International Journal of Production Research
Vol. 00, No. 00, 00 Month 2009, 1–27

RESEARCH ARTICLE

Distributed Policy Search Reinforcement Learning for Job-Shop Scheduling Tasks

Thomas Gabel* and Martin Riedmiller

Machine Learning Laboratory, Department of Computer Science,
Albert-Ludwigs-Universität Freiburg, Germany
(*Received 00 Month 200x; final version received 00 Month 200x*)

We interpret job-shop scheduling problems as sequential decision problems that are handled by independent learning agents. These agents act completely decoupled from one another and employ probabilistic dispatching policies for which we propose a compact representation using a small set of real-valued parameters. During ongoing learning, the agents adapt these parameters using policy gradient reinforcement learning, with the aim of improving the performance of the joint policy measured in terms of a standard scheduling objective function. Moreover, we suggest a lightweight communication mechanism that enhances the agents' capabilities beyond purely reactive job dispatching. We evaluate the effectiveness of our learning approach using various deterministic as well as stochastic job-shop scheduling benchmark problems, demonstrating that the utilization of policy gradient methods can be effective and beneficial for scheduling problems.

Keywords: Distributed reinforcement learning; Job-shop scheduling; Distributed control; Mulit-agent systems; Policy search

1. Introduction

Scheduling and sequencing have emerged as crucial decision-making tasks to support and enhance the productiveness of manufacturing enterprises as well as logistics and service providers. The general goal of scheduling is to allocate a limited number of resources to outstanding tasks over time such that one or several objectives are optimized. Here, resources and tasks, respectively, depict abstractions of real-world entities that may take

*Corresponding author. Email: tgabel@informatik.uni-freiburg.de; postal address: Machine Learning Lab, Albert-Ludwigs-Universität Freiburg, Georges-Koehler-Allee 079, D-79110 Freiburg, Germany; telephone: +49 761 203 8043; fax: +49 761 203 8007.

ISSN: 1478-6451 print/ISSN 1478-646X online
© 2009 Taylor & Francis
DOI: 10.1080/1478645YYxxxxxxx
http://www.informaworld.com

very different forms depending on the application scenario considered. For example, in warehousing they may correspond to storages and stored goods, in personell management to employees and working shifts, in computer program scheduling to CPU cores and processes, and, most prominently, in manufacturing production control to machines on a working floor and operation steps of a production process.

As we will show, many scheduling problems suggest a natural formulation as distributed decision-making tasks. Hence, the employment of learning multi-agent systems represents an evident approach. Furthermore, given the well-known inherent intricacy of solving scheduling problems, decentralized approaches for solving them may yield a promising option.

Distributed problem solving in practice is often characterized by a large number of involved agents and by a factored system state description where the agents base their decisions on local observations (Kok 2006). Moreover, in many applications it holds true that a local action taken by an agent has an influence on only one other agent. This is, for example, the case for application scenarios from manufacturing, production planning, or assembly line optimization, where typically the production of a good involves a number of processing steps that must be performed in a specific order. It is obvious that the decision to further process a good can only be taken, if all preceding processing steps are finished, a fact that we shall exploit dedicatedly.

The article at hand focuses in depth on one particular type of scheduling problems, for which the argument made in the preceding paragraph holds in every respect, namely *job-shop scheduling*. In particular, we target stochastic job-shop scheduling problems where random events, such as delays in the processing of individual operations, complicate the problem setting and impede the application of centralized standard scheduling algorithms. We emphasize, however, that although the remainder of this article particularly targets job-shop scheduling problems (JSSPs), the methods and algorithms we develop may be employed for different multi-agent problems and, thus, for different kinds of scheduling problems as well. Among those are, for example, single-machine models, flow-shop problems, and even flexible shop problems (Pinedo 2002).

In contrast to the standard approach to solving scheduling problems, we will not deploy a centralized authority for finding a good schedule. Instead, we propose a distributed approach and employ a number of dispatching agents that are entirely independent of one another. These agents start off with no prior knowledge, but they are assumed to learn and improve their dispatching behavior over time, i.e. in the course of repeated interaction with the plant. As a result, these agents, in general, make poor dispatching decisions initially, but are improving their capabilities continually.

Overview

In what follows, we model the class of scheduling problems we are targeting as sequential decision problems with the help of decentralized Markov decision processes (DEC-MDPs, Bernstein *et al.*, 2002). In particular, we utilize the framework of factored DEC-MDPs with changing action sets and partially ordered transition dependencies (Gabel and Riedmiller 2008), which turns out to be very useful for modelling scheduling problems. In so doing, we factorize scheduling problems to handle them in a distributed manner, attaching simple and independent agents to each of the resources. These agents employ probabilistic dispatching policies to decide which operations of the jobs waiting currently at the respective resource should be processed next.

Our core learning approach employs policy gradient (PG) reinforcement learning (Williams 1992, Sutton *et al.* 2000) to optimize the agents' dispatch policies. The basic

idea of policy gradient RL algorithms is to estimate the gradient of the expected return of the process. For scheduling tasks with stochastic dispatching policies, where for example an objective function such as the makespan C_{max} shall be optimized, this translates to an estimate of the gradient of the makespan of the resulting schedule, which is derived with respect to a set of real-valued policy parameters. These parameters make up the agents' stochastic policies and determine their dispatching behavior. To this end, we will suggest a compact representation of the agents' local policies. Following the gradient by adjusting the policy parameters' values (and assuming the correctness of the gradient estimate), it is guaranteed that the expected return of the policy is improved, i.e. in the course of repeated interaction with the plant schedules that are better in terms of makespan are created with higher probability.

Policy gradient methods are in general guaranteed to converge to at least a local optimum with respect to the expected return. Given the reactive dispatching behavior of the agents outlined, however, it is clear that basically only non-delay schedules can be obtained and, hence, the optimal solution, which eventually may feature necessary delay times, cannot be found. For these reasons, we also develop a mechanism that enhances the independently learning agents in such a manner that they become partially aware of inter-agent dependencies, can resolve them, and thus are enabled to also create delay schedules. To evaluate our gradient-descent policy search algorithm for scheduling problems, we make use of various established job-shop scheduling benchmark problems from the OR Library[1] (Beasley 1990). Additionally, we investigate the capabilities of our learning method for stochastic versions of the problem, where the duration of the jobs' operations are randomly perturbed. Furthermore, we compare its performance against a number of alternative reactive solution approaches, i.e. dispatching heuristics, which are also challenged to making their decisions based on local and, thus, partial problem knowledge.

In the next section, we start off by summarizing the key characteristics of the subclass of DEC-MDPs with changing action sets that builds the foundation of our work. Subsequently, in Section 3 we motivate a distributed solution approach and show how job-shop scheduling problems can be modelled within the scope of the DEC-MDP framework. In Section 4, we present in detail our gradient-descent policy search approach for solving scheduling problems by learning stochastic dispatch policies. Moreover, we suggest a notification-based mechanism for enhancing the capabilities of purely reactively acting agents, yielding the creation of active schedules from beyond the class of non-delay schedules (Section 5). The remaining part of this article is devoted to an empirical evaluation (Section 6) of our approach as well as to related work and conclusion.

2. DEC-MDPs with Changing Action Sets and Partially Ordered Dependencies

Recently, several researchers have focussed on the task of finding subclasses of DEC-MDPs that feature provably lower complexity than the general problem which is NEXP-complete. Among those, there is the class of DEC-MDPs with changing action sets and partially ordered transition dependencies (Gabel and Riedmiller 2008), where the actions of an arbitrary number of agents may influence, besides their own, the state transitions of maximally one other agent. We will show that this class is well-suited for the type of

[1]OR-Library, `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`, last accessed 08/2010

scheduling problems we are focusing on in this work and we start off by outlining the basic properties of this class.

2.1. *Problem Setting*

Decentralized MDPs with changing action sets and partially ordered dependencies build upon the DEC-MDP framework by Bernstein *et al.* (2002). A factored m-agent DEC-MDP M is defined by a tuple $\langle Ag, S, A, P, R, \Omega, O\rangle$ with

- $Ag = \{1, \ldots, m\}$ as the set of agents,
- S as the set of world states which can be factored into m components $S = S_1 \times ... \times S_m$ (the S_i belong to one of the agents each),
- $A = A_1 \times ... \times A_m$ as the set of joint actions performed by the agents ($a = (a_1, \ldots, a_m) \in A$ denotes a joint action that is made up of elementary actions a_i taken by agent i),
- P as transition function with $P(s'|s, a)$ denoting the probability that the system arrives at 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',
- $\Omega = \Omega_1 \times ... \times \Omega_m$ as the set of all observations of all agents ($o = (o_1, \ldots, o_m) \in \Omega$ denotes a joint observation with o_i as the observation for agent i), and O as the observation function that determines the probability $O(o_1, \ldots, o_m|s, a, s')$ 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. A joint policy π is a set of local policies $\langle\pi_1, \ldots, \pi_m\rangle$ each of which is a mapping from agent i's sequence of local observations and local actions to probabilities of executing the respective action, i.e. $\pi_i : \overline{\Omega}_i \times A_i \to \mathbb{R}$. Subsequently, we allow each agent to fully observe its local state. Being provided with local state information only, however, vast parts of the global state are hidden from each of the agents. 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. $P(o_i|s, a, s', (o_1 \ldots o_{i-1}, o_{i+1} \ldots o_m)) = P(o_i|s_0,\ s_i, a_i, s'_i)$. Then, in combination with local full observability, the observation-related components Ω and O are redundant. While the DEC-MDPs of our interest are observation 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 that determine the way local actions exert influence on other agents' states.

An m-agent DEC-MDP with factored state space $S = S_1 \times ... \times S_m$ 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 $\mathcal{A}_i = \{\alpha_0, \alpha_{i1} \ldots \alpha_{ik}\}$, 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. The former influence an agent's local state only, the latter may additionally influence the state transitions of other agents. As noted, 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. So, a factored m-agent DEC-MDP is said to have *partially ordered transition dependencies*, if there exist dependency functions σ_i for each agent i with

(1) $\sigma_i : \mathcal{A}_i^r \rightarrow Ag \cup \{\emptyset\}$ and
(2) $\forall \alpha \in \mathcal{A}_i^r$ the directed *dependency graph*

$$G_\alpha = (Ag, E) \text{ with } E = \{(j, \sigma_j(\alpha)) | j \in Ag\} \quad (1)$$

is acyclic and contains one directed path

and it holds

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

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 dependency functions σ_i indicate the state of which other agent is affected when agent i takes a local action. Further, condition 2 from above 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. Gabel and Riedmiller (2008) show that, for the class of problems considered, any local action may appear only once in an agent's action set and, thus, may be executed only once. Further, it is proved that solving a factored m-agent DEC-MDP with changing action sets and partially ordered transition dependencies is NP-complete.

3. Job-Shop Scheduling as Decentralized Markov Decision Processes

In job-shop scheduling, a set $\mathcal{J}$ of n jobs must be processed on m resources in a predetermined order. Each job j consists of ν_j operations $o_{j,1} \dots o_{j,\nu_j}$. We use function ϱ to denote on which resource a certain operation $o_{j,k}$ must be handled (i.e. on $\varrho(o_{j,k})$), and function δ states the duration $\delta(o_{j,k})$ of operation $o_{j,k}$. 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 article, we concentrate on the most frequently used scheduling objective, i.e. on the goal of minimizing maximum makespan ($C_{max} = \max_j\{f_j\}$), which corresponds to finishing processing as quickly as possible. A common characteristic of typical 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 Pinedo (2002).

In this section, we argue that reactive scheduling allows for adopting a multi-agent perspective on scheduling problems. We discuss potential merits of this approach and show (Section 3.2) that job-shop scheduling problems are well suited to be modelled using the framework of decentralized Markov decision processes with changing action sets and partially ordered transition dependencies that we summarized in Section 2. In so doing,

we provide a motivation and sound foundation for employing multi-agent reinforcement learning techniques for production management and, particularly, scheduling problems.

3.1. *Motivation for Distributed Job-Shop Scheduling*

In scheduling theory, a distinction between *predictive* production scheduling (also called analytical scheduling or offline-planning) and *reactive* scheduling (or online control) is made (Blazewicz *et al.* 1993). A predictive scheduler assumes complete knowledge over all tasks to be accomplished, e.g. over entire production floors, tries to take all constraints into account and aims at finding a globally coherent solution that maximizes an objective function. By contrast, reactive scheduling can be regarded as an approach to making local scheduling and dispatching decisions based on a shorter planning horizon and on less problem knowledge, and as an approach where decisions are taken during execution (which is why it is sometimes referred to as online production control). In particular, it allows for a larger degree of independence between the entities involved in the decision process.

In this article, our focus is explicitly on reactive scheduling. This means, we target problem settings where we assume that there is no central authority that runs some scheduling solution algorithm or where such a centralized approach cannot be applied, but where distributed dispatching agents are utilized. These agents are supposed to improve their behavior in the course of repeated interaction with the plant. This stands in clear contrast to traditional scheduling approaches where a fully specified problem instance is given and solved using a centralized algorithm.

Taking a decentralized approach for production planning is not a new idea: Manufacturing environments have for a long time been known to require distributed solution approaches for finding high-quality solutions, because of their intrinsic complexity and, possibly, due to an inherent distribution of the tasks involved (Wu *et al.* 2005). Accordingly, the natural distributed character of multi-agent systems may be exploited in a purposive manner when addressing scheduling problems. This has led to the application of a number of agent-based approaches to resource allocation and scheduling problems, including market- and auction-based systems, hybrid systems extending standard scheduling methods by agent techniques, as well as reinforcement learning techniques.

As pointed out, our idea of using a multi-agent system with autonomous dispatching agents for a scheduling task corresponds to performing reactive scheduling. On the one hand, this reactive approach may be considered detrimental since, in general, a globally optimal solution cannot be yielded by doing reactive scheduling. By contrast, predictive schedulers – benefiting from full knowledge of the entire scheduling problem to be solved – typically attain the optimum. Moreover, a centralized solution for a given problem instance may be immediately available (after some computation time), whereas the adaptive learning-based approach we are pursuing in this article requires a certain amount of interaction between the learning agents and their environment and, hence, brings about poor scheduling decisions at the beginning of learning. On the other hand, there are several advantages of performing reactive scheduling using a multi-agent approach:

- Reactive scheduling features the advantage of being able to react to unforeseen events (like a machine breakdown or a delay in the finishing of some operation) appropriately and promptly. By contrast, for a centralized, predictive approach it is necessary to do complete re-planning given the changed situation.
- Operations Research has, for the better part, focused on predictive scheduling and

yielded numerous excellent centralized algorithms capable of finding an optimal schedule in reasonable time. This works well for small and medium-sized JSSPs; for larger problem dimensions, however, computational complexity makes the application of centralized algorithms infeasible whereas.

- Many resource allocation or scheduling problems are intrinsically distributed in nature, meaning that there exists no central authority with all the necessary information to formulate and solve a centralized optimization problem. This is bad, if one wants to solve such problems centrally. But, this is not the approach we advocate in this article.

  We neither interpret the scheduling tasks as centralized optimization problems, nor do we aim at solving them centrally. Instead, decentralization and independence between the processing (dispatching) units are core to our approach. Each agent has access only to a fraction of the full problem specification, thus facing the more intricate, yet practically relevant, challenge of acting under partial system observability. As it turns out, in the course of learning our agents learn to handle this restriction and to jointly make sophisticated dispatching decisions.
- From a practical point of view, a centralized control cannot always be instantiated, which is why a decentralized problem interpretation using a multi-agent system, that we are going to adopt, may sometimes also be of higher impact to real-world applications.
- By combining partial solutions as provided by the agents involved for the local dispatching problems they are facing, it may be feasible to find a more efficient solution for the global problem. Although, in this way, generally only sub-optimal results will be obtained – as job-shop scheduling problems are known to be tightly interacting and non-decomposable (Liu and Sycara 1997) – that kind of divide and conquer strategy may be of higher efficiency.
- Further advantages that can be claimed for taking a multi-agent approach to (practical) manufacturing and scheduling problems include increased flexibility, reduced costs, fault tolerance, and the fact that multi-agent systems may facilitate humans and agent-based machinery to work together as colleagues.

Baker (1998) surveys in detail the utility of multi-agent systems for factory control, resource allocation, and scheduling problems. While this author names various different ways for utilizing agents for distributed production control (e.g. for deciding what and how much to produce, or when to release jobs into a factory), we subsequently focus on their use for deciding upon job routing and operation sequencing. In so doing, we associate to each of the m resources an agent i that locally decides which operation to process next. How this idea relates to DEC-MDPs with changing action sets is the subject of the next section.

3.2. *Job-Shop Scheduling as DEC-MDPs with Changing Action Sets*

Job-shop scheduling problems are well suited to be modelled using factored m-agent DEC-MDPs with changing action sets and partially ordered transition dependencies. We reinforce this claim by showing how the components of a JSSP can be employed to construct a corresponding DEC-MDP.

- *Factored World State:* The world state of a job-shop scheduling problem can be factored: We assume that to each of the resources one agent i is associated that observes the local state at its resource and controls its behavior. Consequently, we have as many agents as resources in the JSSP ($|Ag| = m$).

- *Local Full Observability:* The local state s_i of agent i, hence the situation at its resource, is fully observable (thus, local observations and local states are identical). Additionally, the amalgamation of local observations at all resources fully determines the global state of the scheduling problem. Therefore, the system is jointly observable, i.e. it is a DEC-MDP (cf. Section 2.1).
- *Factored Actions:* Actions correspond to the starting of jobs' operations (job dispatching). So, a local action of agent i reflects the decision to further process one particular job (more precisely, the next operation of that job) out of the set of jobs currently waiting at i.
- *Changing Action Sets:* If actions denote the dispatching of waiting jobs for further processing, then, apparently, the set of actions available to an agent varies over time, since the set of jobs waiting at a resource changes. While $A_i \subseteq \mathcal{A}_i^r$ denotes[1] the currently available actions for agent i, $\mathcal{A}_i^r$ is the set of all potentially executable actions for this agent. Hence, $\mathcal{A}_i^r$ corresponds to the set of jobs j that contain an operation $o_{j,k}$ which must be processed on resource r_i, i.e. $\varrho(o_{j,k}) = i$. Accordingly, it holds

$$\mathcal{A}^r = \cup_{i=1}^m \mathcal{A}_i^r = \mathcal{J}. \tag{2}$$

  Furthermore, the local state s_i of agent i is fully described by the changing set of jobs currently waiting for further processing at the resource to which i belongs. Thus, $s_i = A_i$ and $S_i = \mathcal{P}(\mathcal{A}_i^r)$, as required by the definition of DEC-MDPs with changing action sets.
- *Transition Dependencies:* DEC-MDPs with changing action sets and partially ordered transition dependencies feature some structure according to which the agents' local actions may exert influence on the local states of other agents. 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.
- *Dependency Functions:* The order of resources on which a job's operations must be processed in a JSSP is given a priori. These orders imply that, upon executing a local action by processing a job's next operation, the local state of maximally one further agent is influenced. Let $\alpha \in A_i$ (and so $\alpha \in \mathcal{J}$) be the job whose current operation $o_{\alpha,k}$ is processed by resource i. Then, after having finished $o_{\alpha,k}$, the action set A_i of agent i is adapted according to $A_i := A_i \setminus \{\alpha\}$, whereas the action set of agent $i' = \varrho(o_{\alpha,k+1})$ is extended ($A_{i'} := A_{i'} \cup \{\alpha\}$).

  Therefore, we can define the dependency functions $\sigma_i : \mathcal{A}_i^r \to Ag \cup \{\emptyset\}$ (cf. Section 2.1) for all agents i (and resources, respectively) as

$$\sigma_i(\alpha) = \begin{cases} \varrho(o_{\alpha,k+1}) & \text{if } \exists k \in \{1, \ldots, \nu_\alpha - 1\} : \varrho(o_{\alpha,k}) = i \\ \emptyset & \text{else} \end{cases} \tag{3}$$

  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 $\varrho(o_{\alpha,k}) = i$.
- *Dependency Graphs:* Given the no recirculation property (beginning of Section 3) and the definition of σ_i (Equation 3), the directed graph G_α from Equation 1 is indeed acyclic and contains only one directed path. The corresponding proof is provided by Gabel (2009). For job-shop scheduling problems with recirculation the definition of G_α

[1] Recall that $\mathcal{A}_i^r = \mathcal{A}_i \setminus \{\alpha_0\}$ where α_0 represents an idle action.

must be slightly extended; this depicts a rather technical change and is beyond our scope.

Obviously, the ensemble of agents interacting with the DEC-MDP corresponding to a JSSP have to strive for the same goal, namely for optimizing the objective function of the scheduling problem. Consequently, a crucial precondition to enable the agents to learn to make sophisticated dispatching decisions is that the global reward function of the DEC-MDP coincides with the overall objective of scheduling. As indicated at the start of Section 3, in this work our scheduling objective is to minimize the makespan C_{max} of the resulting schedule. Therefore, the DEC-MDP's reward function can be brought into alignment with the scheduling objective, if negative rewards are incurred for every time step the processing has not been finished, yet.

4. Policy Gradient Methods for Scheduling

Policy gradient algorithms have established themselves as the main alternative RL approach besides value function-based RL methods. Omitting the need to enumerate states and being well applicable to multi-agent settings, PG methods represent a natural option for solving decentralized MDPs with changing action sets and, thus, for tackling job-shop scheduling problems that are modelled using that framework. We emphasize that the policy optimization approach for scheduling problems, that we are going to present in the following, is a natural fit for situations were the scheduling task is subject to random perturbations and has to be processed repeatedly. It is also applicable for deterministic scheduling problems, although, the aspect of online adaptation with respect to the processing fluctuations is less relevant, if the scheduling problem at hand is deterministic.

4.1. *Compact Policy Representation*

We assume that each resource is equipped with a learning agent i whose policy is compactly represented by a small set of parameters $\theta^i = (\theta^i_1, \dots, \theta^i_n)$ where all $\theta^i_j \in \mathbb{R}$. In particular, we presume that there is exactly one parameter for each action from $\mathcal{A}^r_i$, i.e. for each job the agent can execute. As a shortcut, we refer to the parameter belonging to $\alpha \in \mathcal{A}^r_i$ by θ^i_α. Accordingly, for a $m \times n$ JSSP (with no recirculation and m operations in each job), we have m agents with n policy parameters, thus a total of mn parameters to fully describe the agents' joint policy.

These parameters form the basis for defining probabilistic agent-specific (dispatching) policies of action. The use of probabilistic policies is essential, if we want to employ a policy gradient learning method, where we will be required to form the derivative of the policy's performance with respect to its parameters. In order to properly calculate or estimate the gradient, variability in the policy's action choice is necessary (Williams 1992).

Let $s_i \subseteq \mathcal{A}^r_i$ be the current state of agent i where, as explained before, s_i is the set containing all operations currently waiting for further processing at resource i. The probability of action $Pr(\alpha|s_i, \theta^i) = \pi_i(\alpha, s_i|\theta^i)$ for policy π_i is for all actions $\alpha \in \mathcal{A}^r_i$

defined according to the Gibbs distribution,

$$\pi_i(\alpha, s_i|\theta^i) = \begin{cases} \frac{e^{-\theta^i_\alpha}}{\sum_{x \in s_i} e^{-\theta^i_x}} & \text{if } \alpha \in s_i \\ 0 & \text{else} \end{cases} \tag{4}$$

so that actions that are currently not available have zero probability of being executed. With respect to scheduling, this probabilistic action selection scheme represents a stochastic and reactive dispatching policy that, when applied, yields the creation of non-delay schedules. Every time $s_i \neq \emptyset$, some action $\alpha \in \mathcal{A}_i^r$ is being executed, i.e. a resource never remains idle when jobs are waiting for further processing. As a consequence, all stochastic policies (for any values of θ^i) reach the terminal state $s^f = (s_1, \ldots, s_m)$ where all jobs have been finished and, hence, for all i it holds that $s_i = \emptyset$. Later (cf. Section 5), we relax the restriction of acting purely reactively in order to be able to create schedules from beyond the class of non-delay schedules, too.

4.2. *Gradient-Descent Policy Learning*

The general idea of policy optimization in RL is to optimize the policy parameter vector[1] θ such that the expected return

$$J(\theta) = \mathbb{E}_\theta \left[\sum_{t=0}^{T} \gamma^t r(t) \right] \tag{5}$$

is maximized. The sequence of states encountered and actions taken in between is called an episode, which ends after T time steps, when having reached the terminal state s^f. The weighting factor γ^t is time-step dependent and uses γ from $[0, 1]$.

For JSSPs, the notion of an episode translates to the dispatching and execution of all jobs' operations until all jobs have entirely been processed. The expected return corresponds to the general objective of scheduling. Since our goal is to minimize $C_{max}(\theta)$, the expected return can be expressed by providing the learning agents with a summed return of $-C_{max}$ when an episode ends (and with a zero reward otherwise). Equivalently, a constant immediate reward of $r(t) = -1$ may be conveyed to the agents per time step or, also equivalent, if the previous action lasted for several time steps, the cumulation of immediate rewards from the previous decision stage $d-1$ to the current one d. Furthermore, since a finite horizon is guaranteed (i.e. s^f is reached regardless of the values of θ) and, thus, no discounting is necessary ($\gamma = 1$), the goal of policy optimization using policy gradient RL for scheduling problems means to optimize θ such that $J(\theta) = \mathbb{E}\left[-C_{max}(\theta)\right]$ is maximized.

Note, that the sequence of states and actions within a scheduling episode in general differs from the sequence of states and actions in a second episode for two reasons: First, the agents' dispatching decisions do not necessarily have to be deterministic. Stochastic dispatch policies may be allowed and, in fact, during learning we are employing such non-deterministic policies (cf. Equation 4). Second, we consider stochastic JSSPs where the durations of single operations may be randomly perturbed, operations thus become available at different times and, hence, different processing orders arise.

---

[1] For ease of notation, we drop the agent-specific index i when speaking about an agent's policy parameters θ^i.

Algorithm 1: Decentralized Policy Search from the Perspective of Agent i

Input: policy π_i initialized randomly by equally-valued parameters θ^i

```
 1: t ← 0, h_i ← [ ] //counter and (empty) sequence of states and actions
 2: while not stop do
 3:    observe s_i(t) //current state
 4:    if s_i(t) ≠ ∅ or s_i(t) = s^f then
 5:       if t > 0 then
 6:          receive global immediate reward r(t − 1) //r(s(t-1),a(t-1),s(t))
 7:          append [s_i(t − 1), α(t − 1), r(t − 1)] to h_i
 8:       endif
 9:       if s_i(t) = s^f then //episode finished
10:          call PolicyUpdate(h_i)
11:          t ← 0, h_i ← [ ]
12:       else
13:          select α(t) ∈ s_i(t) with probability given by π_i
14:          execute α(t)
15:          s_{σ_i(α(t))} ← s_{σ_i(α(t))} ∪ {α(t)} //influence local state of agent σ_i(α(t))
16:          s_i(t) ← s_i(t) \ {α(t)} //job's operation processed
17:          t ← t + 1
18:       endif
19:    endif
```

4.2.1. Algorithm Outline: The Big Picture

The core idea of policy search-based RL is to directly adapt the policy to be learned with respect to its performance. Besides, the key point of cooperative, distributed multi-agent learning is to have independent agents that try to improve their local policies with respect to a common goal. Algorithm **1** reflects both of these points. It provides a straightforward implementation of a procedure that is tailored for policy search RL and it lets a single agent interact with the environment (in our case, with a DEC-MDP corresponding to a scheduling plant) and improve its policy independently. Note that this algorithm is to be executed in parallel and independently by each agent involved; we here describe it from a single agent's perspective only.

Until some external stop signal indicates the end of the entire learning process, agent i alternates between observing its local state (i.e. not the full global system state), choosing actions, and obtaining global immediate rewards. On the occasion of having finished a single episode, as indicated by having entered the terminal state s^f, the agent calls a policy update algorithm (line 10) whose task is to purposively process the experience collected during the recent episode. In the following subsection we develop and analyze a gradient descent variant of such an update algorithm. Thus, Algorithm **1** realizes a rather generic procedure for performing policy search-based reinforcement learning using decentralized control. Basically, it realizes the interaction of a single agent (agent i) with the DEC-MDP with changing action sets at hand and delegates the task of learning to the respective policy update method. External to this algorithm there is, of course, the environment which repeatedly restarts the scheduling process after the terminal state has been reached, initiating the next scheduling episode.

We note that variable t in Algorithm **1** does not necessarily have to correspond to time steps, but merely represents a counter of decision stages experienced by agent i. Thus, this algorithm is capable of handling environments with temporally extended actions, i.e. where the execution of a local action α lasts longer than one time step, such as scheduling problems where a single operation generally lasts more than one time step. For this to happen, the execution of α in line 14 would correspond to a blocking function call and the global reward to be received (line 6) must be a collection of time step-based rewards accrued during the execution of α. In lines 15 and 16 the action sets of agent i and, possibly, of a dependent agent are changed as denoted by the dependency functions defined in Section 2.1.

4.2.2. Gradient Estimation

Policy gradient methods follow the steepest descent on the expected return. This requires that the expected return $J(\theta)$ must be differentiable with respect to the action parameter θ_α for each $\alpha \in \mathcal{A}_i^r$. In our case, each agent must form the derivative of the (negated) makespan with respect to its parameter vector θ^i (again, for better readability we drop index i in the following). It holds

$$\begin{aligned}\nabla_{\theta_\alpha} J(\theta) &= \nabla_{\theta_\alpha} \mathbb{E}_\theta \left[\sum_{t=0}^{T} \gamma^t r(t)\right] = \nabla_{\theta_\alpha} \mathbb{E}_\theta \left[-C_{max}(\theta)\right] \\ &= \nabla_{\theta_\alpha} \int_{e\in\mathcal{E}} Pr(e|\theta)(-C_{max}(e))de \\ &= \mathbb{E}_\theta \left[-C_{max}(e)\nabla_{\theta_\alpha} \ln Pr(e|\theta)\right] \end{aligned} \quad (6)$$

where e refers to an individual scheduling episode with makespan $C_{max}(e)$ and e is generated, using current policy parameters θ, with probability $Pr(e|\theta)$ from the space $\mathcal{E}$ of all possible episodes. Practically, the evaluation of the term $Pr(e|\theta)$ is infeasible. Therefore, it is crucial that $\nabla_{\theta_\alpha} \ln Pr(e|\theta)$ in Equation 6 can be calculated without knowing $Pr(e|\theta)$, because $Pr(e|\theta) = \prod_{t=0}^{T} p(s_i(t+1)|s_i(t), \alpha(t))\pi_i(\alpha(t), s_i(t))$. When forming the log derivative of this term (Peshkin *et al.* 2000), we obtain

$$\nabla_{\theta_\alpha} \ln Pr(e|\theta) = \sum_{t=0}^{T} \nabla_{\theta_\alpha} \ln \pi_i(\alpha(t), s_i(t)|\theta). \quad (7)$$

In order for Equation 7 to be computable efficiently, $\pi_i(\alpha(t), s_i(t)|\theta)$ must be differentiable with respect to each action parameter θ_α as well. For the compact representation of probabilistic scheduling policies with action probabilities calculated according to Equation 4, this naturally holds true because

$$\nabla_{\theta_\alpha} \ln \pi_i(\alpha(t), s_i(t)|\theta) = \begin{cases} 1 - \pi_i(\alpha(t), s_i(t)|\theta) & \text{if } \alpha = \alpha(t) \\ -\pi_i(\alpha(t), s_i(t)|\theta) & \text{else} \end{cases}. \quad (8)$$

An exact computation of the gradient $\nabla_\theta J(\theta)$, which would involve the evaluation of the integral term in Equation 6, becomes quickly intractable as the problem size grows. Therefore, we make use of Monte-Carlo estimates of the gradient similar to related work (Williams 1992, Sutton *et al.* 2000). These estimates are generated from a fixed number

E of scheduling episodes. Practically, this means that procedure `PolicyUpdate(`h_i`)` in Algorithm **1** returns without doing any modifications to the policy parameters, if less than E episodes were experienced under the current policies π_i. After having collected E episodes, however, the gradient estimate is calculated based on this batch of episodic experience and the function call to `PolicyUpdate(`h_i`)` makes changes to the parameter vectors θ which determines the agent's policy.

For the latter, we use the average performance $\bar{J}(\theta) = \frac{1}{E}\sum_{k=1}^{E} -C_{max}(e_k)$ (average makespan) as a simple baseline to reduce the variance of the gradient estimate (a technique proposed by Greensmith *et al.* (2004)). By replacing the integral by a sum over E episodes and by shifting the makespan of episode e_k by the value of the baseline (i.e. $-C_{max}(e_k) - \bar{J}(\theta)$), we arrive at the following expression for component α of the policy parameter gradient estimate

$$g_\alpha = \nabla_{\theta_\alpha} J(\theta) = \frac{1}{E}\sum_{k=1}^{E}\Bigg(\left(-C_{max}(e_k) - \bar{J}(\theta)\right)\cdot\sum_{t=0}^{T_k}\nabla_{\theta_\alpha}\ln\pi_i(\alpha(t), s_i(t)|\theta)\Bigg). \qquad (9)$$

4.2.3. Updating the Policy

Given a gradient estimate g_α determined according to Equation 9, an update of the agents' scheduling policy parameters uses the standard rule

$$\theta^i_\alpha := \theta^i_\alpha + \beta_u g_\alpha \qquad (10)$$

where $\beta_u \in \mathbb{R}^+$ denotes a learning rate. If $u \in \mathbb{N}$ counts the number of policy updates made and $\sum_u \beta_u = \infty$, $\sum_u \beta_u^2 = const$, then the learning process is guaranteed to converge to a local optimum, at least. The PG update scheme outlined resembles the episodic Reinforce gradient estimator (Williams 1992). For various applications, the fact that this algorithm estimates the gradient for a dedicated recurrent state, is problematic. Hence, other algorithms, such as GPOMDP (Baxter and Bartlett 1999) or the natural actor critic NAC (Peters *et al.* 2005), were suggested that overcome the need of identifying a specific recurrent state at the cost of introducing a bias to the gradient estimate and trading this off against reducing variance. Because, such a recurrent state (starting state) is naturally available for the episodic interpretation of scheduling problems we consider, we keep with doing the gradient calculation using the likelihood ratio method described above.

We note that, in our factored DEC-MDP setting all agents attached to the resources act and perform gradient-based policy updates independently. As shown by Peshkin *et al.* (2000), these factored updates made to the agents' policy parameters lead to the same optimum with respect to the performance of the joint policy, as if they were done by a centralized controller. For more details on this and the corresponding proof see Gabel (2009).

5. Inter-Agent Notifications for Delay Schedules

An obvious shortcoming of the approach presented arises from the fact that each agent behaves in a reactive manner. For scheduling tasks, this means that any resource immediately starts the next operation of a waiting job, if the set of waiting jobs is currently not empty ($s_i = \emptyset$). Hence, the group of agents attached to the resources may yield the

creation on non-delay schedules only, although it is well-known that the optimal schedule may very well be a delay one.

From an agent-theoretic point of view, we may say that we deliberately employ independent agents with partial state information, which do not obtain any information related to other agents or concerning state transition dependencies at all. Consequently, they face particular difficulties in assessing the value of their idle action α_0. Specifically, they are incapable of properly distinguishing when it is favorable to remain idle, in spite of $s_i \neq \emptyset$, and when not.

Aiming at the creation of active schedules from beyond the class of non-delay schedules and, hence, demanding i not to behave purely reactively, we have to redefine its local stochastic policy $\pi_i : \mathcal{A}_i^r \times S_i$ (cf. Equation 4) so as to not select actions from $s_i \subseteq \mathcal{A}_i^r$ only, but to facilitate the execution of the idle action α_0 as well. To this end, we assume that the execution of α_0 lasts until s_i is being changed due to the influence of other agents, i.e. until agent i's action set is extended. Apparently, such an approach can easily result in deadlock situations in which all resources remain idle, waiting for new jobs to come in and where, thus, the terminal state s^f is never reached. Therefore, we need to impose some restrictions on the probability of executing α_0. For these reasons, next we enhance the learning agents towards being able to resolve some of their inter-agent dependencies.

5.1. *Resolving State Transition Dependencies*

The probability that agent i's local state moves to s_i' depends on three factors: On that agent's local state s_i, on its current action a_i, as well as on the set $\Delta_i := \{a_j \in \mathcal{A}_j | i = \sigma_j(a_i), i \neq j\}$, i.e. on the local actions taken by agents that may influence agent i's local state transition. Theoretically, if each agent knew the contents of Δ_i all the time, then all state transition dependencies would be resolved, meaning that all local transitions would be Markovian and that local states would represent a sufficient statistic for each agent to behave optimally. Obviously, advertising Δ_i to all agents conflicts with the idea of intentionally using independent agents that partially observe the global state and act independently of one another.

So, for a distributed approach, knowing Δ_i in general is neither desired nor feasible. Nevertheless, we may increase the capabilities of a reactive agent by allowing it to get at least some partial knowledge about Δ_i. For this, we extend a reactive agent's local state from $S_i = \mathcal{P}(\mathcal{A}_i)$ 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 actions currently *not* in the action set of agent i ($s_i \cap z_i = \emptyset$). Given these preconditions, we can define the resolving of a transition dependency between agents i and j: If agent j decides for executing $a_j \in A_j(s_j)$ and $\sigma_j(a_j) = i$, and if s_i is the local state of agent i and its extended local state $\hat{s}_i = (s_i, z_i)$ as described before, then the transition dependency between i and j is said to be resolved, if we enable agent i to add $\{a_j\}$ to z_i. This mechanism of resolving a transition dependency corresponds to letting agent i know (at least some of) those current local actions of other agents by which the local state of i will soon be influenced.

5.2. *Non-Reactive Policies*

Because, for the class of DEC-MDPs we are dealing with, inter-agent interferences are always exerted by changing (extending) another agent's action set, agent i gets to know which further action(s) will soon be available in $A_i(s_i)$ when a dependency is resolved. 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 s_i is changed, which happens when an announced action $a_j \in z_i$ enters its action set s_i and can finally be executed. We redefine agent i's stochastic local policy as

$$\pi_i(\alpha, \hat{s}_i|\theta^i) = \begin{cases} \frac{e^{-\theta^i_\alpha}}{\sum_{x \in s_i} e^{-\theta^i_x} + \sum_{x \in z_i} e^{-\theta^i_x}} & \text{if } \alpha \in s_i \cup z_i \\ 0 & \text{else} \end{cases} \quad (11)$$

for all $\alpha \in \mathcal{A}_i^r$ and extended local states $\hat{s}_i = (s_i, z_i) \in \hat{S}_i$. If, however, an element $\alpha \in z_i$ is selected during execution given the probabilities defined by π, then in fact agent i remains idle, i.e. it executes α_0, until α enters its local state s_i, and after this immediately continues to process α.

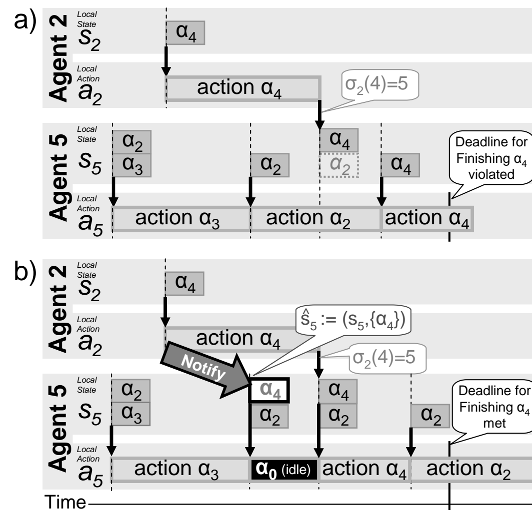

Figure 1. a) Agent 5 behaves purely reactively. b) A notification from agent 2 allows for resolving a dependency, agent 5 may stay willingly idle and the deadline for finishing α_4 is met.

The notification of agent i, which instructs it 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. A simple example for this mechanism is illustrated in Figure 1 where agent 2 notifies agent 5 about having started the execution of a dependent action, which in turn facilitates agent 5 to remain idle and finally meet all deadlines[1]. Generalizing this example, we can say that policies defined over $\mathcal{A}_i \times \hat{S}_i$ are normally more capable than purely reactive ones, because local states $\hat{s}_i$ are extended by information relating to transition dependencies between agents and, hence, at least some information about future local state transitions induced by teammates can be regarded during decision-making.

[1]In this myopic example, action α_4 is assumed to be the last operation of a job whose deadline is at the time denoted in Figure 1.

6. Empirical Evaluation

Imagine a distributed production scenario where, on every day, a number of products must be manufactured across different production sites, each one involving a specific number of processing steps. Although the number of jobs and the processing steps involved as well as the number and characteristics of the processing machines may be known, the calculation and establishment of a fixed schedule may be problematic. On each day, different disturbances and delays may occur in the production process. Under these circumstances, a predictive scheduling approach would be required to (a) collect regularly the updated information about all recent and unforeseen changes in the production process at all resources, to (b) calculate a new schedule in real time, and to (c) communicate it to all local dispatchers. In Section 3.1, we have discussed that any of these three requirements may be unaccomplishable in a practical application, for example, due to the infeasibility of solving the changed scheduling problem in real time, or due to limited communication bandwidth or communication delays between the distributed production sites. Then, it is up to the local decision makers to autonomously make appropriate dispatching decisions, given only information about the current situation at the respective local resource.

In this section, we are going to evaluate the performance of our policy gradient-based learning approach in the context of a scenario like the one delineated. In so doing, we take a two stage approach.

Scenario I: First, we employ our gradient-descent policy search approach to a selection of standard job-shop scheduling benchmark problems from the OR Library, ranging from scenarios with 5 resources and 10 jobs to 15 resources and 30 jobs. These problem instances are, by default, deterministic. Therefore, for those deterministic instances, a classical centralized solution algorithm might be applied and would, in most cases, find the optimal schedule. As a consequence, the application of our multi-agent RL approach is indeed feasible, but does not yield a gain compared to a centralized approach. Nevertheless, these experiments serve as a proof of concept and show the principal functioning of our policy gradient learning approach.

In particular, it should be noted that, despite the determinism, the learning agents face a significant challenge as they are provided with local state information only and, hence, lack full knowledge over the entire scheduling problem. Logically, as far as this set of experiments is considered, we compare the performance of the policy gradient-based learning agents only against the theoretical optimum. We did not include a selection of instances of analytical solution methods that aim at solving a job-shop scheduling problem in a centralized manner (like meta-heuristic search procedures such as beam search (Ow and Morton 1988) or genetic algorithms) because these work under superior preconditions compared to local dispatchers. Instead, we subsume such methods by indicating their upper limit, viz by denoting the theoretically optimal solutions of the respective benchmark instances.

Scenario II: In a second series of experiments, we focus on the task set-up for which our decentralized RL approach is primarily intended. As pointed out before, the agents attached to the resources are required to adapt their dispatching behavior over time with respect to the specifics of the scheduling plant. In so doing, they ought to become capable of appropriately reacting to unforeseen events and changes in the processing of the jobs.

Starting with no prior knowledge and, therefore, dispatching all waiting jobs with identical probability initially, the dispatching agents interact with a scheduling environment like the one described at the beginning of this section. More specifically, we also utilize benchmark problems from the OR Library, but modify them in such a manner that a substantial amount of stochasticity is introduced. By repeatedly dispatching the jobs within these stochastic JSSPs, the agents collect experience and improve their behavior over time using the policy gradient reinforcement learning algorithm presented in Section 4. Still, they act and learn under partial state information.

Due to the reactivity of our approach and due to the restriction to making decisions given local observations only, the performance of our distributed learning dispatching agents must be compared most naturally against dispatching priority rules (DPR). DPRs also perform reactive scheduling and consider only the local situation at the resource for which they make a dispatching decision. However, they are not adaptive and cannot account very well for random fluctuations in the processing.

6.1. *General Experiment Settings*

Given a specific $m \times n$ job-shop scheduling benchmark instance, we initialize all agents' policies by $\theta^i_\alpha = 0$ for all $i \in \{1, \dots, m\}$ and all $\alpha \in \mathcal{A}^r_i$ such that initially the agents dispatch all waiting jobs with equal probability (such a purely random policy, hence, represents a baseline). Throughout all our experiments, we allow the agents to update their local policies $u_{max} = 2500$ times, where we use a constant learning rate $\beta_u = 0.01$ (cf. Equation 10) that has been settled empirically. Any update to the policy parameters is made after $E = 100$ scheduling episodes have been processed, using the estimate $g_\alpha = \nabla_{\theta^i_\alpha} J(\theta^i)$ of the gradient (Equation 9). The schedule that arises, when each agent always picks those jobs that have highest action probabilities according to its local dispatching policy π_i, is called maximum likelihood schedule (MLS) of a joint policy $\pi(\theta)$.

In the context of learning scenario I with deterministic job-shop scheduling problems, where we want to verify that the learners are in principle capable of minimizing maximum makespan, we focus on three different evaluation criteria.

- First, we are interested in the makespan C^{best}_{max} of the best schedule that has been produced occasionally by the set of probabilistic policies during ongoing learning.

  Clearly, this value is only of minor relevance for evaluating the learning method since it has been achieved by incident. Nevertheless, it hints at what results might be achieved in the further course of learning.
- Second, we are interested in the value of the makespan C^{mls}_{max} of the maximum likelihood schedule that arises when all of the agents select jobs greedily, i.e. choose the action $\arg\max_{\alpha \in \mathcal{A}^r_i} \pi_i(\alpha, s_i|\theta^i)$, at all decision points.

  Clearly, this value is of major relevance for evaluating the learning method because this is also the makespan that would be achieved by the agents, if the learning processes was stopped at that moment and all agents were made to select their actions greedily (and, hence, deterministically), when applying their current policies for the deterministic JSSP at hand.
- Our third concern is the convergence behavior and speed of the algorithm. By convergence we here refer to the case that for all agents' policies and for all states s_i there is an $\alpha \in s_i$ such that $\pi_i(\alpha, s_i|\theta^i) > 1 - \varepsilon$ for some small $\varepsilon > 0$, which implies that the agent's probabilistic policy has approached a nearly deterministic one. This indicates that the agents have finally learned what are the specifics of the respective plant and which are the best actions to be executed in any state. To this end, the best case

arises when there is a $u^\star$ such that for all policy update steps u with $u > u^\star$ it holds that $C_{max}^{best} = C_{max}^{mls}$. However, it may also happen that C_{max}^{mls} converges to another local optimum of a value worse than C_{max}^{best} ($C_{max}^{mls} \gneq C_{max}^{best}$).

Regarding the use of limited inter-agent communication in order to overcome the agents' limitation of being capable of generating non-delay schedules only, it must be acknowledged that this enhancement brings about an aggravation of the learning problem. Since it holds $|\hat{S}_i| \gg |S_i|$, the agents must handle a clearly increased number of local states. Also, the number of actions to be considered in each extended state is equal or larger than in its non-extended counterpart s_i. In order to be able to trade off between the goal of learning policies superior to reactive policies and the rising of the difficulty of the learning task, we introduce an additional parameter $d_{max} \geq 0$ that stands for the maximal number of time steps an agent is allowed to remain idle. Given the current local state $\hat{s_i}(t) = (s_i(t), z_i(t))$, agent i is allowed to execute an $\alpha \in z_i$ (by executing α_0 in fact) only, if the notification regarding α has announced that α enters s_i after maximally d_{max} time steps, i.e. if $\exists \tau > t : \alpha \in s_i(\tau)$ and $\tau - t \leq d_{max}$. This restriction can easily be realized by adapting the first case of Equation 11 appropriately. Thus, when setting $d_{max} = 0$, we again arrive at purely reactive agents, which can generate non-delay schedules only, whereas for $d_{max} \geq \max_{x \in \mathcal{A}} \delta(x)$ (with δ denoting the operations' durations) the communication-based resolving of transition dependencies is fully activated. For the experiments whose results we report in the next sections, we either made use of purely reactive agents ($d_{max} = 0$) or used a value of $d_{max} = 20$.

In the context of learning scenario II with stochastic job-shop scheduling problems, we analyze the capabilities of our adaptive agents in adapting their dispatch behavior with respect to the intrinsic stochasticity of the scheduling plant. In so doing, we perturb the processing times of all operations randomly during execution.

As before, the learning agents start with no prior knowledge and select all waiting jobs with equal probability for further processing, thus implementing a purely random dispatch policy at the beginning. Accordingly, random dispatching represents the baseline against which the performance of the adaptive agents must be compared, and the corresponding question we would like to be answered is whether the learning agents are capable of improving their behavior over this random dispatching behavior. Moreover, we consider a selection of dispatching priority rules. These rules perform reactive scheduling as well and they also make their decisions which job to process next based solely on their local view on the respective resource. Subsequently, we consider five instances of this group: The LPT/SPT rules choose operations with longest/shortest processing times first, the FIFO (first in first out) rule considers how long operations had to wait at some resource. The RANDOM rule picks one of the waiting jobs randomly and, hence, is identical to the behavior of the adaptive agents at the beginning of learning. Finally, the SQNO (shortest queue next operation) rule obtains more than just local state information: It is allowed to consider how long are the waiting queues at other resources and selects that job whose next operation must be processed on the resource with the least number of jobs waiting. While all these dispatching priority rules are static, the dispatching policies of the learning agents are not. Hence, the next interesting question to be answered is whether our policy gradient learning approach yields dispatching policies that are superior to the results that DPRs achieve.

In reporting the learning curves and final performance achieved by the learning agents we always refer to the expected makespan they achieve when they act greedily, i.e. when

they, in any state, pick those jobs with the highest action probabilities given their current stochastic policy $\pi(\theta)$. Hence, during learning the learners choose their actions probabilistically according to Equation 4, but when evaluating their performance for some stochastic JSSP, each agent executes the action with maximal probability (which corresponds to the creation of a maximum likelihood schedule as explained at the beginning of this section), i.e. in state s_i the action

$$\arg\max_{\alpha \in A_i^r} \pi_i(\alpha, s_i|\theta^i)$$

is selected.

Finally, we also investigate the influence of using limited inter-agent communication in the scope of stochastic job-shop scheduling benchmarks. To this end, we make use of the same setting as in the context of scenario I (see above), i.e. we compare purely reactively dispatching agents ($d_{max} = 0$) and communicating agents that are allowed to remain idle for some time ($d_{max} = 20$) and, thus, are capable of creating delay schedules.

6.2. *Experiment Results: Scenario I*

We start the analysis of the behavior of our decentralized learning algorithm by focusing on the FT10 benchmark problem. After that, we broaden the scope by considering an entire suite of problem instances.

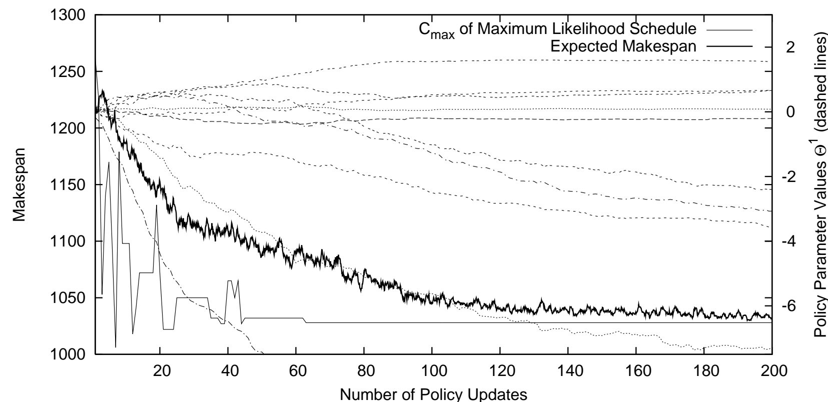

Figure 2. Policy parameters θ^2 of agent 2 during ongoing learning for the FT10 benchmark.

6.2.1. Example Problem

Figure 2 provides an exemplary visualization of what is happening within agent 2 during learning for the deterministic FT10 benchmark. Here, the agents are disallowed to communicate. The dashed lines (secondary ordinate) show the development of policy parameters θ_1^2 through θ_{10}^2 subject to the number of policy updates. Also shown (primary ordinate) are the expected makespan $\mathbb{E}[C_{max}(\theta)]$ of the joint dispatching policy, which of course depends on the other agents' local policies (and, thus, on their $9 \cdot 10$ policy parameters) to a large extent, as well as the makespan C_{max}^{mls} of the maximum likelihood schedule (solid lines). Apparently, the latter two curves are approaching each other which indicates that π_θ is converging towards a deterministic policy. Policy gradient learning methods in general guarantee the achievement of a local optimum. Therefore, as to be

expected, the learners converge to a local optimum only, i.e. the global optimum (best schedule with a makespan of $C_{max}^{opt} = 930$) is not attained.

The overall learning results for the FT10 benchmark can be read from Figure 3 (log scale abscissa). Besides C_{max}^{mls}, here also the makespan of the best schedule encountered intermediately ($C_{max}^{best} = 964$) is shown, and the relation to the starting point of learning (initial, random policies with average makespan of $C_{max}^{init} = 1229$) and to the theoretical optimum ($C_{max}^{opt} = 930$) is highlighted. Additionally, the corresponding C_{max}^{mls} and C_{max}^{best} curves for communicating agents with $d_{max} = 20$ are drawn which obviously outperform purely reactive agents, but require more learning time to achieve that result. The remaining percentual error of the acquired joint policy (converged to the maximum likelihood policy with $C_{max}^{mls} = 993$ after $u^\star = 429$ updates) relative to the optimum is thus 6.8%.

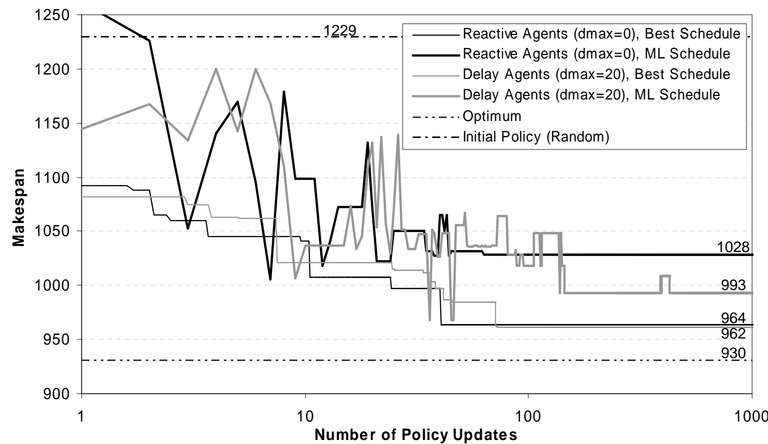

Figure 3. PG learning progress for FT10, opposed for purely reactive and communicating agents.

6.2.2. Benchmark Suites

Table 1 summarizes the learning results for a set of different benchmarks averaged over problems of different $m \times n$ sizes. In any case, the starting point of learning is represented by the initial, random dispatching policy (all $\theta_j^i = 0$) whose relative error $e_r = 100\% \cdot (C_{max}^{init}/C_{max}^{opt} - 1)$ is typically in the range of 20-30%. Starting from this baseline, the error values $e_b = 100\% \cdot (C_{max}^{best}/C_{max}^{opt} - 1)$ for the best intermediate schedule found as well as $e_m = 100\% \cdot (C_{max}^{mls}/C_{max}^{opt} - 1)$ for the maximum likelihood schedule (obtained after u_{max} policy updates) can be decreased significantly.

As can be read from the table, the theoretical optimum is achieved ($e = 0\%$) only occasionally which is to be expected since the PG learning algorithm in general converges to a local optimum. The time to arrive at that local optimum is given by the average number $u^\star$ of policy updates necessary until C_{max}^{mls} does not change any further. In some cases convergence could not be obtained within u_{max} updates which is denoted by a '-' in Table 1 (in brackets: number of problem instances for which convergence was attained). Apparently, the problem aggravation introduced by setting $d_{max} > 0$ brings about a clear reduction of the learning speed, but superior performance (numbers in bold).

6.3. *Experiment Results: Scenario II*

In what follows, we leave the deterministic setting and allow for sudden, unexpected changes in the production process. We introduce stochasticity into the considered job-shop scheduling problems by perturbing the processing times of all operations such that

Table 1. Gradient-descent policy learning results for scheduling benchmarks grouped by problem sizes. Instances ABZ5-9, FT10/20, ORB1-9, and LA01-40 (Beasley 2005) are covered. Error values e_b, e_m, and e_r are in %, $u^\star$ gives average numbers of policy updates, '-' indicates that no convergence was achieved within u_{max} policy update steps.

Size	#Prbl	Optimal	Initial Pol.	$d_{max}=0$			$d_{max}=20$		
$m \times n$		C_{max}	Error e_r	e_b	e_m	$u^\star$	e_b	e_m	$u^\star$
5x10	5	620.2	23.4	1.9	**3.3**	229	1.9	3.9	367
5x15	5	917.6	14.7	0.0	0.1	69	0.0	**0.0**	670
5x20	6	1179.2	15.2	0.2	0.2	226	0.2	0.2	2069
10x10	17	912.5	26.9	4.2	6.1	158	**2.2**	**4.6**	495
10x15	5	983.4	30.7	4.6	5.9	948	**2.5**	**4.2**	1047
10x20	5	1236.2	30.1	2.9	**3.9**	556	**2.4**	4.7	1738
10x30	5	1792.4	18.2	0.0	-	- (3/5)	0.0	-	- (0/5)
15x15	5	1263.2	29.9	6.0	8.2	159	**4.6**	**7.3**	624
15x20	3	676.0	29.9	5.4	7.8	678	**6.8**	-	- (0/3)

$\delta(o_{j,k}) := \delta(o_{j,k}) + \kappa$ with κ chosen randomly from $[0, \delta(o_{j,k})/10]$. As these perturbations are determined online, i.e. during the actual processing of the jobs in the plant, this renders the practical application of a well-established, centralized job-shop scheduling algorithm difficult. Again, we start by an investigation of a single stochastic benchmark instance, before we continue with a larger set of problem instances.

6.3.1. Example Problem

In this experiment, we focus on a stochastically perturbed version of the well-known FT10 benchmark problem where the processing times of all operations were increased randomly. As a consequence, the theoretical optimum in terms of minimal makespan is no longer $C_{max}^{opt} = 930$ as in the deterministic case, but some random value subject to the respective random fluctuations occurring in the operations' processing times.

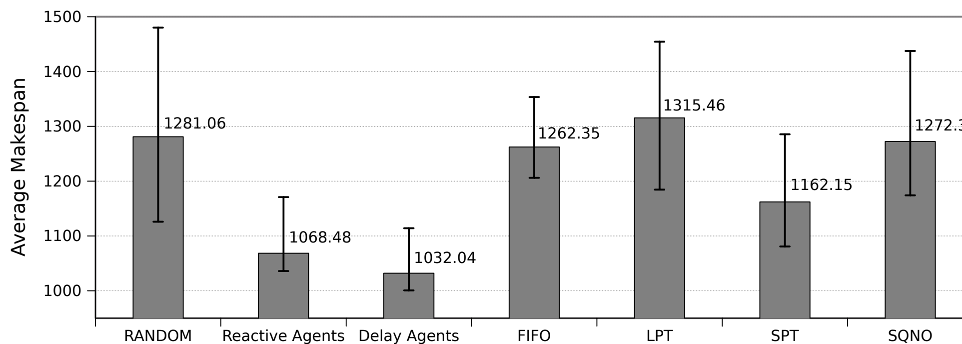

Figure 4. Comparison of PG learning results achieved for the stochastic FT10 benchmark. A random dispatching policy corresponds to the starting point of learning, the bars for reactive ($d_{max} = 0$) and delay ($d_{max} = 20$) agents denote the performance of the distributed policy after 2500 policy updates.

From Figure 4 we can see that a purely random job dispatcher (representing our baseline) would yield an average makespan of 1281.06. Apparently, such a random dispatching strategy is outperformed by the FIFO, SPT, and SQNO rules each of which achieve lower

expected makespans. The LPT rule, by contrast, performs even slightly worse than the baseline. The error bars in that figure indicate the best and worst makespan that has occurred in the course of 100.000 episodes, i.e. within as many repetitions of this stochastic job-shop problem.

Starting with an initially random dispatching policy, the learning agents quickly improve their behavior (Figure 5). It is interesting to note that the first policy update actually decreases the joint performance of the learners (getting worse than the random rule), but that from the second update onward the baseline is clearly superseded, and after already 14 policy updates all static dispatching priority rules are outperformed. Note that a logarithmic abscissa is used and that each data point corresponds to an average of 100 repetitions of the stochastic FT10 problem. As a consequence, in the right part of the figure the impression arises that the noise level in the performance of the DPRs is increasing, but this is due to the fact that there are much more data points shown on narrow space. Indeed, the fluctuations of the performance of static rules is constant throughout the experiment.

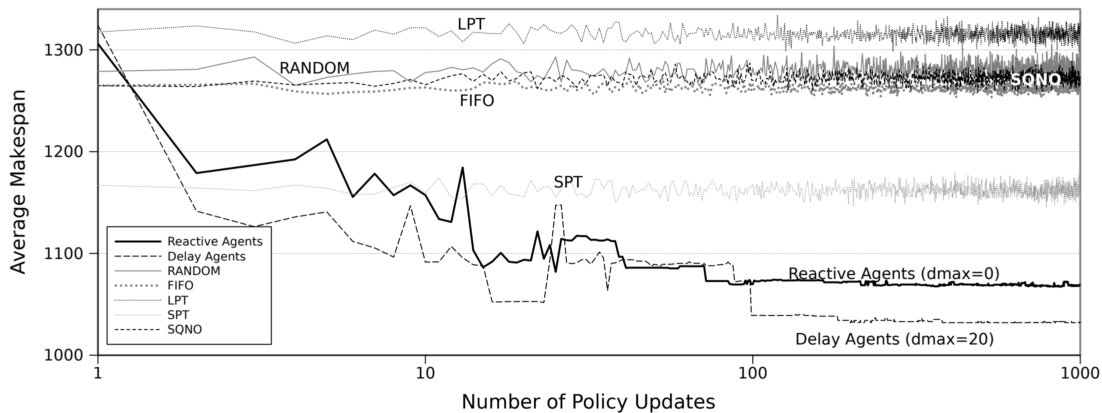

Figure 5. PG learning progress for the stochastic FT10 benchmark, opposed for purely reactive and communicating agents as well as a selection of dispatching priority rules.

After about 100 updates to the agents' policy parameters, almost no more improvements in terms of makespan can be observed. The final performance of the learning approach obtained after 2500 policy updates can be read from the bar chart in Figure 4. As emphasized in Section 6.1, the performance reported here corresponds to the makespan C_{max}^{mls} of a maximum likelihood schedule, which means that the agents choose the action α with highest probability $\pi_i(\alpha, s_i|\theta^i)$ in any state s_i. Purely reactive agents yield an expected makespan of 1068.48 on the stochastic FT10 problem. If the agents are allowed to resolve some of their interdependencies using the communication mechanism we proposed (with $d_{max} = 20$) and, thus, may also create delay schedules, they achieve a makespan of 1032.04 on average.

6.3.2. Benchmark Suites

While the previous example concentrated on one individual stochastic JSSP, we also investigated whether the trend found so far also holds for a larger number of stochastic schedule benchmarks. We selected 15 benchmark problems from the OR Library and introduced stochasticity in all operations' durations as described above. Since the benchmarks considered vary considerably in the lengths of their operations and, hence in the makespan of (optimal) schedules, forming average values over that set of problems would

not be informative. Therefore, we report *relative* makespan values in the following, where we define the average makespan C_{max}^{init} of a random (initial) policy as a baseline and denote it by 100%.

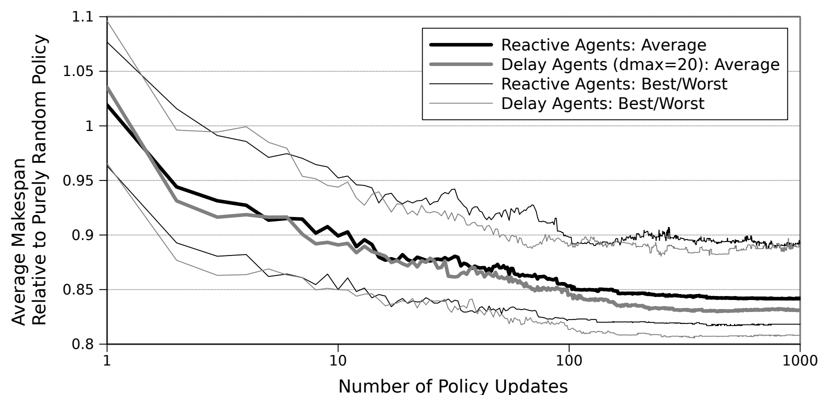

Figure 6. Learning progress for stochastic versions of a set of 15 10×10 benchmarks. Thick lines are averages, thin lines denote best/worst runs. Black curves are for reactive policies ($d_{max} = 0$), gray ones use $d_{max} = 20$. The ordinate plots the makespan achieved relative to the average makespan of a random dispatcher.

In Figure 6, we sketch, averaged over the 15 benchmark problems considered, the makespan achieved by the learning agents relative to the average makespan a purely random dispatcher would achieve and subject to the number of policy updates made. Again, each data point plotted represents the average of 100 episodes experienced by the learners. The corresponding best and worst episodes are shown as well (thin lines).

After the second policy update, the average performance of the learners has already improved by more than 5% compared to their initial policies, after approximately 10 updates this number has increased to 10%, and after circa 100 updates to 15%. The advantage of using communication and, thus, eventually creating delay schedules is more pronounced in later stages of the learning process. This must be tributed to the problem aggravation and state space augmentation introduced with the ability to notify other agents about future incoming jobs.

As outlined in Section 6.1, we allowed the agents to do an update after $E = 100$ episodes of interaction with the plant. For a practical setting, one might consider to do policy updates after significantly less interaction with the environment. Clearly, reducing the value of E increases the variance of the Monte-Carlo gradient estimate (cf. Equation 9) or, stated differently, introduces more noise into the gradient calculation. As a consequence, the policy gradient algorithm performs a more pronounced stochastic gradient descent and, hence, the number of policy updates required to reach a local optimum is increasing likewise. As far as the results reported here are concerned, we employed a comparatively high value of E in order to obtain reliable gradient estimates. In another set of experiments, we observed that, in fact, similar final average performance is achieved by the learning agents with a tenth of interaction ($E = 10$). A more comprehensive discussion of the trade-off between required interaction with the plant versus the correctness of the gradient estimate and its impact on the learning process is provided by Gabel (2009).

Figure 7 summarizes the final average performance achieved by our decentralized policy gradient learning approach. The results are opposed to the results brought about by the DPR representatives we are considering. With a relative average makespan of 84.2%

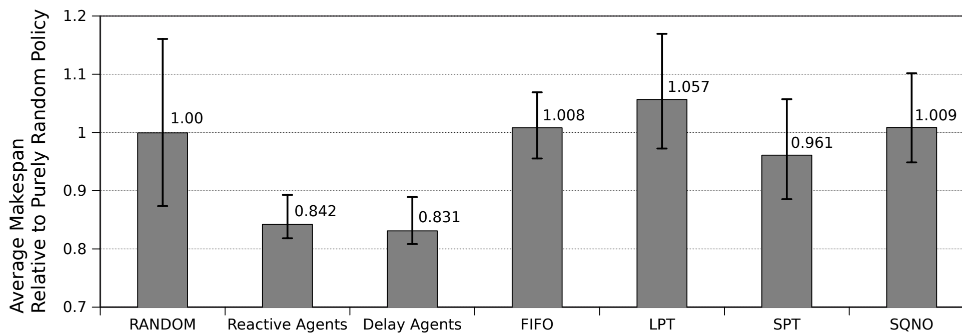

Figure 7. Comparison of the performance of different DPRs averaged over 15 stochastically perturbed JSS benchmark problems. All values are relative to the average makespan a random dispatcher yields. The values reported for the PG learning agents denote their performance after 2500 policy updates.

(83.1% for $d_{max} = 20$), the learning dispatching agents unambiguously grasp the characteristics of the respective scheduling problems and their inherent stochasticity. Note that the type of stochasticity present in the environment is not known to the learners and that the agents achieve these results in an independent manner, using local, i.e. resource-specific information only, and without the existence of a centralized or coordinating entity.

7. Related Work

The utilization of policy gradient methods in the context of distributed problem solving is not new. Building upon the statistical gradient-following policy learning scheme by Williams (1992), Peshkin *et al.* (2000) show that, when employing distributed control of factored actions, it is possible to find at least local optima in the space of the agents' policies. While these authors evaluate their gradient-descent learning algorithm for a simulated soccer game, another prominent application domain targeted by several authors using PG approaches is the task of network routing (Tao *et al.* 2001, Peshkin and Savova 2002), which had previously been examined with the value function-based Q-Routing algorithm (Boyan and Littman 1993). In contrast to these pieces of work, the application of gradient-descent policy search to distributed problems modelled as decentralized MDPs with changing action sets, and in particular for job-shop scheduling problems, as pursued in this paper, is new.

Yagan and Tham (2007) study policy gradient methods for reinforcement learning agents in the DEC-POMDP framework (Bernstein *et al.* 2002), as we do. In order to establish coordination, they define a neighborhood of locally interacting agents which are allowed to fully exchange their local policies. By contrast, using the mechanism for resolving transition dependencies we have proposed, agents dedicatedly notify a single agent about a dependent action they have just taken. With regard to inter-agent communication, the idea of exploiting locality of interaction in distributed systems to optimize a global objective function has already been adopted in the context of dynamic constraint optimization and satisfaction problems (e.g. Modi *et al.*, 2005). Moreover,

job-shop scheduling problems have also been interpreted and solved as constraint optimization problems (e.g. Liu and Sycara, 1995) with the goal of finding an optimal solution through applying a sequence of distributed repair operations. In fact, such an approach bears some resemblance to the repair-based reinforcement learning approach to job-shop scheduling by Zhang and Dietterich (1995), but it is less related to our work since we interpret the scheduling task as a sequential decision problem tackled by independently acting and learning agents.

By contrast, of higher relevance to the article at hand is the work by Aberdeen and Buffet (2007) on PG methods for planning problems. Here, also a factorization of the global policy is made and independently learning (yet, non-communicating) agents are employed for various temporal planning tasks. Another related work that utilizes simple and independent learners, focuses on value function-based RL with neural networks (Gabel and Riedmiller 2007). In that work, we developed a constructive approach to solving multi-agent scheduling problems (not a repair-based one), but did not utilize policy search-based reinforcement learning algorithms, but methods that first learn value functions and induce their policies from those functions. Similarly, Pontrandolfo *et al.* (2002) focus on value function-based average reward reinforcement learning algorithms and apply them for supply chain management problems, also tackling the issue of inter-agent coordination.

8. Conclusion

Nearly all work on solving scheduling problems using approaches enhanced by computational intelligence methods assume full knowledge about the problem and, hence, perform predictive scheduling. In this work, we have proposed a clear departure from this centralized approach, cast job-shop scheduling problems as decentralized Markov decision processes, and proposed a novel algorithm to approximate solutions to this problem.

Policy gradient methods have recently gained much popularity within the RL and distributed AI community. To this end, we have shown how to apply a gradient-descent policy search method for scheduling problems. We have modelled the task of job-shop scheduling as sequential decision process using the framework of factored DEC-MDPs. In so doing, we attached independent and simply structured agents to each of the processing resources which improved their local dispatching policies using an algorithm that updates their policies following the gradient of expected makespan. This distributed approach in general does not allow for finding the best solution of job-shop scheduling instances, but it facilitates discovering near-optimal approximations thereof in little time. To overcome the purely reactive dispatching behavior of the agents, we also suggested a straightforward inter-agent notification mechanism that enables the agents to partially get to know future incoming jobs such that they are allowed to dedicatedly decide to remain idle and, hence, are able create solutions corresponding to delay schedules. Our empirical evaluation using established benchmark problems has demonstrated the effectiveness of our approach for deterministic as well as stochastic job-shop scheduling problems.

The work performed and described in the scope of this article opens a number of opportunities for interesting directions of future research. We have shown that the class of decentralized problems identified in Section 2 matches well with job-shop scheduling problems. We have also pointed to the fact that this class' usability is not restricted to this application area, but can be employed for different scheduling problems and application domains beyond manufacturing as well. Hence, an interesting avenue for

future work is represented by the application of our policy gradient RL algorithm to different scheduling scenarios and application fields, e.g. to network routing or traffic control problems. As pointed out in Section 4, we have applied our algorithm consistently with constant learning rates. However, adaptive methods exist (e.g. the Rprop method, Riedmiller and Braun, 1993) that allow for dynamically changing learning rate vectors as learning proceeds. Since such adaptations typically result in a significant speed-up of the learning progress (Kocsis *et al.* 2006), a combination our approach with the Rprop technique is a promising idea.

Acknowledgements

The authors thank the anonymous reviewers for several comments and suggestions which helped considerably in improving the article.

References

Aberdeen, D. and Buffet, O., 2007. Concurrent Probabilistic Temporal Planning with Policy-Gradients. *In*: *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling* Providence, USA: AAAI Press, 10–17.

Baker, A., 1998. A Survey of Factory Control Algorithms which Can Be Implemented in a Multi-Agent Heterarchy: Dispatching, Scheduling, and Pull. *Journal of Manufacturing Systems*, 17 (4), 297–320.

Baxter, J. and Bartlett, P., 1999. Direct Gradient-Based Reinforcement Learning: I. Gradient Estimation Algorithms. Technical report, Research School of Information Sciences and Engineering, Australian National University. [online] [Last accessed 2009].

Beasley, J., 1990. OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41 (11), 1069–1072.

Beasley, J., 2005. OR-Library, `http://people. brunel.ac.uk/~mastjjb/jeb/info.html.` [online] [accessed, July 2009].

Bernstein, D., *et al.*, 2002. The Complexity of Decentralized Control of Markov Decision Processes. *Mathematics of Operations Research*, 27 (4), 819–840.

Blazewicz, J., *et al.*, 1993. *Scheduling in Computer and Manufacturing Systems*. Berlin, Germany: Springer.

Boyan, J. and Littman, M., 1993. Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. *In*: *Advances in Neural Information Processing Systems* San Francisco, USA: Morgan Kaufmann, 671–678.

Gabel, T., 2009. Multi-Agent Reinforcement Learning Approaches for Distributed Job-Shop Scheduling Problems. Thesis (PhD). University of Osnabrück, Germany.

Gabel, T. and Riedmiller, 2008. 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: MIT Press, 369–376.

Gabel, T. and Riedmiller, M., 2007. Scaling Adaptive Agent-Based Reactive Job-Shop Scheduling to Large-Scale Problems. *In*: *Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)* Honolulu, USA: IEEE Press, 259–266.

Greensmith, E., Bartlett, P., and Baxter, J., 2004. Variance Reduction Techniques for

Gradient Estimates in Reinforcement Learning. *Journal of Machine Learning Research*, 5, 1471–1530.

Kocsis, L., Szepesvári, C., and Winands, M., 2006. RSPSA: Enhanced Parameter Optimization in Games. *In*: *Proceedings of the 11th International Conference Advances in Computer Games (ACG 2006)* Taipei, Taiwain: Springer, 39–56.

Kok, J.R., 2006. Coordination and Learning in Cooperative Multiagent Systems. Thesis (PhD). Faculty of Science, University of Amsterdam.

Liu, J. and Sycara, K., 1995. Exploiting Problem Structure for Distributed Constraint Optimization. *In*: *Proceedings of the First International Conference on Multiagent Systems* San Francisco, USA: The MIT Press, 246–253.

Liu, J. and Sycara, K., 1997. Coordination of Multiple Agents for Production Management. *Annals of Operations Research*, 75 (1), 235–289.

Modi, P., *et al.*, 2005. Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence*, 161 (1), 149–180.

Ow, P. and Morton, T., 1988. Filtered Beam Search in Scheduling. *International Journal of Production Research*, 26, 297–307.

Peshkin, L., *et al.*, 2000. Learning to Cooperate via Policy Search. *In*: *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI 2000)* Stanford, USA: Morgan Kaufmann, 489–496.

Peshkin, L. and Savova, V., 2002. Reinforcement Learning for Adaptive Routing. *In*: *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2002)* Honolulu, USA: IEEE Press, 1825–1830.

Peters, J., Vijayakumar, S., and Schaal, S., 2005. Natural Actor-Critic. *In*: *Proceedings of the 16th European Conference on Machine Learning (ECML 2005)* Porto, Portugal: Springer, 280–291.

Pinedo, M., 2002. *Scheduling. Theory, Algorithms, and Systems.* USA: Prentice Hall.

Pontrandolfo, P., *et al.*, 2002. Global Supply Chain Management: A Reinforcement Learning Approach. *International Journal of Production Research*, 40 (6), 1299–1317.

Riedmiller, M. and Braun, H., 1993. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. *In*: *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, San Francisco, USA, 586–591.

Sutton, R., *et al.*, 2000. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *In*: *Advances in Neural Information Processing Systems 12 (NIPS 1999)* Denver, USA: MIT Press, 1057–1063.

Tao, N., Baxter, J., and Weaver, L., 2001. A Multi-Agent, Policy-Gradient Approach to Network Routing. *In*: *Proceedings of the 18th International Conference on Machine Learning (ICML 2001)* San Francisco, USA: Morgan Kaufmann, 553–560.

Williams, R., 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8, 229–256.

Wu, T., Ye, N., and Zhang, D., 2005. Comparison of Distributed Methods for Resource Allocation. *International Journal of Production Research*, 43 (3), 515–536.

Yagan, D. and Tham, C., 2007. Coordinated Reinforcement Learning for Decentralized Optimal Control. *In*: *Proceedings of the IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)* Honolulu, USA: IEEE Press, 296–302.

Zhang, W. and Dietterich, T., 1995. A Reinforcement Learning Approach to Job-Shop Scheduling. *In*: *Proceedings of the International Joint Conference on Artificial Intellience (IJCAI 1995)* Montreal, Canada: Morgan Kaufmann, 1114–1120.