Gradient-Descent Policy Search for Distributed Job-Shop Scheduling Problems*

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

Abstract

We interpret job-shop scheduling problems as sequential decision problems that are handled by independent learning agents. These agents 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.

Introduction

Distributed problem solving in practice is often characterized by a larger number of involved agents and by a factored system state description where the agents base their decisions on local observations. 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, but in a factory it is usually the case that a variety of products is assembled concurrently, which is why an appropriate scheduling of individual operations is of crucial importance.

In this paper, we model the class of scheduling problems outlined 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, where we attach 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 learning approach employs policy gradient (PG) reinforcement learning (Williams 1992; Sutton et al. 2000) to optimize the agents' policies. In contrast to value function-based reinforcement learning (RL), this class of algorithms does not estimate state-action values and, thus, does not consume amounts of memory proportional to the state space size. 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 that can also directly be mapped to a complete selection of a disjunctive graph for the problem at hand. 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. that 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 (JSS)

*This work has been supported by the Deutsche Forschungsgemeinschaft (DFG) under grant number Ri 923/2-3.

benchmark problems from the OR Library (Beasley 2005). 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.

In the next section, we start off by summarizing the key characteristics of the sub-class of DEC-MDPs with changing action sets and show how job-shop scheduling problems can be modelled within the scope of this framework. Subsequently, 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. The remaining part of this paper is devoted to an empirical evaluation as well as to related work and conclusion.

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

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

- $Ag = \{1, \dots, m\}$ as the set of agents,
- S as the set of world states which can be factored into m components $S = S_1 \times \dots \times S_m$ (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 ($a = (a_1, \dots, a_m) \in A$ 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',
- $\Omega = \Omega_1 \times \dots \times \Omega_m$ as the set of all observations of all agents ($o = (o_1, \dots, o_m) \in \Omega$ denotes a joint observation with o_i as the observation for agent i), and O as the observation function that determines the probability $O(o_1, \dots, o_m|s, a, s')$ that agent 1 through m perceive observations o_1 through o_m upon the execution of a in s and entering s'. Moreover, M is jointly fully observable, i.e. the current state is entirely determined by the amalgamation of all agents' observations: if $O(o|s, a, s') > 0$, then $Pr(s'|o) = 1$.

To the agent-specific components $s_i \in S_i$, $a_i \in A_i$ and $o_i \in \Omega_i$, it is referred as the local state, action, and observation of agent i. A joint policy π is a set of local policies $\langle \pi_1, \dots, \pi_m\rangle$ each of which is a mapping from agent i's sequence of local observations and local actions to probabilities of executing the respective action, i.e. $\pi_i : \overline{\Omega}_i \times A_i \rightarrow \mathbb{R}$. 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. $P(o_i|s, a, s', (o_1 \dots o_{i-1}, o_{i+1} \dots o_m)) = P(o_i|s_0, s_i, a_i, s'_i)$. Then, in combination with local full observability, the observation-related components Ω and O are redundant. 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 $S = S_1 \times \dots \times S_m$ is said to feature *changing action sets*, if the local state of agent i is fully described by the set of actions currently selectable by that agent ($s_i = A_i \setminus \{\alpha_0\}$) and A_i is a subset of the set of all available local actions $\mathcal{A}_i = \{\alpha_0, \alpha_{i1} \dots \alpha_{ik}\}$, thus $S_i = \mathcal{P}(\mathcal{A}_i \setminus \{\alpha_0\})$. Here, α_0 represents a null action that does not change the state and is always in A_i. Subsequently, we abbreviate $\mathcal{A}_i^r = \mathcal{A}_i \setminus \{\alpha_0\}$.

Concerning state transition dependencies, one can distinguish between dependent and independent local actions. 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* $G_\alpha = (Ag, E)$ with $E = \{(j, \sigma_j(\alpha))|j \in Ag\}$ 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 is NP-complete.

Job-Shop Scheduling Problems as DEC-MDPs

In job-shop scheduling, n jobs must be processed on m resources in a pre-determined 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 this paper, 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 ($\nu_j = m$).

JSS problems are very well suited to be modelled using the framework of factored m-agent DEC-MDPs with changing action sets and partially ordered transition dependencies:

- The world state can be factored: We assume that to each of the resources one agent i is associated whose local action is to decide which waiting job to process next.
- The local state of i can be fully described by the changing set of jobs currently waiting for further processing. Since choosing and executing a job represents a local action (i.e. $\mathcal{A}_i^r$ is the set of jobs that must be processed on resource i), it holds that $S_i = \mathcal{P}(\mathcal{A}_i^r)$.
- After having finished an operation of a job, this job is transferred to another resource, which corresponds to influencing another agent's local state by extending that agent's action set.
- The order of resources on which a job's operation must be processed is given in a JSS problem. Therefore, we can define dependency functions $\sigma_i : \mathcal{A}_i^r \rightarrow Ag \cup \{\emptyset\}$ for all agents/resources i as

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

  where k corresponds to the number of that operation within job α that has to be processed on resource i, i.e. k such that $\varrho(o_{\alpha,k}) = i$.
- Given the no recirculation property[1] from above and the definition of σ_i, the directed graph G_α is indeed acyclic with a single directed path.

Apparently, local full observability can easily be granted to all agents, as states are fully described by the respective current action sets.

[1] If recirculation is allowed, then a more complex definition of G_α with duplicated agent vertices guarantees the graph is acyclic.

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 JSS problems that are modelled using that framework.

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 $\mathcal{A}_i^r$, i.e. for each job the agent can execute. As a shortcut, we refer to the parameter belonging to $\alpha \in \mathcal{A}_i^r$ by θ^i_α. Accordingly, for a $m \times n$ JSS problem (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 policies of action. Let $s_i \subseteq \mathcal{A}_i^r$ 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 $\alpha \in \mathcal{A}_i^r$ defined according to the Gibbs distribution,

$$\pi_i(\alpha, s_i|\theta^i) = \begin{cases} \frac{e^{-\theta^i_\alpha}}{\sum_{x \in s_i} e^{-\theta^i_x}} & \text{if } \alpha \in s_i \\ 0 & \text{else} \end{cases} \quad (1)$$

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 scheduling policy that, when applied, yields the creation of non-delay schedules. Every time $s_i \neq \emptyset$, an 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 $s^t = (s_1, \dots, s_m)$ where all jobs have been finished and, hence, for all i it holds that $s_i = \emptyset$. Later, 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.

Illustration: Probabilistic Disjunctive Graphs

Disjunctive graphs $G = (V, C, D)$ are frequently used to represent schedules for JSS problems (Brucker and Knust 2006). The set V of vertices represents the set of all operations, the set C of conjunctions represents the precedence constraints between consecutive operations of the same job, and the set D is meant to represent the different orders in which operations on the same machine might be processed. Fixing the directions of all disjunctions in D, while assuring that G remains acyclic, corresponds to finding a feasible schedule for the JSS problem at hand.

By attaching action choice probabilities to the jobs' operations as described above, we implicitly determine probabilistic directions for the disjunctive arcs defined by π.

When we sample from the joint distribution of probabilistic disjunctive arcs, we obtain a set $\mathcal{S}$ of fixed disjunctions (a complete selection). Additionally, the selection $\mathcal{S}$ is also consistent, i.e. graph $G(\mathcal{S}) = (V, C \cup \mathcal{S})$ is acyclic, if we do the sampling by interacting with the DEC-MDP (corresponding to the JSS problem at hand): Then, at each decision point, an agent decides probabilistically to process an operation of one of the waiting jobs and in so doing assigns an orientation to up to n disjunctive arcs. This way of sampling a selection of orientations for the disjunctive arcs is what we also call a scheduling episode in the following.

We call the schedule that arises when each agent always picks those jobs with the highest action probabilities the maximum likelihood schedule (MLS) of a joint policy $\pi(\theta)$. Generally, the spread of selections that can be sampled varies depending highly on the agents' policy parameters and the corresponding action probabilities. The goal of adapting policy parameters is, of course, to find parameters such that the corresponding MLS is of high quality (low makespan). How this goal can be reached shall be discussed in the next section.

Gradient-Descent Policy Learning

The general idea of policy optimization in RL is to optimize the policy parameter vector[2] θ such that the expected return

$$J(\theta) = \mathbb{E}_\theta \left[\sum_{t=0}^{T} \gamma^t r(t) \right] \tag{2}$$

is optimized. The sequence of states and actions forms a scheduling episode (also called roll-out) that ends after T steps when having reached the terminal state s^t. The weighting factor γ^t is time-step dependent and uses γ from $[0, 1]$.

For JSS problems, the notion of an episode translates to a (simulated) scheduling and execution of all jobs' operations until all jobs have entirely been processed, and 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 $r(s_t) = -C_{max}$ when an episode ends (and with a zero reward otherwise). Furthermore, since a finite horizon is guaranteed (see above) and, thus, no discounting is necessary, 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.

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 $\alpha \in \mathcal{A}_i^r$. It holds

$$\begin{aligned} \nabla_{\theta_\alpha} J(\theta) &= \nabla_{\theta_\alpha} \mathbb{E}_\theta \left[\sum_{t=0}^{T} \gamma^t r(t) \right] = \nabla_{\theta_\alpha} \mathbb{E}_\theta \left[-C_{max}(\theta) \right] \\ &= \nabla_{\theta_\alpha} \int_{e \in \mathcal{E}} Pr(e|\theta)(-C_{max}(e)) de \\ &= \mathbb{E}_\theta \left[-C_{max}(e) \nabla_{\theta_\alpha} \ln Pr(e|\theta) \right] \end{aligned} \tag{3}$$

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

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. It is worth noting that $\nabla_{\theta_\alpha} \ln Pr(e|\theta)$ can be calculated without knowing $Pr(e|\theta)$, because $Pr(e|\theta) = \prod_{t=0}^{T} p(s_i(t+1)|s_i(t), \alpha(t)) \pi_i(\alpha(t), s_i(t))$, and forming the log derivative yields

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

Consequently, $\pi_i(\alpha(t), s_i(t)|\theta)$ 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 1, this naturally holds true as

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

An exact computation of the gradient $\nabla_\theta J(\theta)$ becomes quickly intractable as the problem size grows. Therefore, we revert to determining Monte-Carlo estimates of the gradient similar to related work (Williams 1992; Sutton et al. 2000; Peshkin et al. 2000) generated from a fixed number E of scheduling roll-outs. Finally, 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 (Greensmith, Bartlett, and Baxter 2004). Thus, component α of the policy parameter gradient estimate becomes

$$\begin{aligned} g_\alpha = \nabla_{\theta_\alpha} J(\theta) &= \frac{1}{E} \sum_{k=1}^{E} \Bigg(\left(-C_{max}(e_k) - \bar{J}(\theta) \right) \\ &\quad \cdot \sum_{t=0}^{T_k} \nabla_{\theta_\alpha} \ln \pi_i(\alpha(t), s_i(t)|\theta) \Bigg) \end{aligned} \tag{4}$$

Updating the Policy An update of the agents' scheduling policy parameters uses the standard rule

$$\theta_\alpha^i := \theta_\alpha^i + \beta_u g_\alpha \tag{5}$$

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 estimated 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, Vijayakumar, and Schaal 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 act and perform gradient-based policy updates independently. As shown by Peshkin (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.

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 1) 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^t 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.

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 what it means for a transition dependency between agents i and j to be resolved: If agent j decides on 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 $\hat{s}_i = (s_i, z_i)$ its extended local state 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.

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

$$\pi_i(\alpha, \hat{s}_i|\theta^i) = \begin{cases} \frac{e^{-\theta^i_\alpha}}{\sum_{x \in s_i} e^{-\theta^i_x} + \sum_{x \in z_i} e^{-\theta^i_x}} & \text{if } \alpha \in s_i \cup z_i \\ 0 & \text{else} \end{cases} \tag{6}$$

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.

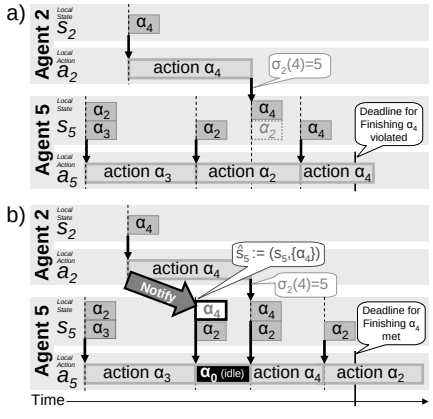

Figure 1: a) Agent 5 behaves purely reactively. b) A notification from agent 2 allows for resolving a dependency, agent 5 may stay willingly idle and meet its deadline.

The notification of agent i, which instructs him to extend

its local state component z_i by a_j, may easily be realized by a simple message passing scheme (assuming cost-free communication between agents) that allows agent i to send a single directed message to agent $\sigma_i(\alpha)$ upon the local execution of α. A simple example for this mechanism is illustrated in Figure 1 where agent 2 notifies 5 about having started the execution of a dependent action, which in turn facilitates agent 5 to remain idle and finally meet all deadlines. 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.

Empirical Evaluation

We employed our gradient-descent policy search approach to a selection of JSS benchmark problems from the OR Library, ranging from scenarios with 5 resources and 10 jobs to 15 resources and 30 jobs. It should be noted that PG methods in general allow for convergence to a local optimum only. Despite that, good approximations of the optimal policy can often be found after relatively few policy update steps.

Experiment Overview

Given a specific $m \times n$ job-shop scheduling benchmark instance, we initialize all agents' policies by $\theta^i_\alpha = 0$ for all $i \in \{1, \ldots, m\}$ and all $\alpha \in \mathcal{A}^r_i$ such that initially the agents dispatch all waiting jobs with equal probability (such a 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 5) that has been settled empirically. Estimating the gradient $g_\alpha = \nabla_{\theta^i_\alpha} J(\theta^i)$ in a Monte-Carlo manner according to Equation 4, we allot $E = 100$ scheduling roll-outs to be performed.

Pursuing the overall goal 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. Second, we are interested in the value of the makespan C^{mls}_{max} of the maximum likelihood schedule (cf. section on Probabilistic Disjunctive Graphs) 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. 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 a $\alpha \in s_i$ 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. To this end, the best case arises when there is a $u^\star$ such that for all policy update steps u with $u^\star < u$ it holds that $C^{best}_{max} = C^{mls}_{max}$. However, it may also happen that C^{mls}_{max} converges to another local optimum of a value worse than C^{best}_{max} ($C^{mls}_{max} \gtrsim C^{best}_{max}$).

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 a new parameter $d_{max} \geq 0$ to the agent policies 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 6 appropriately. Thus, when setting $d_{max} = 0$, we again arrive at purely reactive agents, which for JSS problems could 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 section, we either made use of purely reactive agents ($d_{max} = 0$) or used a value of $d_{max} = 20$.

Experiment Results

Figure 2 provides an exemplary visualization of what is happening within agent 2 during learning for the FT10 benchmark. The dashed lines (secondary ordinate) show the development of policy parameters θ^2_1 through θ^2_{10} subject to the number of policy updates. Also shown are the expected makespan $\mathbb{E}[C_{max}(\theta)]$ of the joint dispatching policy, which of course depends on the other agents' local policies to a large extent, as well as the makespan C^{mls}_{max} 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.

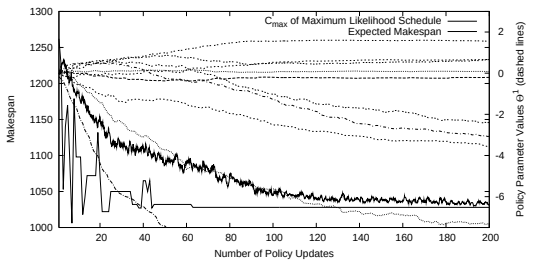

Figure 2: Policy parameters θ^2 of agent 2 during ongoing learning for the FT10 benchmark.

The overall learning results for the FT10 benchmark can be read from Figure 3 (log scale abscissa). Besides C^{mls}_{max}, here also the makespan of the best schedule encountered intermediately ($C^{best}_{max} = 964$) is shown, and the relation to the starting point of learning (initial, random policies with average makespan of $C^{init}_{max} = 1229$) and to the theoretical optimum ($C^{opt}_{max} = 930$) is highlighted. Additionally, the

corresponding C^{mls}_{max} and C^{best}_{max} 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 $C^{mls}_{max} = 993$ after $u^\star = 429$ updates) relative to the optimum is thus 6.8%.

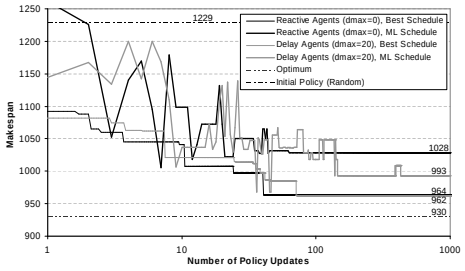

Figure 3: PG learning progress for FT10, opposed for purely reactive and communicating agents.

Table 1 summarizes the learning results for different JSS 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 $\theta^i_j = 0$) whose relative error $e_r = 100\% \cdot (C^{init}_{max}/C^{opt}_{max} - 1)$ is typically in the range of 20-30%. Starting from this baseline, the error values $e_b = 100\% \cdot (C^{best}_{max}/C^{opt}_{max} - 1)$ for the best intermediate schedule found as well as $e_m = 100\% \cdot (C^{mls}_{max}/C^{opt}_{max} - 1)$ for the maximum likelihood schedule (obtained after u_{max} policy updates) can be decreased significantly. 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^{mls}_{max} 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.

In an attempt to investigate the usability of our approach for *stochastic* JSS problems, we perturbed the processing times of all operations such that $\delta(o_{j,k}) := \delta(o_{j,k}) + \kappa$ with κ chosen randomly from $[0, \delta(o_{j,k})/10]$. The effectiveness of gradient-descent policy search for such settings can be seen in Figure 4: Using the makespan of a random (initial) policy as a baseline and denoting its average makespan by 100%, the makespan of the maximum likelihood schedule obtained after 1000 policy updates is being reduced to about 83.1% of its initial value for reactive agents. Using our notification mechanism to resolve inter-agent dependencies with $d_{max} = 20$, this effect can even be improved (reduction to about 80.5%, thick gray curve). It is worth noting that these results are almost as good as the achievements for the deterministic case where for 10×10 problems the value C^{mls}_{max} relative to C^{init}_{max} lies at 83.6% and 82.4%, respectively.

Size $m \times n$	#Prbl	Optimal C_{max}	Initial Pol. Error e_r	$d_{max} = 0$ e_b	e_m	$u^\star$	$d_{max} = 20$ 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)

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.

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, Baxter, and Weaver 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.

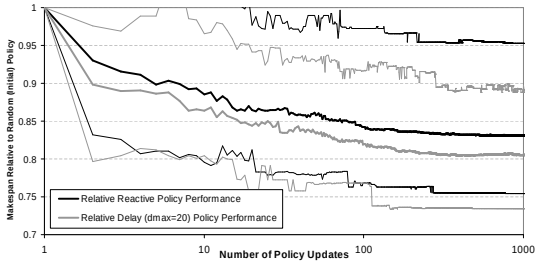

Figure 4: Learning progress for stochastic versions of a set of 17 10×10 benchmarks. Thick lines are averages, thin lines denote best/worst runs. Black curves are for reactive policies ($d_{max} = 0$), gray ones use $d_{max} = 20$.

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 lo-

cally 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 had 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.

By contrast, probably related closest to the paper 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). Since these authors pursue a not repair-based, but constructive approach to solve multi-agent scheduling problems, too, their empirical results are interesting to compare against ours.

Conclusion

Policy gradient methods have recently gained much popularity within the RL and distributed AI community. In this paper, 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 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 JSS 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.

References

Aberdeen, D., and Buffet, O. 2007. Concurrent Probabilistic Temporal Planning with Policy-Gradients. In *Proceedings of ICAPS'07*, 10–17. Providence, USA: AAAI Press.

Baxter, J., and Bartlett, P. 1999. Direct Gradient-Based Reinforcement Learning (Part 1). Technical report, School of Information Sciences, Australian National University.

Beasley, J. 2005. OR-Library, `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`.

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

Boyan, J., and Littman, M. 1993. Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. In *NIPS 1993*, 671–678. Morgan Kaufmann.

Brucker, P., and Knust, S. 2006. *Complex Scheduling*. Berlin Heidelberg: Springer-Verlag.

Gabel, T., and Riedmiller, M. 2007. Scaling Adaptive Agent-Based Reactive Job-Shop Scheduling to Large-Scale Problems. In *Proceedings of the IEEE Symposium CI-Sched 2007*, 259–266. IEEE Press.

Gabel, T., and Riedmiller. 2008. Reinforcement Learning for DEC-MDPs with Changing Action Sets and Partially Ordered Dependencies. In *Proceedings of AAMAS 2008*.

Greensmith, E.; Bartlett, P.; and Baxter, J. 2004. Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning. *Journal of ML Research* 5:1471–1530.

Liu, J., and Sycara, K. 1995. Exploiting Problem Structure for Distributed Constraint Optimization. In *Proceedings of ICMAS'95*, 246–253. San Francisco, USA: The MIT Press.

Modi, P.; Shen, W.; Tambe, M.; and Yokoo. 2005. Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence* 161(1):149–180.

Peshkin, L., and Savova, V. 2002. Reinforcement Learning for Adaptive Routing. In *Proceedings of IJCNN'02*, 1825–1830. Honolulu, USA: IEEE Press.

Peshkin, L.; Kim, K.; Meuleau, N.; and Kaelbling, L. 2000. Learning to Cooperate via Policy Search. In *Proceedings of UAI'00*, 489–496. Morgan Kaufmann.

Peters, J.; Vijayakumar, S.; and Schaal, S. 2005. Natural Actor-Critic. In *Proceedings of ECML 2005*, 280–291. Porto, Portugal: Springer.

Sutton, R.; McAllester, D.; Singh, S.; and Mansour, Y. 2000. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *NIPS 12*, 1057–1063.

Tao, N.; Baxter, J.; and Weaver, L. 2001. A Multi-Agent, Policy-Gradient Approach to Network Routing. In *Proceedings of ICML 2001*, 553–560. San Francisco, USA: Morgan Kaufmann.

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

Yagan, D., and Tham, C. 2007. Coordinated Reinforcement Learning for Decentralized Optimal Control. In *Proceedings of the IEEE Symposium ADPRL 2007*, 296–302.

Zhang, W., and Dietterich, T. 1995. A Reinforcement Learning Approach to Job-Shop Scheduling. In *Proceedings of IJCAI'95*, 1114–1120. Canada: Morgan Kaufmann.