutions for the team can generally not be obtained in this way, we show empirically that acquiring approximate joint policies of high quality is possible within little time.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed AI

General Terms

Algorithms, Design, Theory

Keywords

Decentralized MDPs, Interaction, Communication

1. INTRODUCTION

Research on distributed control of cooperative multi-agent systems has received a lot of attention during the past years. Among the models discussed in the literature, the DEC-MDP framework [4], that is characterized by each agent having only a partial view of the global system state, has been frequently investigated. In this regard, it has been shown that the complexity of general DEC-MDPs is NEXP-complete, even for the benign case of two cooperative agents.

Decentralized decision-making is required in many real-life applications. Examples include distributed sensor networks or teams of autonomous robots for monitoring and/or exploring a certain environment, rescue operations where units must decide independently which sites to search and secure, or production planning and factory optimization where machines may act independently with the goal of achieving optimal joint productivity.

Being important for practice, the enormous computational complexity of solving DEC-MDPs conflicts with the fact that real-world tasks do typically have a considerable problem size. Taking this into consideration, we will pursue two lines of closely related investigations.

First, we will identify a subclass of general DEC-MDPs that features regularities in the way the agents interact with one another. For this class, we can show that the complexity of optimally solving an instance of such a DEC-MDP is provably lower (NP-complete) than the general problem. Moreover, we analyze methods for the agents to benefit from partially knowing about the state transition dependencies. To this end, we propose the use of a restricted message passing scheme that notifies other agents about forthcoming effects on their state transitions and we investigate its usefulness.

Second, it is our goal to tackle decentralized problems of moderate and larger size that are of practical interest. This involves settings with ten and more agents, which is why optimal solution methods can hardly be applied. We propose the utilization of a multi-agent reinforcement learning approach, where the agents are independent learners and do their learning online. The advantage is that the entire learning process is done in a completely distributed manner with each agent deciding on its own local action based on its partial view of the world state and on any other information it eventually gets from its teammates. So, our aim is to enable the agents to obtain high quality solutions in the vicinity of the optimal one as quickly and efficiently as possible.

In Section 2, we formally define the class of factored m-agent DEC-MDPs with changing action sets and partially ordered transition dependencies, which are in the center of our interest, and we analyze the complexity of that class. Section 3 elaborates on different possibilities for representing agent-specific policies and on how to best exploit known inter-agent dependencies. Subsequently, we provide the basics of our learning approach to acquire approximate joint policies using coordinated online multi-agent reinforcement learning. Finally, in Section 5 we show that numerous real-world problems from the field of production planning can be modelled using the class of DEC-MDPs specified. Moreover, empirical results for solving a number of scheduling problems are presented.

Padgham, Parkes, Mueller and Parsons (eds.): Proc. of AAMAS 2008, Estoril, May, 12-16., 2008, Portugal. **Cite as:** Reinforcement Learning for DEC-MDPs with (...), Thomas Gabel, Martin Riedmiller, Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systens, 2008.

2. PROBLEM DESCRIPTION

It is well-known that solving optimally decentralized cooperative Markov decision problems is extremely hard. Therefore, in this section we identify and characterize a subclass of general DEC-MDPs whose complexity is provably lower. Moreover, this class of problems is of high practical relevance since its characteristics are typical for many real-world applications.

2.1 DEC-MDP Framework

Basically, the subclass of problems we are focusing on in this paper may feature an arbitrary number of agents whose actions influence, besides their own, the state transitions of maximally one other agent in a specific manner. To formally define the problem settings of our interest, we embed them into the framework of decentralized Markov decision processes (DEC-MDP) by Bernstein et al. [4].

Definition 1. A factored m-agent DEC-MDP M is defined by a tuple $\langle Ag, S, A, P, R, \Omega, O\rangle$ with

- $Ag = \{1, \dots, m\}$ as the set of agents,
- S as the set of world states which can be factored into m components $S = S_1 \times \cdots \times S_m$ (the S_i belong to one of the agents each),
- $A = A_1 \times ... \times A_m$ as the set of joint actions to be performed by the agents ($a = (a_1, \dots, a_m) \in A$ denotes a joint action that is made up of elementary actions a_i taken by agent i),
- P as the transition function with $P(s'|s, a)$ denoting the probability that the system arrives at state s' upon executing a in s,
- R as the reward function with $R(s, a, s')$ denoting the reward for executing a in s and transitioning to s',
- $\Omega = \Omega_1 \times \cdots \times \Omega_m$ as the set of all observations of all agents ($o = (o_1, \dots, o_m) \in \Omega$ denotes a joint observation with o_i as the observation for agent i),
- O as the observation function that determines the probability $O(o_1, \dots, o_m|s, a, s')$ that agent 1 through m perceive observations o_1 through o_m upon the execution of a in s and entering s'.
- Moreover, M is jointly fully observable, i.e. the current state is entirely determined by the amalgamation of all agents' observations: if $O(o|s, a, s') > 0$, then $Pr(s'|o) = 1$.

We refer to the agent-specific components $s_i \in S_i$, $a_i \in A_i$, and $o_i \in \Omega_i$ as the local state, action, and observation of agent i, respectively. A joint policy π is a set of local policies $\langle \pi_1, \dots, \pi_m\rangle$ each of which is a mapping from agent i's sequence of local observations to local actions, i.e. $\pi_i : \overline{\Omega}_i \to A_i$. Simplifying subsequent considerations, we may allow each agent to fully observe its local state.

Definition 2. A factored m-agent DEC-MDP has local full observability, if for all agents i and for all local observations o_i there is a local state s_i such that $Pr(s_i|o_i) = 1$.

It must be noted that joint full observability and local full observability of a DEC-MDP do generally not imply full observability, which would allow us to consider the system as a single large MDP and to solve it with a centralized approach. Instead, typically vast parts of the global state are hidden from each of the agents.

We also need to characterize the problems of our interest with respect to the inter-agent dependencies in their reward, transition, and observation functions. A factored m-agent DEC-MDP is called *reward independent*, if there exist local functions R_1 through R_m, each depending on local states and actions of the agents only, as well as a function r that amalgamates the global reward value from the local ones, such that maximizing each R_i individually also yields a maximization of r. Throughout this paper, we will consider the global reward to be the sum of the local ones.

If, in a factored m-agent DEC-MDP, the observation each agent sees depends only on its current and next local state and on its action, then the corresponding DEC-MDP is called *observation independent*, i.e. $P(o_i|s, a, s', (o_1 \dots o_{i-1}, o_{i+1} \dots o_m) = P(o_i|s_0, s_i, a_i, s'_i)$. Then, in combination with local full observability, the observation-related components Ω and O are redundant and can be removed from Definition 1.

While the DEC-MDPs of our interest are observation independent and reward independent, they are *not* transition independent. That is, the state transition probabilities of one agent may very well be influenced by another agent. However, we assume that there are some regularities, to be discussed in the next section, that determine the way local actions exert influence on other agents' states.

2.2 Variable Action Sets

The following two definitions characterize the specific subclass of DEC-MDPs we are interested in. Firstly, we assume that the sets of local actions A_i change over time.

Definition 3. An m-agent DEC-MDP with factored state space $S = S_1 \times \cdots \times S_m$ is said to feature *changing action sets*, if the local state of agent i is fully described by the set of actions currently selectable by that agent ($s_i = A_i \setminus \{\alpha_0\}$) and A_i is a subset of the set of all available local actions $\mathcal{A}_i = \{\alpha_0, \alpha_{i1} \dots \alpha_{ik}\}$, thus $S_i = \mathcal{P}(\mathcal{A}_i \setminus \{\alpha_0\})$. Here, α_0 represents a null action that does not change the state and is always in A_i. Subsequently, we abbreviate $\mathcal{A}_i^r = \mathcal{A}_i \setminus \{\alpha_0\}$.

Concerning state transition dependencies, one can distinguish between dependent and independent local actions. While the former influence an agent's local state only, the latter may additionally influence the state transitions of other agents. As pointed out, our interest is in non-transition independent scenarios. In particular, we assume that an agent's local state can be affected by an arbitrary number of other agents, but that an agent's local action affects the local state of maximally one other agent.

Definition 4. A factored m-agent DEC-MDP is said to have *partially ordered transition dependencies*, if there exist functions σ_i for each agent i with

1. $\sigma_i : \mathcal{A}_i^r \to Ag \cup \{\emptyset\}$ and
2. $\forall \alpha \in \mathcal{A}_i^r$ the directed graph $G_\alpha = (Ag \cup \{\emptyset\}, E)$ with $E = \{(j, \sigma_j(\alpha))|j \in Ag\}$ is acyclic and contains a path of length m

and it holds

$$\begin{aligned} & P(s_i'|s,(a_1\dots a_m),(s_1'\dots s_{i-1}',s_{i+1}'\dots s_m')) \\ = \quad & P(s_i'|s_i,a_i,\{a_j\in\mathcal{A}_j|i=\sigma_j(a_j),j\neq i\}) \end{aligned}$$

The influence exerted on another agent always yields an extension of that agent's action set: If $\sigma_i(\alpha) = j$, i takes local action α, and the execution of α has been finished, then α is added to $A_j(s_j)$, while it is removed from $A_i(s_i)$.

That is, the σ_i functions indicate whose other agent' state is affected when agent i takes a local action. Further, condition 2 in Definition 4 implies that for each local action α there is a total ordering of its execution by the agents. While these orders are total, the global order in which actions are executed is only partially defined by that definition and subject to the agents' policies.

The following lemma states that for the problems considered any local action may appear only once in an agent's action set and, thus, may be executed only once.

Lemma 1. In a factored m-agent DEC-MDP with changing action sets and partially ordered transition dependencies it holds: $\forall i \in Ag$, $\forall \alpha \in \mathcal{A}_i^r$, $\forall t \in \{1\dots T\}$ and $\forall \overline{s}_i = (s_i^1 \dots s_i^t)$: If there is a t_a $(1 \leq t_a < T)$ with $\alpha \in s_i^{t_a}$ and a t_b $(t_a < t_b \leq T)$ with $\alpha \notin s_i^{t_b}$, then $\forall \tau \in \{t_b \dots T\} : \alpha \notin s_i^{\tau}$.

Proof. By contradiction. Let t_a, t_b and $\alpha \in \mathcal{A}_i^r$ with $\alpha \in s_i^{t_a}$, $\alpha \notin s_i^{t_b}$ as required by the lemma. Then, there is an $f \in Ag \cup \{\emptyset\}$ such that $\sigma_i(\alpha) = f$, and hence edge $(i, f) \in E$. Assume there is a $t_c > t_b$ with $\alpha \in s_i^{t_c}$. Then, there is an agent $j \in Ag$ such that $\sigma_j(\alpha) = i$, and hence edge $(j, i) \in E$. Since graph G_α is acyclic, it follows that the time when α joins s_f is after the time that α leaves s_j, i.e. $t_a > t_c$. Contradiction. □

2.3 Implications on Complexity

While it has been proven that the complexity of solving general DEC-MDPs is NEXP-complete [4, 10], several authors have identified subclasses of the general problem that provably yield lower (NP-complete) complexity (e.g. [3, 9]). As shown by Shen et al. [13], a key factor that determines whether the problem complexity is reduced to NP-completeness is whether the agents' histories can be compactly represented. In particular, there must exist an encoding function $Enc_i : \overline{\Omega}_i \to E_i$ such that

1. a joint policy $\pi = \langle \pi_1 \dots \pi_m \rangle$ with $\pi_i : E_i \to \mathcal{A}_i$ is capable of maximizing the global value and
2. the encoding is polynomial, meaning that $|E_i| = o(|S|^{c_i})$.

For our class of factored m-agent DEC-MDPs with changing action sets and partially ordered transition dependencies we can define an encoding that adheres to both of these conditions, thus showing that those problems are NP-complete.

The interaction history of a DEC-MDP is the sequence of local observations $\overline{o}_i \in \overline{\Omega}_i$ which in our case correspond to the history of local states $\overline{s}_i \in \overline{S}_i = \bigcup_{t=1}^{T} S_i$, since we assume local full observability (where T refers to the finite problem horizon and $S_i = \mathcal{P}(\mathcal{A}_i^r)$ as before).

Definition 5. Given a local action set $\mathcal{A}_i = \{\alpha_0 \dots \alpha_k\}$ and a history $\overline{s}_i = (s_i^1 \dots s_i^t) \in \overline{S}_i$ of local states of agent i, the encoding function is defined as $Enc_i : \overline{S}_i \to E_i$ with $E_i = C_{\alpha_1} \times \cdots \times C_{\alpha_k}$ and $C_{\alpha_j} = \{0, 1, 2\}$. And it holds $Enc_i(\overline{s}_i) = (c_{i,\alpha_1} \dots c_{i,\alpha_k}) \in E_i$ with

$$c_{i,\alpha_j} = \begin{cases} 0 & \text{if } \nexists \tau \text{ with } \alpha_j \in s_i^{\tau} \\ 1 & \text{if } \alpha_j \in s_i^t \\ 2 & \text{else} \end{cases}$$

Basically, the encoding guarantees that each agent knows whether some local action has not yet been in its action set, is currently in its action set, or had been in its action set. Proving that this encoding is capable of representing the optimal policy and showing that it is a polynomial encoding, we can conclude that the subclass of DEC-MDPs we identified is NP-complete.

Lemma 2. Enc_i provides a polynomial encoding of agent i's observation history.

Proof. Because of $S_i = \mathcal{P}(\mathcal{A}_i^r)$, it holds $|S_i| \in O(2^k)$, if $|\mathcal{A}_i| = k$. The encoding E_i provided by Enc_i is of size $|E_i| \in O(3^k)$. Since $O(3^k) = O((2^n)^{log_2 3})$, this encoding is polynomial in the size of S_i, i.e. $|E_i| \in O(|S_i|^{c_i})$ with constant $c_i = log_2 3$. □

Lemma 3. Enc_i provides an encoding of agent i's observation history such that a joint policy $\pi = \langle \pi_1 \dots \pi_m \rangle$ with $\pi_i : E_i \to \mathcal{A}_i$ is sufficient to maximize the global value.

Proof. Let $\pi = \langle \pi_1 \dots \pi_m \rangle$ be a set of local policies for a factored m-agent DEC-MDP with changing action sets and partially ordered transition dependencies. Because each local action can be performed only once by each agent (Lemma 1), π defines a total execution order over the elements of $\mathcal{A}_i^r$ for all agents i. Thus, each local policy must allow for realizing any possible such order. Since E_i contains the historical information which actions $\alpha \in \mathcal{A}_i^r$ have already been executed, each π_i can select the next $\alpha_x \in \mathcal{A}_i$ to be executed depending on e_i: If $\alpha_x \in s_i$ (s_i as current action set), i.e. $e_{i,x} = 1$, then that action is taken, otherwise (α_x is not yet in the action set, $c_{i,x} = 0$) the null action α_0 is selected. □

Because deciding a polynomially encodable DEC-MDP is NP-complete [13], solving a factored m-agent DEC-MDP with changing action sets and partially ordered transition dependencies is so, too.

2.4 Example Applications

Distributed problem solving in practice is often characterized by a larger number of agents involved ($m > 2$) and by a factored system state description where the agents base their decisions on local observations. Also, our assumptions that local actions may influence the state transitions of maximally one other agent and that any action has to be performed only once are frequently fulfilled. Sample real-world applications include scenarios from manufacturing, production planning, or assembly line optimization, where typically the production of a good involves a number of processing steps that have to be performed in a specific order. In a factory, however, usually a variety of products is assembled concurrently, which is why an appropriate sequencing and scheduling of single operations is of crucial importance for overall performance. Our class of factored m-agent DEC-MDPs with changing action sets and partially ordered transition dependencies covers a variety of such scheduling problems, for example flow-shop and job-shop scheduling scenarios [12], even scheduling problems with recirculating tasks can be modelled. In the evaluation part of this paper (Section 5), we will focus in depth on larger job-shop scheduling problems where a number of tasks have to be allocated to a limited number of resources in such a manner that one or more objectives are optimized.

3. RESOLVING TRANSITION DEPENDENCIES

The definition of an encoding function of an agent's interaction history as provided in Section 2 represents just one way to exploit the regularities in the transition dependencies of the class of DEC-MDPs we identified. Next, we suggest two approaches that also aim at utilizing these properties and that are at least capable of providing approximate solutions.

3.1 Reactive Policies and Their Limitations

An agent taking its action based solely on its most recent local observation $s_i \subseteq \mathcal{A}_i$ will in general not be able to contribute to optimal corporate behavior. In particular, it will have difficulties in assessing the value of taking its idle action α_0. Taking α_0, the local state remains unchanged except when it is influenced by dependent actions of other agents. Since a purely reactive agent, however, has no information related to other agents and dependencies at all, it is incapable of properly distinguishing when it is favorable to remain idle and when not. For these reasons, we exclude α_0 from all $\mathcal{A}_i$ for purely reactive agents.

Definition 6. For a factored m-agent DEC-MDP with changing action sets and partially ordered transition dependencies, a *reactive policy* $\pi^r = \langle \pi_1^r \dots \pi_m^r \rangle$ consists of m reactive local policies with $\pi_i^r : S_i \to \mathcal{A}_i^r$ where $S_i = \mathcal{P}(\mathcal{A}_i^r)$.

That is, purely reactive policies always take an action $\alpha \in \mathcal{A}_i(s_i) = s_i$ (except for $s_i = \emptyset$), even if it was more advisable to stay idle and wait for a transition from s_i to some $s_i' = s_i \cup \{\alpha'\}$ induced by another agent, and then execute α' in s_i'.

3.2 Awareness of Dependencies

In Definition 5, we stated that the probability that agent i's local state moves to s_i' depends on that agent's current local state s_i, its action a_i, as well as on the set $\{a_j \in \mathcal{A}_j | i = \sigma_j(a_j), i \neq j\} = \Delta_i$, i.e. on the local action of all agents that may influence agent i's transition. Let us for the moment assume that agent i always knows Δ_i. Then, all transition dependencies would be resolved as they would be known to each agent. As a consequence, all local transitions would be Markovian and local states would represent a sufficient statistic for each agent to behave optimally.

Unfortunately, fulfilling the assumption of all Δ_i to be known conflicts with the idea of decentralized decision-making. In fact, knowing σ_j and relevant actions a_j of other agents, enables agent i to determine their influence on its local successor state and to best select its local action a_i. This action, however, generally also influences another agent's transition and, hence, that agent's action choice if it knows its set Δ_j, as well. Thus, it can be seen that even in the benign case of a two-agent system, there may be circular dependencies, which is why knowing all Δ_i entirely would only be possible if a central decision-maker employing a joint policy and deciding for joint actions is used.

Enhancing a Reactive Agent

Although knowing Δ_i in general is not feasible, we may enhance the capabilities of a reactive agent i by allowing it to get at least some partial information about this set. For this, we extend a reactive agent's local state space from $S_i = \mathcal{P}(\mathcal{A}_i^r)$ to $\hat{S}_i$ such that for all $\hat{s}_i \in \hat{S}_i$ it holds $\hat{s}_i = (s_i, z_i)$ with $z_i \in \mathcal{P}(\mathcal{A}_i^r \setminus s_i)$. So, z_i is a subset of the set of actions currently *not* in the action set of agent i.

Definition 7. Let $1 \dots m$ be reactive agents acting in a DEC-MDP, as specified in Definition 4, whose local state spaces are extended to $\hat{S}_i$. Assume that current local actions $a_1 \dots a_m$ are taken *consecutively*. Given that agent j decides for $a_j \in A_j(s_j)$ and $\sigma_j(a_j) = i$, let also s_i be the local state of i and $\hat{s}_i$ its current extended local state with $\hat{s}_i = (s_i, z_i)$. Then, the transition dependency between j and i is said to be *resolved*, if $z_i := z_i \cup \{a_j\}$.

The resolving of a transition dependency according to Definition 7 corresponds to letting agent i know some of those current local actions of other agents by which the local state of i will soon be influenced[1]. Because, for the class of problems we are dealing with, inter-agent interferences are always exerted by changing (extending) another agent's action set, in this way agent i gets to know which further action(s) will soon be available in its action set. By integrating this piece of information into i's extended local state description $\hat{S}_i$, this agent obtains the opportunity to willingly stay idle (execute α_0) until the announced action $a_j \in z_i$ enters its action set and can finally be executed.

Obviously, $\hat{S}_i$ has the same size as E_i ($|\hat{S}_i| \in O(3^k)$ with $k = |\mathcal{A}_i^r|$). Given the above definition of z_i, however, it is clear that a policy $\hat{\pi} = \langle \hat{\pi}_1 \dots \hat{\pi}_m \rangle$ whose components $\hat{\pi}_i$ are defined over extended local state spaces $\hat{S}_i$ is generally not an encoding for agent i's interaction history (except for the theoretical case when Δ_i is entirely known) and thus not capable of optimizing the global value. Yet, because local states $\hat{s}_i$ are extended by information relating to transition dependencies between agents, such policies are normally more capable than purely reactive ones, since at least some information about future local state transitions induced by teammates can be regarded during decision-making.

Since, as argued before, in practice not all transition dependencies can be resolved, z_i will adopt only few values from its domain, which is why $O(3^k)$ represents the worst-case size of $\hat{S}_i$, but generally $\hat{S}_i$ will be sized only marginally beyond $O(2^k)$.

The notification of agent i, which instructs him to extend its local state component z_i by a_j, may easily be realized by a simple message passing scheme (assuming cost-free communication between agents) that allows agent i to send a single directed message to agent $\sigma_i(\alpha)$ upon the local execution of α. Obviously, this kind of partial resolving of transition dependencies is particularly useful in applications where the execution of atomic actions takes more than a single time step and where, hence, decision-making proceeds asynchronously across agents. Under those conditions, up to half of the dependencies in Δ_i (over all i) may be resolved.

[1] To fully comply with Definition 4 it must be noted that the influence one dependent action a_j now exerts on agent i yields not just an extension of i's action set ($s_i = A_i(s_i) := A_i(s_i) \cup \{a_j\}$), but also a removal from z_i, i.e. $z_i := z_i \setminus \{a_j\}$.

4. APPROXIMATE SOLUTIONS

Solving a DEC-MDP optimally is NEXP-hard and intractable for all except the smallest problem sizes. Unfortunately, the fact that the subclass of DEC-MDPs we identified in Section 2 is in NP and hence simpler to solve, does not rid us from the computational burden implied, in particular when intending to tackle problems of larger size that are of relevance for practical problems.

Given that fact, our goal is not to develop yet another optimal solution algorithm that is applicable to small problems only, but to look for a technique capable of quickly obtaining approximate solutions in the vicinity of the optimum.

Reinforcement learning (RL) has proven to be a suitable tool for decision-making under uncertainty and is also usable for acquiring approximate policies in decentralized MDPs. Here, however, some of the properties of standard RL algorithms, such as convergence guarantees, get lost due to inter-agent dependencies, so that care must be taken in the design of the learning algorithm. In contrast to offline planning algorithms, reinforcement learning allows for a real decentralization of the problem employing independently learning agents. In the remainder of this section, we outline the basic characteristics of our approach to applying reinforcement learning in distributed settings aiming at the acquisition of joint policies for m-agent factored DEC-MDPs with changing action sets and partially ordered transition dependencies.

4.1 Challenges for Independent Learners

Boutilier [5] pointed out that any multi-agent system can be considered as a single MDP when adopting an external point of view. The difficulties induced when taking the step towards decentralization can be grouped into three categories. First, in addition to the (single-agent) temporal credit assignment problem, the multi-agent credit assignment problem arises, which corresponds to answering the question of whose agent's local action contributed how much to a corporate success. To this end, we consider reward independent DEC-MDPs only (see Section 2.1) with the global reward being the sum of local ones.

A second challenge is represented by the agents' uncertainty regarding the other agents' policies pursued during learning. To sidestep that problem, we revert to an inter-agent coordination mechanism introduced in [11]. Here, the basic idea is that each agent always optimistically assumes that all other agents behave optimally (though they often will not, e.g. due to exploration). Updates to the value function and policy learned are only done when an agent is certain that a superior joint action has been executed. Since the performance of that coordination scheme quickly degrades in the presence of noise, we focus on deterministic DEC-MDPs in the remainder of the paper.

Third, the subclass of DEC-MDPs identified in Section 2 has factored state spaces providing each agent with (locally fully observable) state perceptions. Since the global state is unknown, each agent must necessarily remember the full history of local states to behave optimally, which quickly becomes intractable even for toy problems. In Section 2.3, however, we have proposed an encoding of that history that is polynomial in $|S_i|$ and allows for optimality, too. Furthermore, in Section 3.2 we have suggested a message passing scheme that enables the learners to inform other agents about expected state transitions and thus enhances the capabilities of a purely reactive agent. Although, in this way the optimal policy can generally not be represented, the functioning of the encoding function Enc_i is approximately emulated while the state information to be memorized by an agent is not increased as much as in the encoding approach.

4.2 Joint Policy Acquisition with RL

We let the agents acquire their local policies independently of the other agents by repeated interaction with the DEC-MDP and concurrent evolvement of their policies. Our learning approach is made up of alternating data collection and learning stages that are being run concurrently within all agents. At its core, a fitted Q iteration (FQI) algorithm [7] is used that allows the agents to determine a value function over their local state-action spaces. Next, we provide a brief explanation of that learning method only, a detailed description can be found in the related publication [8].

4.2.1 Data Collection

Our multi-agent extension of FQI denotes a batch-mode RL algorithm where agent i computes an approximation of the optimal policy, given a finite set $\mathbb{T}$ of local four-tuples [7]. $\mathbb{T} = \{(s_i^k, a_i^k, r_i^k, {s_i'}^k) | k = 1 \dots p\}$ can be collected in any arbitrary manner (e.g. by pursuing an ϵ-greedy policy) and contains agent-specific local states s_i^k, local actions $a_i^k \in A_i(s_i^k) = s_i^k \subseteq \mathcal{A}_i$, corresponding rewards r_i^k, as well as local successor states ${s_i'}^k$ entered. If the final state of the DEC-MDP has been reached ($A_i(s_i) = \emptyset$ for all i), the system is reset to its starting state (beginning of what we call a new training episode), and if a sufficient amount of tuples has been collected, the learning stage (4.2.2) is entered.

4.2.2 Applying Fitted Q Iteration

Given a set $\mathbb{T}$ and a regression algorithm, FQI iteratively computes an approximation $\tilde{Q}_i : S_i \times \mathcal{A}_i \rightarrow \mathbb{R}$ of the optimal state-action value function, from which a policy $\tilde{\pi}_i : S_i \rightarrow \mathcal{A}_i$ can be induced by greedy exploitation via $\tilde{\pi}_i(s_i) = \arg\max_{\alpha \in A_i(s_i)} \tilde{Q}_i(s_i, \alpha)$. After having initialized $\tilde{Q}_i$ and a counter q to zero, FQI repeatedly processes the following two steps until some stop criterion becomes true:

1. construct a training set $\mathbb{F}_i$ as input for the regression algorithm according to $\mathbb{F}_i = \{(v^k, w^k) | k = 1 \dots p\}$, where $v^k = (s_i^k, a_i^k)$ and the target values w^k are calculated using the well-known Q learning [16] update rule, $w^k = r_i^k + \gamma \max_{\alpha \in s_i^k} \tilde{Q}_i^q({s_i'}^k, \alpha)$,
2. use the regression algorithm and $\mathbb{F}_i$ to induce a new approximation $\tilde{Q}_i^{q+1} : S_i \times \mathcal{A}_i \rightarrow \mathbb{R}$, and increment q.

For the second step, we primarily employ neural networks, but other regression methods like support vector regression or decision trees may be used as well.

4.2.3 Optimistic Inter-Agent Coordination

For the multi-agent case, we modify step 2 of applying FQI: Agent i creates a reduced (optimistic) training set $\mathbb{O}_i$ such that $|\mathbb{O}_i| \leq |\mathbb{F}_i|$. Given a deterministic environment and the resetting mechanism during data collection (4.2.1), the probability that agent i enters some s_i^k more than once is larger than zero. Hence, if a certain action $a_i^k \in A_i(s_i^k)$ has been taken multiple times in s_i^k, it may—because of differing local actions selected by other agents—have yielded very different rewards and local successor states for i. Instead of considering all tuples from $\mathbb{T}_i$, only those are used for creating $\mathbb{O}_i$ that have resulted in maximal expected rewards. This means, we assume that all other agents take their best possible local action, which are, when combined with a_i^k, most suitable for the current state. Accordingly, we compute the optimistic target values w^k for a given local state-action pair $v^k = (s_i^k, a_i^k)$ according to

$$w^k := \max_{\substack{(s_i^k, a_i^k, r_i^k, {s_i'}^k) \in \mathbb{T}, \\ (s_i^k, a_i^k) = v^k}} \left(r_i^k + \gamma \max_{\alpha \in s_i^k} \tilde{Q}_i^k({s_i'}^k, \alpha) \right)$$

Consequently, $\mathbb{O}_i$ realizes a partitioning of $\mathbb{T}_i$ with respect to identical values of s_i^k and a_i^k, and w^k is the maximal sum of the immediate rewards and discounted expected costs over all tuples $(s_i^k, a_i^k, \cdot, \cdot) \in \mathbb{T}$.

5. EXPERIMENTS

As pointed out, our class of factored m-agent DEC-MDPs with changing action sets and partially ordered transition dependencies is of relevance to various practical applications. In this section, we show how this class of DEC-MDPs can be utilized for modeling production planning problems. We focus on job-shop scheduling (JSS) benchmarks, though other manufacturing problems could be covered as well.

5.1 Scheduling Problems

The goal of scheduling is to allocate a specified number of jobs to a limited number of resources (also called machines) such that some objective is optimized. In job-shop scheduling, n jobs must be processed on m machines in a pre-determined order. Each job j consists of ν_j operations $o_{j,1} \dots o_{j,\nu_j}$ that have to be handled on a certain resource $\varrho(o_{j,k})$ for a specific duration $\delta(o_{j,k})$. A job is finished after its last operation has been entirely processed (completion time f_j). In general, scheduling objectives to be optimized all relate to the completion time of the jobs. In this paper, we concentrate on the goal of minimizing maximum makespan ($C_{max} = max_j\{f_j\}$), which corresponds to finishing processing as quickly as possible, since most publications on results for JSS benchmarks focus on that objective, too.

Solving JSS problems is well-known to be NP-hard. Over the years, numerous benchmark problem instances of varying sizes have been proposed and have been frequently used to compare different solution approaches. A collection of sample problems is provided by the OR Library [1]. A common characteristic of those JSS benchmarks is that usually no recirculation of jobs is allowed, i.e. that each job must be processed exactly once on each resource ($\nu_j = m$). For more basics on scheduling, the reader is referred to [12].

JSS problems can be modelled using factored m-agent DEC-MDPs with changing action sets and partially ordered transition dependencies:

- The world state can be factored: We assume that to each of the resources one agent i is associated whose local action is to decide which waiting job to process next.

- The local state of i can be fully described by the changing set of jobs currently waiting for further processing. Since choosing and executing a job represents a local action (i.e. $\mathcal{A}_i^r$ is the set of jobs that must be processed on resource i), it holds $S_i = \mathcal{P}(\mathcal{A}_i^r)$.

- After having finished an operation of a job, this job is transferred to another resource, which corresponds to influencing another agent's local state by extending that agent's action set.

- The order of resources on which a job's operation must be processed is given in a JSS problem. Therefore, we can define $\sigma_i : \mathcal{A}_i^r \to Ag \cup \{\emptyset\}$ (cf. Definition 4) for all agents/resources i as

  $$\sigma_i(\alpha) = \begin{cases} \emptyset & \text{if } k = \nu_\alpha \\ \varrho(o_{\alpha,k+1}) & \text{else} \end{cases}$$

  where k corresponds to the number of that operation within job α that has to be processed on resource i, i.e. k such that $\varrho(o_{\alpha,k}) = i$.

- Given the no recirculation property from above and the definition of σ_i, the directed graph G_α from Definition 4 is indeed acyclic with a path of length m.

For more low-level details on our approach to solving JSS problems in a decentralized manner as well as on parameter settings of the RL algorithm involved, we refer to [8].

5.2 Experiment Outline

Classically, JSS problems are solved in a centralized manner, assuming that a central control over the process can be established. From a certain problem size on, however, the NP-hardness of the problem precludes the search for an optimal solution even for a centralized approach. That is why frequently dispatching priority rules are employed that take local dispatching decisions in a reactive and independent manner (the FIFO rule is a well-known example).

In the following experiment, however, a comparison to alternative scheduling methods is only our secondary concern. For comparison, we just provide results for two of the best-performing priority rules (SPT chooses operations with shortest processing time δ next and AMCC makes use of knowing the global system state), as well as the theoretic optimum, representing a lower bound, as it may be found by a centralized brute-force search.

Our primary concern is on an analysis of the three approaches discussed in this paper. We compare agents that independently learn

- purely reactive policies π_i^r (see Section 3.1) defined over $S_i = \mathcal{P}(\mathcal{A}_i^r)$ that never remain idle when their action set is not empty **[RCT]**,

- reactive policies $\hat{\pi}_i$ that are partially aware of their dependencies on other agents (being notified about forthcoming influences exerted by other agents) **[COM]**,

- policies $\pi_i : E_i \to \mathcal{A}_i$ where E_i is an encoding of that agent's observation history $\overline{S}_i$ according to Definition 5 **[ENC]**.

In JSS problems, it typically holds that $d(o_{j,k}) > 1$ for all j and k. Since most of such durations are not identical, decision-making usually proceeds asynchronously across agents. We assume that a COM-agent i sends a message to agent $\sigma_i(\alpha)$ when it starts the execution of an operation from job α, announcing to that agent the arrival of α, whereas the actual influence on agent $\sigma_i(\alpha)$ (its action set extension) occurs $d(o_{\alpha,\cdot})$ steps later (after $o_{\alpha,\cdot}$ has been finished).

Classes of Schedules

For a problem with m resources and n jobs consisting of m operations each, there are $(n!)^m$ possible schedules (also called set of active schedules, $\mathbb{S}_a$). Considering such a problem as a DEC-MDP, this gives rise to, for example, about $1.4 \cdot 10^{17}$ possible joint policies for $m = n = 6$.

Considering purely reactive agents, the number of policies/schedules that can be represented is usually dramatically reduced. Unfortunately, only schedules from the class of non-delay schedules $\mathbb{S}_{nd}$ can be created by applying reactive policies. Since $\mathbb{S}_{nd} \subseteq \mathbb{S}_a$ and it is known that the optimal schedule is always in $\mathbb{S}_a$ [12], but not necessarily in $\mathbb{S}_{nd}$, RCT-agents can at best learn the optimal solution from $\mathbb{S}_{nd}$. By contrast, learning with ENC-agents, in principle the optimal solution can be attained, but we expect that the time required by our learning approach for this to happen will increase significantly[2].

We hypothesize that the awareness of inter-agent dependencies achieved by partial dependency resolvements via communication may in fact realize a good trade-off between the former two approaches. On the one hand, when resolving a transition dependency according to Definition 7, an agent i can become aware of an incoming job. Thus, i may decide to wait for that arrival, instead of starting to execute another job. Hence, also schedules can be created that are not non-delay. On the other hand, very poor policies with unnecessary idle times can be avoided, since a decision to stay idle will be taken very dedicatedly, viz only when a future job arrival has been announced. This falls into place with the fact that the extension of an agent's local state to $\hat{s_i} = (s_i, z_i)$ is rather limited and consequently the number of local states is only slightly increased.

5.3 Illustrative Benchmark

We start off with the FT6 benchmark problem taken from [1]. This depicts a problem with 6 resources and 6 jobs consisting of 6 operations each, hence we consider a DEC-MDP with 6 independently learning agents. Figure 1 summarizes the learning curves for the three approaches we want to compare (note that the SPT/FIFO/AMCC rules yield $C_{max} = 88/77/55$, here, and are not drawn for clarity). results are averaged over 10 experiment repetitions and indicators for best/worst runs are provided.

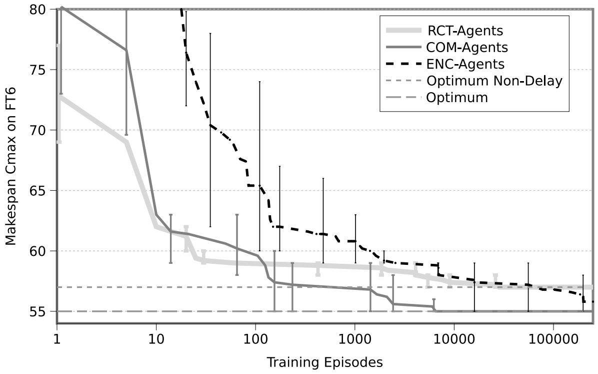

Figure 1: Learning Curves for the FT6 Benchmark

$m \times n$	#Prbl	SPT	AMCC	Opt.	RCT	COM	ENC	Err
$5x10$	3	734.7	702.7	614.0	648.7	642.0	648.3	4.6
$10x10_a$	3	1174.3	1096.0	1035.7	1078.0	1062.7	1109.0	2.6
$10x10_b$	5	1000.2	894.2	864.2	899.0	894.6	928.6	3.5
$10x10_c$	9	1142.6	977.1	898.2	962.7	951.0	988.4	5.9
$5x20$	1	1267.0	1338.0	1165.0	1235.0	1183.0	1244.0	1.5
$15x20$	3	888.3	771.0	676.0	747.7	733.7	818.0	8.6

Table 1: Learning results for scheduling benchmarks of varying size. All entries are average makespan values. The last column shows the relative remaining error (%) of the COM-agents compared to the theoretical optimum. Indices a, b, and c stand for problem sets provided by different authors.

First of all, this experiment shows the effectiveness of our approach, since each type of learning agents considered manages to attain its respective optimum and because static dispatching rules with a local view are clearly outperformed. The FT6 benchmark is a problem, where the best reactive policy (hence, the best non-delay schedule with $C_{max} = 57$) is dragging behind, since the optimal solution corresponds to a delay schedule with makespan of 55. The steepest learning curve emerges for purely reactive agents that achieve the best non-delay solution, hence little interaction with the process is required for those agents to obtain high-quality policies. By contrast, ENC- and COM-agents are capable of learning the optimal policy, where the former require significantly more training time than the latter (note the log scale in Figure 1). This can be tributed to the clearly increased local states of ENC-agents and to the fact that they may take idle actions in principle in any state, while COM-agents do so only when a notification regarding forthcoming externally influenced state transitions has been received.

5.4 Benchmark Results

We also applied our framework to a large variety of different-sized benchmarks from [1] involving up to 15 agents and 20 jobs. In 12 out of the 37 benchmarks examined already the RCT version of our learning agents succeeded in acquiring the optimal joint policy. This also means that in those scenarios (all of them involved 5 resources) the optimal schedule is a non-delay one and we omit experiments using ENC- or COM-agent as no further improvement is possible.

Table 1 provides an overview of the results for the remaining, more intricate 25 benchmark problems (expect for FT6, cf. Section5.3), grouped by problem sizes ($m \times n$). This summary gives the quality of policies obtained after 25000 training episodes. Since ENC-agents have shown to require substantially longer to acquire high-quality policies, the results in the corresponding column are expectedly poor. However, while purely reactive agents already outperform standard rules, their enhancement by means of dedicated communication yields excellent improvements in all cases.

6. RELATED WORK

One of the first formal approaches to model cooperative multi-agent systems was the MMDP framework by Boutilier [5], which requires every agent to be aware of the current global state. By contrast, factored state information including local partial/full observability are key ingredients of

[2] Local state space sizes are increased from 2^n to 3^n per agent; the number of local policies is always $n!$, many of which are very bad in terms of makespan.

the DEC-POMDP framework of Bernstein et al. [4]. While the general problem has NEXP-complete complexity, other researchers have subsequently identified specific subclasses with lower computational complexity: In [3], transition independent DEC-MDPs are introduced in which no agent affects transitions of other agents. And in [9], DEC-MDPs with a synchronizing communication protocol, that allows for exchanging full local states with other agents from time to time, are considered. While these subclasses are quite distinct, our class of factored m-agent DEC-MDPs with changing action sets and partially ordered transition dependencies features some commonalities with DEC-MDPs with event-driven interactions [2] where the latter focus on systems with two agents only and assume less structure in the inter-agent dependencies. The formal relation between the complexity of a subclass of general DEC-MDPs and that class' properties has been established by Shen et al. [13].

Independently learning agents have been targeted in a number of recent publications (e.g. [17, 6]). Communication as a means of conveying information that is local to one agent to others has been investigated, e.g., in [9]. Here, policy computation is facilitated by allowing agents to fully synchronize their local histories of observations. By contrast, we have explored a very limited form of directed communication that informs other agents about forthcoming interferences on state transition. Other approaches with limited communication can be found in [14] where each agent broadcasts its expected gain of a learning update and coordination is realized by performing collective learning updates only when the sum of the gains for the team as a whole is positive, or in [15] where communication is employed to enable a coordinated multi-agent exploration mechanism.

7. CONCLUSION

In the first part of this paper, we have identified a class of cooperative decentralized MDPs that features a number of regularities in the way agents influence the state transitions of other agents. Exploiting the knowledge about these correlations, we have proven that this class of problems is easier to solve (NP-hard) than general DEC-MDPs (NEXP-hard). In the second part, we have focused on more practical issues relevant to distributed control. On the one hand, we have looked at possibilities for modeling memoryless agents and enhancing them by restricted allowance of communication. On the other hand, we have outlined a coordinated batch-mode reinforcement learning algorithm that facilitates the agents to concurrently and independently learn their local policies of action online.

The subclass of DEC-MDPs specified covers a wide range of practical problems. In support of this, we applied our learning approach to production planning problems and evaluated it using scheduling benchmarks that are already NP-hard when solved in a centralized manner. The results obtained are convincing as benchmark problems of current standards of difficulty can be approximately solved by the learning method we suggest. The policies our agents acquire clearly surpass traditional dispatching rules and, in some cases, are able to solve the problem instances optimally.

8. ACKNOWLEDGEMENTS

This research has been supported by the German Research Foundation (DFG) under grant number Ri-923/2-3.

9. REFERENCES

[1] J. Beasley. OR-Library, 2005, http://people.brunel.ac.uk/∼mastjjb/jeb/info.html.

[2] R. Becker, S. Zilberstein, and V. Lesser. Decentralized Markov Decision Processes with Event-Driven Interactions. In *Proceedings of AAMAS 2004*, pages 302–309, New York, USA, 2004. ACM Press.

[3] R. Becker, S. Zilberstein, V. Lesser, and C. Goldman. Solving Transition Independent Decentralized MDPs. *Journal of AI Research*, 22:423–455, 2004.

[4] D. Bernstein, D. Givan, N. Immerman, and S. Zilberstein. The Complexity of Decentralized Control of Markov Decision Processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

[5] C. Boutilier. Sequential Optimality and Coordination in Multiagent Systems. In *Proceedings of IJCAI-99*, pages 478–485, Stockholm, 1999. Morgan Kaufmann.

[6] O. Buffet, A. Dutech, and F. Charpillet. Shaping Multi-Agent Systems with Gradient Reinforcement Learning. *Autonomous Agent and Multi-Agent System Journal*, 15(2):197–220, 2007.

[7] D. Ernst, P. Geurts, and L. Wehenkel. Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research*, (6):504–556, 2005.

[8] T. Gabel and M. Riedmiller. On a Successful Application of Multi-Agent RL to Operations Research Benchmarks. In *Proceedings of ADPRL 2007*, pages 68–75, Honolulu, USA, 2007. IEEE Press.

[9] C. Goldman and S. Zilberstein. Optimizing Information Exchange in Cooperative Multi-Agent Systems. In *Proceedings of AAMAS 2003*, pages 137–144, Melbourne, Australia, 2003. ACM Press.

[10] C. Goldman and S. Zilberstein. Decentralized Control of Cooperative Systems: Categorization and Complexity Analysis. *Journal of Artificial Intelligence Research*, 22:143–174, 2004.

[11] M. Lauer and M. Riedmiller. An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems. In *Proceedings of ICML 2000*, pages 535–542, Stanford, USA, 2000. AAAI Press.

[12] M. Pinedo. *Scheduling. Theory, Algorithms, and Systems*. Prentice Hall, 2002.

[13] J. Shen, R. Becker, and V. Lesser. Agent Interaction in Distributed POMDPs and Implications on Complexity. In *Proceedings of AAMAS 2006*, pages 529–536, Hakodate, Japan, 2006. ACM Press.

[14] D. Szer and F. Charpillet. Coordination through Mutual Notification in Cooperative Multiagent RL. In *Proceedings of AAMAS 2004*, pages 1254–1255, New York, USA, 2004. IEEE Computer Society.

[15] K. Verbeeck, A. Nowe, and K. Tuyls. Coordinated Exploration in Multi-Agent RL: An Application to Load-Balancing. In *Proceedings of AAMAS 2005*, pages 1105–1106, Utrecht, 2005. ACM Press.

[16] C. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8:279–292, 1992.

[17] P. Xuan, V. Lesser, and S. Zilberstein. Communication Decisions in Multi-Agent Cooperation: Model and Experiments. In *Proceedings of AGENTS01*, pages 616–623, Montreal, Canada, 2001. ACM Press.