Joint Equilibrium Policy Search for Multi-Agent Scheduling Problems

Thomas Gabel and Martin Riedmiller

Neuroinformatics Group
Department of Mathematics and Computer Science
Institute of Cognitive Science
University of Osnabrück, 49069 Osnabrück, Germany
{thomas.gabel|martin.riedmiller}@uni-osnabrueck.de

Abstract. We propose joint equilibrium policy search as a multi-agent learning algorithm for decentralized Markov decision processes with changing action sets. In its basic form, it relies on stochastic agent-specific policies parameterized by probability distributions defined for every state as well as on a heuristic that tells whether a joint equilibrium could be obtained. We also suggest an extended version where each agent employs a global policy parameterization which renders the approach applicable to larger-scale problems. Joint-equilibrium policy search is well suited for production planning, traffic control, and other application problems. In support of this, we apply our algorithms to a number of challenging scheduling benchmark problems, finding that solutions of very high quality can be obtained.

1 Introduction

Establishing inter-agent coordination in multi-agent systems depicts a challenging task. Agents that are disallowed to exchange coordinative messages must both determine where equilibria are located in the joint state-action space and also find out which equilibria are strived for by other agents. In this paper, we consider teams of cooperative agents that all seek to optimize a global reward. We assume that there exists at least one sequence of joint actions that leads the collective to a joint equilibrium, i.e. to a final state reaching which means to collect maximal summed rewards for all agents. Our goal is to enable the agents to learn to reach a joint equilibrium with increasing frequency by allowing them to adjust their probabilities of executing actions appropriately.

On the one hand, we build upon the framework of decentralized Markov decision processes with changing action sets that we recently [7] proposed as a mean to model a subclass of general multi-agent problems that features provably lower complexity than solving general DEC-MDPs does. A key property of this class is that each action can be executed only once by each agent. On the other hand, we borrow from an equilibrium selection algorithm for single-stage games by Fulda [5] and extend it (a) towards scenarios with multiple states at which actions can be executed and (b) towards a compact and efficient representation of the agents' policies. In so doing, we obtain substantial savings in terms of computation time and memory requirements.

Although our learning approach is applicable to various practical problems, such as network routing or railway traffic control, the paper at hand specifically targets an application from the realm of production planning: We focus on job-shop scheduling problems (JSSP) that can easily be posed as multi-agent problems and represent and interesting testbed for distributed learning algorithms. Problems of this type are well-known for their intricacy (NP-hardness) and, what makes them appealing for use as a testbed, there exist various established benchmark problem suites using which we can compare our algorithms to other approaches.

The remainder of this paper is structured as follows. In the next section, we describe the general problem setting, clarify necessary notation, and introduce joint equilibrium policy search as a distributed learning algorithm for independent agents with common interests. In Section 3 we propose and theoretically investigate a substantial extension of that algorithm towards the use of a global, instead of local (state-specific) policy parameterization which renders the approach better scalable to larger problems. Section 4 presents the application domain of job-shop scheduling, explains how scheduling problems can be cast as multi-agent learning problems, and evaluates empirically the usability of the algorithms proposed in this paper for several benchmark problems.

2 Joint Equilibrium Policy Search

Joint equilibrium policy search (JEPS) is a distributed purely policy-based algorithm. Before presenting its details, we start off by providing some necessary notation.

2.1 Basics

We embed the problem settings of our interest into the framework of decentralized Markov decision processes (DEC-MDP) [2].

Definition 1 (Factored m-Agent DEC-MDP). *A factored m-agent DEC-MDP M is defined by a tuple*

$$\langle Ag, S, A, P, R, \Omega, O \rangle$$

where $Ag = \{1, \dots, m\}$ is a set of agents, S is the set of world states that can be factored into m components $S = S_1 \times \dots \times S_m$ (S_i belong to one of the agents each, all local states are fully observable), $A = A_1 \times ... \times A_m$ is 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 is the transition function with $P(s'|s, a)$ denoting the probability that the system arrives at state s' upon executing a in s, R is a reward function with $R(s, a, s')$ denoting the reward for executing a in s and transitioning to s'. Moreover, M is jointly fully observable, i.e. the current state is entirely determined by the amalgamation of all agents' state observations.

We refer to the agent-specific components $s_i \in S_i$ and $a_i \in A_i$ as the local state and action of agent i. Each agent has only its local view, i.e. gets no information about other agents' local states and actions. We assume that there are some regularities that determine the way local actions exert influence on other agents' states. First, we assume that the sets of local actions A_i change over time.

Definition 2 (DEC-MDP with Changing Action Sets). *A factored m-agent DEC-MDP 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 i ($s_i = A_i$) and A_i is a subset of the set of all available local actions $\mathcal{A}_i = \{\alpha_{i1} \dots \alpha_{ik}\}$, thus $S_i = \mathcal{P}(\mathcal{A}_i)$.*

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. 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 (see [7] for a formalization). Also, there are no circular state transition influences which implies that each agent can execute each of its local actions only once.

The influence exerted on another agent always yields an extension of that agent's action set: If the execution of local action α by agent i influences the local state of agent j, and 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 $A_i(s_i)$. Thus, the multi-agent system is guaranteed to reach a final state $s^f \in S$ at which all actions have been processed and it holds $s_i^f = \emptyset$ for all i.

2.2 Learning Joint Policies

JEPS is a purely policy-search based algorithm (i.e. no value functions are employed), where all agents' policies are stochastic and are dependent on state-specific probability vectors denoting the probabilities with which each action is executed.

Definition 3 (JEPS Policy with Local Parameterization). *Let $s_i \in \mathcal{P}(\mathcal{A}_i)$ be the current state of agent i, where $s_i = A_i(s_i) = \{\alpha_1, \dots, \alpha_{|s_i|}\}$ corresponds to the set of actions agent i can currently execute. Let $P_L(s_i) = \{p(\alpha_1|s_i), \dots, p(\alpha_{|s_i|}|s_i)\}$ be a probability distribution over all actions from s_i, thus $0 \leq p(\alpha_j|s_i) \leq 1$ and $\sum_j p(\alpha_j|s_i) = 1$. Then, for agent i's policy of action $\pi_i : S_i \rightarrow \mathcal{A}_i$ it holds $s_i \mapsto \alpha$ where α is selected from s_i with probability $p(\alpha|s_i)$. Accordingly, the joint policy is defined as $\pi = \langle \pi_1, \dots, \pi_m \rangle$.*

We assume that all action probability vectors are randomly initialized and that the set of agents repeatedly interacts with the DEC-MDP until the final state s^f has been reached (also called the processing of a single episode). Then, the global reward r is distributed to all agents and the system is reset to a starting state. JEPS borrows from [5] in that it employs a binary heuristic $H(r)$ that is capable of telling whether a joint equilibrium has been attained. If so, it returns true, otherwise false. In the remainder of this paper, we utilize a rather simplistic implementation of H that returns true only if the current global reward equals or exceeds the maximal reward r_{max} obtained so far, i.e. $H(r) = 1 \Leftrightarrow r \geq r_{max}$. This idea has been exploited in a different context already by the Rmax algorithm [3] and by optimistic assumption Q learning [8].

After having finished a single episode and only if having found that $H(r) = 1$, each agent starts updating its action probabilities for all states it has encountered during that episode. Here, the probabilities of all actions that were executed (and thus contributed to reaching the joint equilibrium) are increased, while the probabilities for executing any of the actions despised is decreased (see Algorithm 1). Note, that this update scheme preserves that $\sum_j p(\alpha_j|s_i) = 1$ for all s_i. While the updates JEPS does to the action probabilities are calculated in a similar manner as in [5], the crucial difference is that JEPS is capable of distinguishing between multiple states s_i, and can thus handle more than single-stage games as it stores a single action probability vector for each local state.

```
Input: learning rate $\gamma \in (0,1]$, state-action history of current episode
       $h = [s_i(0), a_i(0), s_i(1), \ldots, s_i(T-1), a_i(T-1), s^f]$
       where $T = |\mathcal{A}_i|$ denotes the episode's length, global reward $r \in \mathbb{R}$
1: if $H(r) = 1$ then
2:   for $t = 0$ to $t < T$ do
3:     for all $\alpha \in s_i(t)$ do
4:       if $\alpha = a_i(t)$ then $p(\alpha|s_i(t)) \leftarrow p(\alpha|s_i(t)) + \gamma(1 - p(\alpha|s_i(t)))$
5:       else $p(\alpha|s_i(t)) \leftarrow (1-\gamma)p(\alpha|s_i(t))$
```

Algorithm 1: JEPS Policy Updates by Agent i Using Local Action Parameters

2.3 Discussion

JEPS extends the mentioned learning approach for single-stage games in a purposive manner to problems with multiple states. Consequently, the policy update mechanism is guaranteed to converge[1] to a joint equilibrium as long as the heuristic H is correct in the sense that it tells a true joint equilibrium. This follows immediately from the convergence proof for single-stage games, since each of JEPS' states together with its belonging action probability vector can be regarded as an individual single-stage game considered by Fulda [5].

When intending to apply the version of JEPS presented to practical problems, two considerable problems arise. First, with a growing number of actions $|\mathcal{A}_i|$ available to the agents, the size of the state space grows exponentially, since

[1] Here, convergence means that for all states s_i there is an $\alpha \in s_i$ such that $p(\alpha|s_i) \to 1$ in the course of learning.

states correspond to sets of available actions and, hence, in the worst case it holds $|S_i| = |\mathcal{P}(\mathcal{A}_i)| = 2^{|\mathcal{A}_i|}$. Accordingly, storing action probability vectors for all states (separately for each of the agents) quickly becomes intractable as the problem size grows. Additionally, the large number of action probability vectors also increases the learning time needed until convergence to a nearly deterministic policy is achieved.

To tackle these problems, in the next section, we suggest a compact policy representation in combination with an alternative policy update mechanism that clearly reduces the computational complexity and memory requirements while still allowing for convergence to a joint equilibrium.

3 JEPS with Global Action Parameterization

Knowing the properties of DEC-MDPs with changing action sets (Definition 2) and given the problems mentioned in Section 2.3, a crucial observation is that each agent actually just has to be capable of learning a *total order* in which it executes all actions from $\mathcal{A}_i$.

3.1 Learning Total Orders of Action Execution

The basic idea for a version of JEPS that employs global action parameters (JEPS$_G$) is that, for each of the agents, we attach a single, or global, parameter to each action in $\mathcal{A}_i$ from which then its probability of execution is induced.

Definition 4 (JEPS with Global Action Parameterization). *Let $P_G = \{p_G(\alpha_k)|\alpha_k \in \mathcal{A}_i\}$ be a probability distribution over the set $\mathcal{A}_i$ of local actions agent i can execute, and let $s_i = A_i(s_i) = \{\alpha_1, \dots, \alpha_{|s_i|}\} \in \mathcal{P}(\mathcal{A}_i)$ be its current state. Then, for agent i's policy of action $\pi_i : S_i \to \mathcal{A}_i$ it holds $s_i \mapsto \alpha$ where α is selected with probability*

$$p(\alpha|s_i) = \begin{cases} \frac{p_G(\alpha)}{\sum_{\alpha_k \in s_i} p_G(\alpha_k)} & \textit{if } \alpha \in s_i \\ 0 & \textit{else} \end{cases}, \tag{1}$$

and the joint policy π is the concatenation of local policies $\langle \pi_1, \dots, \pi_m \rangle$.

Using this kind of policy representation each agent must store only $|\mathcal{A}_i|$ parameters which represents an enormous saving in terms of memory requirements compared to the JEPS version with local action probabilities.

Based on the policy representation with global parameters according to Definition 4, we suggest a learning algorithm that, for each agent, performs the parameter updates directly on the global parameter vector P_G. The distinguishing property of Algorithm 2 is that all positive updates, i.e. updates for actions taken when having reached a joint equilibrium (line 4), are performed relative to a state-specific baseline $\kappa_{s_i(t)}$ that is defined as

$$\kappa_{s_i(t)} := \sum_{\alpha_k \in s_i(t)} p_G(\alpha_k). \tag{2}$$

By this, it is possible to relate the local situation of agent i, i.e. its current local state, to the set of global action parameters, and it also ensured that P_G stays a proper probability distribution with $\sum_{\alpha_k \in s_i(t)} p_G(\alpha_k) = 1$.

```
Input: learning rate $\gamma \in (0,1]$, state-action history of current episode
       $h = [s_i(0), a_i(0), s_i(1), \ldots, s_i(T-1), a_i(T-1), s^f]$
       where $T = |\mathcal{A}_i|$ denotes the episode's horizon, global reward $r \in \mathbb{R}$
1: if $H(r) = 1$ then
2:   for $t = 0$ to $t < T$ do
3:     forall $\alpha \in s_i(t)$ do
4:       if $\alpha = a_i(t)$ then $p_G(\alpha) \leftarrow p_G(\alpha) + \gamma(\sum_{\alpha_k \in s_i(t)} p_G(\alpha_k) - p_G(\alpha))$
5:                  else $p_G(\alpha) \leftarrow (1-\gamma) p_G(\alpha)$
```

Algorithm 2:
Policy Updates by JEPS Agent i Using Global Action Parameters

For this algorithm, we can show that for every agent and each local state s_i the probability of executing an action $\alpha \in s_i$ that does *not* support yielding a joint equilibrium is declining if it exceeds some threshold.

Lemma 1. *Let $\alpha \in s_i$ and $p_G(\alpha) > \frac{\kappa_{s_i}}{2}$. If the execution of α in state s_i does not yield a joint equilibrium, then $\Delta p_G(\alpha) < 0$, where Δp_G represents the difference of $p_G(\alpha)$ after and prior to the call to Algorithm 2.*

Proof. If the current episode did not reach an equilibrium, no updates are performed. Consider the case when an equilibrium has been reached and focus on the smallest value of t for which it holds $\alpha \in s_i(t)$ for an arbitrary $\alpha \in \mathcal{A}_i$. Let $t+v$ ($v \geq 1$) be the stage at which α has finally been selected for execution. Then, the value of $p_G(\alpha)$ will have been decremented v times according to line 5 (denote the result of this calculation as $p_G^-(\alpha)$) and been increased a single time at $s_i(t+v)$. Thus,

$$p'_G(\alpha) := p_G(\alpha) + \Delta p_G(\alpha) = p_G^-(\alpha) + \gamma(\kappa_{s_i(t+v)} - p_G^-(\alpha))$$
$$= (1-\gamma)^{v+1} p_G(\alpha) + \gamma \sum_{\alpha_k \in s_i(t+v)} p_G^-(\alpha_k).$$

For the sum on the right-hand side there exist values $v_k \geq 0$ for all $\alpha_k \in s_i(t+v)$ such that $p_G^-(\alpha_k) = (1-\gamma)^{v_k} p_G(\alpha_k)$. Since we are looking for the circumstances under which $p'_G(\alpha) < p_G(\alpha)$, i.e. $\Delta p_G(\alpha) < 0$, we finally arrive at

$$\Delta p_G(\alpha) < 0 \Leftrightarrow p_G(\alpha) > \frac{\gamma \sum_{\alpha_k \in s_i(t+v)} (1-\gamma)^{v_k} p_G(\alpha_k)}{1 - (1-\gamma)^{v+1}} =: \delta(\gamma).$$

The term $\delta(\gamma)$ attains its maximal value for $v = 1$ and $v_k = 0 \forall k$. Then, $\delta(\gamma) = \frac{1-\gamma}{2-\gamma} \sum_{\alpha_k \in s_i(t+v)} p_G(\alpha_k)$. Maximizing subject to γ ($\gamma \to 0$), we obtain $\delta = \frac{\kappa_{s_i(t+v)}}{2}$. And because by definition $\kappa_{s_i(t)} > \kappa_{s_i(t+v)}$ for all $v \geq 1$, we finally see that for $p_G(\alpha) > \frac{\kappa_{s_i(t)}}{2}$ it holds $\Delta p_G(\alpha) < 0$. $\square$

Lemma 1 shows that probability updates cannot enforce convergence to suboptimal action choices. Unfortunately, still there may be the case of two joint equilibria with identical global reward between which the agent may oscillate. However, we can show that for any state s_i there is a critical action probability value such that upon exceeding that value one joint equilibrium starts dominating another one.

Lemma 2. *If $\alpha \in s_i(t)$ is an action within a joint equilibrium episode, then there exists a value $p^\star$ such that, if $p_G(\alpha) > p^\star$, then $p_G(\alpha)$ is more likely to increase over time than to decrease.*

Proof. The critical case of $p_G(\alpha)$ decreasing can occur, if there is a $\beta \in s_i(t)$ such that still a joint equilibrium can be obtained when β is executed in s_i. If α is executed, then $p_G(\alpha)$ is increased (line 4), whereas $p_G(\beta)$ is decreased (line 5) at least one time and later increased at a $t+v > t$ when β is finally executed. If β is selected in s_i, the situation is the other way round ($p_G(\alpha)$ decreased v times according to line 5, if it is selected v decision time points later). Consequently, with a probability of $\frac{p_G(\alpha)}{\kappa_{s_i(t)}}$ it holds

$$p_G^\alpha(\alpha) := p_G(\alpha) + \Delta p_G(\alpha) = p_G(\alpha) + \gamma(\kappa_{s_i(t)} - p_G(\alpha))$$

and with a probability of $\frac{p_G(\beta)}{\kappa_{s_(t)}}$ it holds

$$\begin{aligned} p_G^\beta(\alpha) &:= p_G^-(\alpha) + \gamma(\kappa_{s_i(t+v)} - p_G^-(\alpha)) \\ &= (1-\gamma)^v p_G(\alpha) + \gamma(\kappa_{s_i(t+v)} - (1-\gamma)^v p_G(\alpha)). \end{aligned}$$

Since we look for the conditions under which $\Delta p_G(\alpha) = p'_G(\alpha) - p_G(\alpha) > 0$, we can express this inequation using a weighted average as

$$\frac{p_G(\alpha)p_G^\alpha(\alpha) + p_G(\beta)p_G^\beta(\alpha)}{\kappa_{s_i(t)}(p_G(\alpha)+p_G(\alpha))} - p_G(\alpha) > 0.$$

After a number of algebraic reformulations, this simplifies to

$$\frac{\kappa_{s_i(t)}}{p_G(\beta)} + \frac{\kappa_{s_i(t+v)}}{p_G(\alpha)} > \frac{1+\gamma-(1-\gamma^{v+1})}{\gamma}.$$

The right-hand side of this inequation attains its maximum for $v \to \infty$ which becomes $1 + \frac{1}{\gamma}$. For the left-hand side, we know that $\kappa_{s_i(t)} \geq p_G(\alpha) + p_G(\beta)$ and $\kappa_{s_i(t+v)} \geq p_G(\alpha)$. Assuming the worst case (both equalities) here, too, we arrive at

$$\frac{p_G(\alpha)+p_G\beta}{p_G(\beta)} + \frac{p_G(\alpha)}{p_G(\alpha)} > 1 + \frac{1}{\gamma} \text{ and thus } \frac{p_G(\alpha)}{p_G(\beta)} > \frac{1-\gamma}{\gamma}.$$

Consequently, if for a state s_i one joint equilibrium action $\alpha \in s_i$ dominates all other actions by a share of at least $p^\star := \frac{1-\gamma}{\gamma}$, then $\Delta p_G(\alpha)$ tends to be positive. □

3.2 Discussion

If for some action α within an equilibrium episode the probability of execution exceeds some critical value, then $p_G(\alpha)$ tends to be increasing continually. Since updates are not just made for single actions, but for all actions taken during an equilibrial episode, this argument transfers to the remaining actions from $\mathcal{A}_i$ as well. With continued positive updates all $p_G(\alpha_k)$ converge such that for each s_i

there is a $\alpha^{\star}_{s_i}$ with $\frac{p_G(\alpha^{\star}_{s_i})}{\kappa_{s_i}} \to 1$, which means that the policy the agent pursues approaches a deterministic one.

Of course, the time required for convergence to occur may be high. Setting the learning rate γ to a higher value, learning can be speeded up. However, this comes at the cost of an increased probability, that learning converges prematurely to a non-equilibrium, because the heuristic H we use is imperfect with respect to the true joint equilibrium of the system. Insofar, adjusting γ represents a mean to trade off learning speed and the goal of obtaining a joint policy very close to a joint equilibrium.

Returning to the point of view of a total order of action execution that is represented by the vector of global action parameters P_G, we observe that JEPS$_G$ may drive the parameters $p_G(\alpha)$ and $p_G(\beta)$ for some actions α and β (in particular for actions whose execution is repeatedly postponed) to very small numerical values – while at the same time it may be required that the share of $p_G(\alpha)$ and $p_G(\beta)$ must be either very large or small. As a consequence, a limiting factor when implementing and using Algorithm 2 is given by the smallest real-valued number that can be represented on the respective hardware[2]. Accordingly, the convergence behavior of a practical implementation of JEPS$_G$ will be as follows:

a) Convergence to a joint equilibrium policy, as indicated by heuristic H in conjunction with r_{max}, occurring with a probability of nearly one may occur. This means, after λ learning episodes it holds for all agents i, for all states s_i, and for all $\alpha \in s_i$ that $\frac{p_G(\alpha)}{\kappa_{s_i}} > 1 - \epsilon$ for some small $\epsilon > 0$.
b) Numerical underflow problems arise[3], i.e. that there is an agent i and a state s_i where for a $\alpha \in s_i$ it holds $p_G(\alpha) < \epsilon_{min}$, where $\epsilon_{min} \in \mathbb{R}^+$ is the smallest floating number representable on the respective hardware platform.
c) The learning time allotted to the algorithm is exceeded, i.e. λ_{max} learning episodes have been processed without situation a) and b) having occurred.

Note that, although no convergence is achieved in cases b) and c), the algorithm does not diverge – in fact, it rather stops its learning process too early. At least, in these cases we can use the value of the presumed joint equilibrium found so far (r_{max}) as an indicator of the true equilibrium that eventually would have been discovered if λ_{max} was larger or ϵ_{min} smaller.

4 Empirical Evaluation

In this section, we use the class of DEC-MDPs with changing action sets to model job-shop scheduling problems (JSSP), and we evaluate the performance of JEPS and JEPS$_G$ in this context using various established scheduling benchmarks.

[2] According to the IEEE standard for binary floating-point arithmetic (IEEE 754), when using 64 bit, the smallest number is approximately $2.2 \cdot 10^{-308}$ (*double* type).

[3] This case is more likely to occur, the larger $|P_G|$ is.

4.1 Application Domain: Job-Shop Scheduling

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 given order. Each job j consists of ν_j operations $o_{j,1} \ldots o_{j,\nu_j}$ that have to be handled on a certain resource for a specific duration. 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 (and hence reaching the final state s^f) as quickly as possible, since most publications on results for job-shop scheduling benchmarks focus on that objective, too.

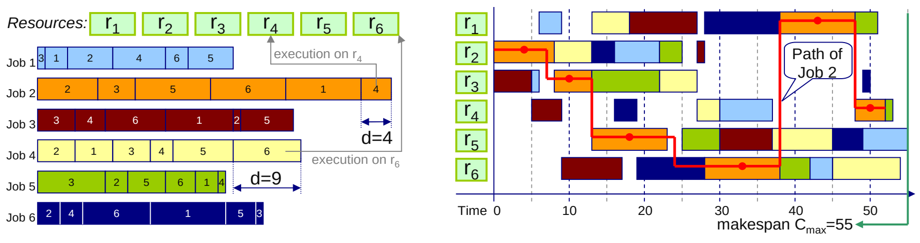

Fig. 1. Example Job-Shop Scheduling Problem FT6 (left) and Optimal Solution (right)

Solving JSSPs 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. We revert to a collection of sample problems that is provided by the OR Library [1]. A common characteristic of those scheduling benchmarks is that usually no recirculation of jobs is allowed, i.e. that each job must be processed exactly once on each resource ($\nu_j = m$). Figure 1 shows an example of a small job-shop scheduling problem with six resource and six jobs consisting of six jobs each; also an optimal solution of that problem with respect to minimal makespan is illustrated using a Gantt chart. For more details on scheduling, the reader is referred to [10, 4].

We model JSSPs as factored m-agent DEC-MDPs with changing action sets as follows. We attach to each of the resources one agent i whose local action is to decide which waiting job to process next. Agent i's local state of i can be fully described by the changing set of jobs currently waiting for further processing. Choosing and executing a job represents a local action ($\mathcal{A}_i$ is the set of jobs that must be processed on resource i), which is why it holds $S_i = \mathcal{P}(\mathcal{A}_i)$. After finishing the processing of a job's operation, this job is transferred to another resource, where the order of resources on which a job's operations must be processed is given a priori. In conjunction with the no recirculation property mentioned above, in fact, each job (one of its operations, respectively) has to be executed on each resource exactly once. As a consequence, for JEPS$_G$ is will be sufficient that each agent stores one action probability parameter for each job.

4.2 Benchmark Results

Given an instance of a JSSP, all agents process waiting jobs in a reactive manner, i.e. they select jobs with respect to the probability determined by their current policy parameters, and never remain idle, if there is at least one job waiting. When all jobs are finished and, hence, s^f has been reached, the global reward $r = -C_{max}$ is conveyed to the agents, the policy update algorithm (Algorithm 1/2) is called, and finally the system is reinitialized to the starting state where no jobs have been processed. We allow the agents to maximally process $\lambda_{max} = 250k$ episodes, however, in most cases convergence is achieved much faster. For consistency, during all experiments we set $\gamma = 0.1$, a value that ad hoc brought about good results and whose optimization should be subject to further studies.

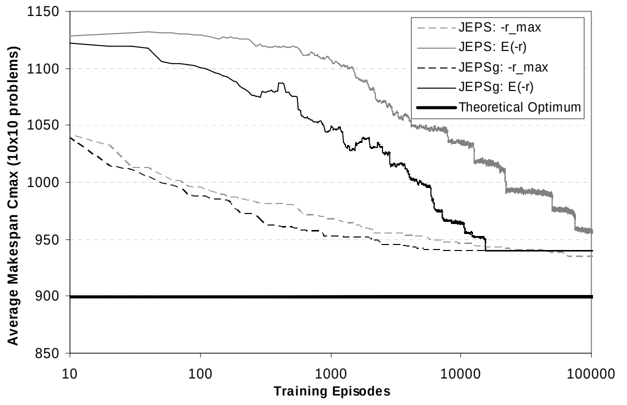

Fig. 2. Learning Progress for JEPS and JEPS$_G$

Figure 2 illustrates the learning progress averaged over 15 JSSP problems involving 10 jobs and 10 machines using JEPS as well as JEPS$_G$. The solid curves show the average expected performance (in terms of makespan C_{max}, i.e. negative reward) of the stochastic joint policies subject to the number of training episodes. Dashed curves indicate the development of the value of the supposed joint equilibrium $-r_{max}$, as utilized by the heuristic H.

Apparently the $-r_{max}$ and $E[-r]$ curves approach each other much faster for the JEPS$_G$ variant of the algorithm than for JEPS with local policy parameterization. For the 15 scenarios considered, JEPS$_G$ converges at the latest after about 11k episodes (note the log scale x-axis). By contrast, JEPS needs much longer to yield convergence, but achieves finding slightly superior values of r_{max}, i.e. on average the learnt joint policy comes closer to the true joint equilibrium (indicated by the average theoretical optimum for the scenarios considered).

Size		Theor.	**JEPS**				Pol.	**JEPS**$_G$						Pol.
$m \times n$	#Prbl	Optim.	#a	$-r_{max}$	$E[-r]$	Err.	Size	#a	#b	#c	$-r_{max}$	$E[-r]$	Err.	Size
5×10	5	620.2	5	631.8	631.8	1.9%	1029k	4	0	1	635.4	644.2	2.5%	0.6k
5×15	5	917.6	5	917.6	917.6	0.0%	18M	5	0	0	917.6	917.6	0.0%	1.1k
5×20	6	1179.2	0	-	-	-	∞	5	1	0	1188.3	1196.5	0.8%	1.5k
$10 \times 10_a$	3	1035.7	3	1071.0	1071.0	3.4%	3.5M	3	1	0	1076.7	1076.7	3.9%	1.2k
$10 \times 10_b$	5	864.2	5	902.4	902.4	4.4%	973k	5	1	0	894.2	894.2	3.5%	1.1k
$10 \times 10_c$	9	898.2	8	935.3	937.9	4.1%	6.4M	8	1	0	952.4	953.6	6.0%	1.2k
10×15	5	983.4	0	-	-	-	∞	2	1	2	1032.4	1142.4	5.0%	2.1k
15×15	5	1263.2	0	-	-	-	∞	3	1	1	1341.2	1375.8	6.1%	3.0k
15×20	3	676.0	0	-	-	-	∞	0	0	3	732.0	819.7	8.3%	4.1k

Table 1. Learning results for scheduling benchmarks of varying size, opposed for *JEPS* and *JEPS*$_g$. All entries are averaged over *#Prbl.* *#a*, *#b*, and *#c* correspond to the convergence possibilities listed in Section 3.2. The last column in each part shows the average size of a policy measured in bytes. *Err.* columns denote the relative remaining error (%) of the makespan ($-r_{max}$) achieved by the joint policy compared to the theoretical optimum and, thus, indicate to what extent reaching the true joint equilibrium was missed. Indices a, b, c stand for problem sets provided by different authors.

The limitation of the basic form of JEPS becomes obvious when having a look at the sizes of the policies that must be kept in memory by the agents (see the rightmost columns in the of the JEPS and JEPS$_G$ part in Table 1, measured in bytes per policy). Since the number of policy parameters grows exponentially with n, the application of JEPS for $m \times n$ problems with larger values of n is infeasible due to excessive memory requirements. On the contrary, the average policy sizes of JEPS$_G$ agents are negligible. Here, instead the underflow problem (cf. Section 3.2) may occur for larger values of n. However, using JEPS$_G$, policies of high quality can be learnt even for larger-sized problem instances.

The remaining error values achieved can well compete with alternative approaches that tackle the scheduling problem from a decentralized perspective (centralized algorithms mostly find the optimum). For example, dispatching priority rules are clearly outperformed (best rules are SPT, which chooses operations with shortest processing time, and AMCC [9], which is a heuristic to avoid the maximum current C_{max}, with an average error of 20.6% and 7.8% for the 46 problems mentioned in Table 1). OA-NFQ [6], a value-function based reinforcement learning approach to these problems, reaches an error of 4.2%.

We expect that, in future work, we will be able to further boost the performance of JEPS. In the version presented the reactive functioning of JEPS can generate schedules of the class $\mathbb{S}_n$ of *non-delay* schedules exclusively: If a resource has finished processing one operation and has at least one job waiting, the respective agent immediately continues processing by picking one of the waiting jobs. JEPS does not allow a resource to remain idle, if there is more work to be done. From scheduling theory, however, it is known that for certain scheduling problem instances the optimal schedule may be a delay schedule from the set of active schedules $\mathbb{S}_a \supsetneq \mathbb{S}_n$, i.e. a schedule where some resource has to remain idle for some time units in order to achieve minimal makespan. As a consequence, JEPS is currently able to produce near-optimal schedules from $\mathbb{S}_n$ and may miss the best schedule possible, though in several cases the true joint equilibrium is indeed found. Yet, an extension of JEPS towards behaving not purely reactively depicts an important and promising issue for future work.

5 Conclusion

We have presented a multi-agent policy search method, JEPS, that is effective in learning joint equilibria, or near-optimal approximations thereof, for decentralized Markov decision processes with changing action sets. Using a variant of the algorithm that employs a highly compacted policy representation, it is possible to apply JEPS to even larger problem instances without impairing performance.

A limiting factor of the approach is the necessity for a heuristic that indicates whether a joint equilibrium has been reached by the ensemble of agents. In future work, we will investigate more sophisticated versions of this heuristic and, moreover, we will explore state of the art mechanisms, such as policy-gradient descent methods, for updating the policy parameters, which we expect to significantly speed up the learning process.

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

References

1. J. Beasley. Or-library, 2005.
   `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`.
2. 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.
3. R. Brafman and M. Tennenholtz. Learning to Cooperate Efficiently: A Model-Based Approach. *Journal of Artificial Intelligence Research*, 19:11–23, 2003.
4. P. Brucker and S. Knust. *Complex Scheduling.* Springer-Verlag, Berlin Heidelberg, 2006.
5. N. Fulda and D. Ventura. Incremental Policy Learning: An Equilibrium Selection Algorithm for Reinforcement Learning Agents with Common Interests. In *Proceedings of the 2004 IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 1121–1125, Budapest, Hungary, 2004. IEEE Press.
6. T. Gabel and M. Riedmiller. Adaptive Reactive Job-Shop Scheduling with Learning Agents. *International Journal of Information Technology and Intelligent Computing*, 2(4), 2008.
7. T. Gabel and M. Riedmiller. Reinforcement Learning for DEC-MDPs with Changing Action Sets and Partially Ordered Dependencies. In *Proceedings of the 7th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008)*, Estoril, Portugal, 2008, to appear.
8. M. Lauer and M. Riedmiller. An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems. In *Proceedings of the International Conference on Machine Learning (ICML 2000)*, pages 535–542, Stanford, USA, 2000. AAAI Press.
9. A. Mascis and D. Pacciarelli. Job-Shop Scheduling with Blocking and No-Wait Constraints. *European Journal of Operational Research*, 143:498–517, 2002.
10. Michael Pinedo. *Scheduling. Theory, Algorithms, and Systems.* Prentice Hall, USA, 2002.