Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)

Scaling Adaptive Agent-Based Reactive Job-Shop Scheduling to Large-Scale Problems

Thomas Gabel and Martin Riedmiller
Neuroinformatics Group
Department of Mathematics and Computer Science, Institute of Cognitive Science
University of Osnabrück, 49069 Osnabrück, Germany
Email: thomas.gabel@uos.de, martin.riedmiller@uos.de

***Abstract*— Most approaches to tackle job-shop scheduling problems assume complete task knowledge and search for a centralized solution. In this work, we adopt an alternative view on scheduling problems where each resource is equipped with an adaptive agent that, independent of other agents, makes job dispatching decisions based on its local view on the plant and employs reinforcement learning to improve its dispatching strategy. We will delineate which extensions are necessary to render this learning approach applicable to job-shop scheduling problems of current standards of difficulty and present results of an adequate empirical evaluation.**

I. Introduction

In production scheduling, tasks have to be allocated to a limited number of resources in such a manner that one or more objectives are optimized. Though various classical approaches can be shown to provide optimal solutions to various scheduling problem variants, they typically do not scale with problem size, suffering from an exponential increase in computation time. In previous work [1], [2], we have explored a novel alternative approach to production scheduling that performs reactive scheduling and is capable of producing approximate solutions in minimal time. Here, each resource is equipped with a scheduling agent that makes the decision on which job to process next based solely on its partial view on the plant. As each agent follows its own decision policy, thus rendering a central control unnecessary, this approach is particularly suitable for environments where unexpected events, such as the arrival of new tasks or machine breakdowns, may occur and, hence, frequent re-planning would be required.

We employ reinforcement learning (RL, [3]) to let the scheduling agents acquire their control policies on their own on the basis of trial and error by repeated interaction within their environment. After that *learning* phase, each agent will have obtained a purposive, reactive behavior for the respective environment. Then, during the *application* phase, e.g. during application in a real plant, each agent can make its scheduling decisions very quickly by utilizing its reactive behavior.

So far, we had empirically evaluated our agent-based reactive scheduling approach merely for randomly created job-shop scheduling scenarios of moderate sizes (e.g. three resources and up to twelve jobs), where usually only one learning agent was involved (the other agents worked according to some fixed dispatching rule). From a scheduling researcher's point of view, however, problem suites of that size would be considered trivial by current standards of difficulty. Therefore, although challenging from a machine learning point of view, the empirical results already published may be regarded as little more than a thorough proof of concept.

In this paper, we focus on a consistent further development of our approach, showing its general applicability and, in particular, its applicability to job-shop scheduling problems on a large scale. Accordingly, the empirical part of this paper is devoted to an evaluation of our multi-agent production scheduling approach to established scheduling benchmark problems from the field of Operations Research.

In Section II, we provide a thorough review of our reinforcement learning approach to learning local dispatching policies in production scheduling. Based on that foundation, in Section III we will develop and discuss necessary extensions of the basic approach that are required to gain its applicability for large-scale benchmark problems. Next, Section IV discusses related work and clarifies similarities and differences between our approach to solving job-shop scheduling problems and other techniques from the field of Artificial Intelligence and Operations Research. In Section V, we present and discuss the results of an empirical evaluation obtained using the methods we suggest. Finally, Section VI summarizes.

II. A Reinforcement Learning Approach to Reactive Job-Shop Scheduling

In this section, we summarize our approach to having learning agents associated to the resources that autonomously acquire dispatching policies and in so doing adapt themselves to the plant structure and inherently consider constraints [1].

A. Basics

In job-shop scheduling n jobs must be processed on m machines in a given order. Each job j_i ($i \in \{1,\ldots,n\}$) consists of v_i operations $o_{j_i,1},\ldots,o_{j_i,v_i}$ that have to be handled on a specific resource for a certain duration. A job is finished after completion of its last operation (completion time c_{j_i}) and it is said to be tardy if it is not finished by its due date d_{j_i}. In general, scheduling objectives to be optimized all relate to the completion times of the jobs. In previous work, we focused on the optimization goal of minimizing summed tardiness $\sum T_j = \sum_{i=1}^{n} max(0, c_{j_i} - d_{j_i})$ which subsumes other variants of the optimization problem, such as

the total completion time criterion or the maximum makespan problems. For reasons of better comparability with other scheduling techniques, in this paper minimizing maximum makespan C_{max} (length of the schedule $C_{max} = max\{c_{j_i}\}$) is our primary optimization goal.

We equip each resource with an agent that determines which job to process next at this resource. The agent's dispatching policy, however, is not fixed but learned autonomously by the agent by getting feedback on the overall dynamic behavior of the system. Compared to common dispatching rules our approach bears the advantage that a virtually arbitrarily complex combination of features describing the situation of the resource and of waiting jobs can be considered.

Classical solution algorithms that perform predictive scheduling are likely to find the optimal solution to a given problem instance (at least, up to certain problem sizes). This is possible because a central control and complete knowledge of the problem and its constraints is assumed (e.g. a corresponding disjunctive graph can be constructed). Unfortunately, the schedule found using an analytical solution method is tailored to one specific problem instance. Knowledge transfer to similar scheduling problems and reactions to unforeseen changes in the environment are not possible without re-planning. So, a main goal of our learning approach to solving scheduling tasks is to allow for a time-efficient computation and total decentralization, while still obtaining high-quality solutions for scheduling problems of current standards of difficulty.

B. MDP Modelling of Production Scheduling

We model the scheduling problem as a Markov Decision Process (MDP) where the system's state $s(t) \in S$ is characterized by the current situation of the plant. This comprises the processing situation of the m resources (we use index $k \in \{1, \ldots, m\}$ to refer to one specific agent/resource subsequently), as well as the processing status of each of the jobs currently in the system. An action $a(t) \in A$ describes the decision of which jobs are to be processed next. The goal of scheduling is to find a policy $\pi^\star$ that minimizes production costs $\mathcal{C}(s, a, t)$ accumulated over time

$$\pi^\star := \min_{\pi} \sum_{t=1}^{C_{max}} \mathcal{C}(s, a, t). \tag{1}$$

Costs may depend on the current situation, as well as on the selected decision, and have to relate closely to the desired optimization goal. For instance, costs may arise when a job is tardy or when a resource remains idle; we will discuss issues of defining an appropriate cost function in Section III-A.

In our approach, the global decision $a(t)$ is a vector of single decisions $a_k(t)$ that are made by agents associated to the m resources. For an agent taking an action means deciding which job to process next out of the set $A_k(t)$ of currently waiting jobs at the corresponding resource k.

C. Local vs. Global View

The global view $s(t)$ on the plant, including the situation at all resources and the processing status of all jobs, would allow some classical solution algorithm to construct a disjunctive graph for the problem at hand and solve it. In this respect, however, we introduce a significant aggravation of the problem: First, we require a reactive scheduling decision in each state to be taken in real-time, i.e. we do not allot arbitrary amounts of computation time. Second, we restrict the amount of state information the agents get. Instead of the global view, each agent k has a local view $s_k(t)$ only, containing condensed information about its associated resource and the jobs waiting there. On the one hand, this partial observability increases the difficulty in finding an optimal schedule. On the other hand, it allows for complete decentralization in decision-making, since each agent is provided with information only that are relevant for making a local decision at resource k.

D. Details of the Learning Algorithm

When there is no explicit model of the environment and of the cost structure available, Q learning [4] is one of the RL methods of choice to learn a value function for the problem at hand, from which a control policy can be derived. The Q function $Q : S \times A \rightarrow \mathbb{R}$ expresses the expected costs when taking a certain action in a specific state. The Q update rule directly updates the values of state-action pairs according to

$$Q(s, a) := (1 - \alpha)Q(s, a) + \alpha(c(s, a, \overline{s}) + \gamma \min_{b \in A(\overline{s})} Q(\overline{s}, b)) \tag{2}$$

where α is the learning rate, γ the discount factor, and where the successor state $\overline{s}$ and the immediate costs $c(s, a, \overline{s})$ are generated by simulation or by interaction with a real process.

Since our approach enforces a distributed decision-making by independent agents, the Q update rule is implemented within each learning agent and adapted to the local decision process ($\alpha = 1$ for better readability):

$$\begin{aligned} Q_k(s_k(t), a_k(t)) \quad := \quad & \mathcal{C}_{sa}(t, \Delta t_k) \\ & + \gamma \min_{b \in A_k(t+\Delta t_k)} Q_k(s_k(t + \Delta t_k), b) \end{aligned} \tag{3}$$

This learning rule establishes a relation between the local dispatching decisions and the overall optimization goal, since the global immediate costs are taken into consideration (e.g. costs caused due to tardy jobs). Since a resource is not allowed to take actions at each discrete time step[1], $\mathcal{C}_{sa}$ collects the immediate global costs arising between t and the next decision time point $t + \Delta t_k$ according to

$$\mathcal{C}_{sa}(t, \Delta t_k) := \sum_{i=t}^{t+\Delta t_k} \mathcal{C}(s, a, i). \tag{4}$$

If we assume convergence of Q_k to the optimal local value function $Q_k^\star$, we obtain a predictor of the expected accumulated global costs that will arise, when in state s_k a job denoted by a_k would be processed next. Then, a policy π that

[1] After having started operation o_{j_i} the resource remains busy until that operation is finished.

exploits Q_k *greedily* will lead to optimized performance of the scheduling agent. A greedy policy chooses as next action

$$a_k(t) := \pi(s_k, a_k, t) = \min_{b \in A_k(t)} Q_k(s_k(t), b). \quad (5)$$

As indicated in the Introduction, we distinguish between the learning and the application phases of the agents' dispatching policies. During the latter, the learned Q_k function is exploited greedily according to Equation 5. During the former, updates to Q_k are made (cf. Equation 4) and an exploration strategy is pursued which chooses random actions with some probability.

When considering a single job-shop problem, the number of possible states is, of course, finite. The focal point of our research, however, is not to concentrate just on individual problem instances, but on arbitrary ones. Hence, we need to assume the domain of Q to be infinite or even continuous, and will have to employ a function approximator to represent it.

E. Goals of this Work

The learning approach sketched so far does not comply with some of the assumptions required by Q learning. First, an agent uses only compressed information of the complete state (local view). However, empirical evidence (e.g. [5], [6] or our previous work [1], [2]) suggests that Q learning can bring about good results in such scenarios. Second, no centralized global decisions are made, but instead the global decision is split into a number of local ones made by independently learning agents. Unless the policies of other agents are stable, the environment as experienced by a single agent appears to be non-stationary. In earlier work, we have circumvented that problem by considering situations primarily where a single learning agent was part of the system. In this paper, we focus on multiple agents learning in parallel to acquire high-quality policies for established scheduling benchmark problems.

The main goal of this work consists of a further development of our reinforcement learning approach to reactive scheduling to make it scalable and applicable to problem instances of current standards of difficulty. We aim at tackling standard scheduling benchmark problems (with ten and more resources and jobs, respectively) and at comparing the performance of our approach to other reactive techniques, as well as to the best known solutions. Consequently, in the next section we identify a number of open questions crucial for its applicability and describe necessary extensions required to render it usable for large-scale scheduling problems.

III. Scaling the Adaptive Scheduling Approach

Challenging job-shop scheduling benchmarks cover problems involving ten and more resources and jobs, respectively. When intending to consistently apply our learning approach to such benchmark problems several extensions to the basic approach have to be introduced.

A. Issues of Task Modelling

A crucial precondition for our adaptive agent-based approach to learning to make sophisticated scheduling decisions is that the global direct costs (as feedback to the learners) coincide with the overall objective of scheduling. We define the global costs $\mathcal{C}$ to be the sum of the costs that are associated with the resources (sum over k) and jobs (sum over i):

$$\mathcal{C}(s, a, t) := \sum_{k=1}^{m} u_k(s, a, t) + \sum_{i=1}^{n} r_{j_i}(s, a, t) \quad (6)$$

When focusing on minimizing overall tardiness, it is possible to set $u_k \equiv 0$ and to let r_{j_i} capture the tardiness $T_{j_i} = max(0, c_{j_i} - d_{j_i})$ of the jobs by

$$r_{j_i}(s, a, t) := \begin{cases} T_{j_i}, & \text{if } j_i \text{ is being finished at } t \\ 0, & \text{else} \end{cases} \quad (7)$$

In this formulation it is not reflected when the tardiness actually occurs. So we prefer the following, equivalent definition that assigns costs at each time step during processing, from which the learning algorithm may benefit:

$$r_{j_i}(s, a, t) := \begin{cases} 1, & \text{if } j_i \text{ is tardy at } t \\ 0, & \text{else} \end{cases} \quad (8)$$

Equations 7 and 8 are no longer useful when the overall objective is to minimize the makespan C_{max} of the resulting schedule. Accordingly, information about tardy jobs or finishing times c_{j_i} of individual jobs provide no meaningful indicator relating to the makespan. However, the makespan of the schedule is minimized, if as many resources as possible are processing jobs concurrently and if as few as possible resources with queued jobs are in the system: Usually, a high utilization of the resources implies a minimal makespan [7]. This argument gives rise to setting $r_{j_i} \equiv 0$ and to defining

$$u_k(s, a, t) := |\{j_i \mid j_i \text{ queued at } k\}| \quad (9)$$

so that high costs are incurred when many jobs, that are waiting for further processing, are in the system and, hence, the overall utilization of the resources is poor.

We represent states $s_k \in S$ and actions/jobs $a_k \in A_k$ by feature vectors generated by the resources' local view. The features have to exhibit some relation to the future expected costs, hence to the makespan, and must allow for a comprehensive characterization of the current situation. Moreover, it is advisable to have features that represent properties of typical problem classes instead of single problem instances, so that acquired knowledge is general and valid for similar problems as well. With respect to the desired real-time applicability of the system, the features should also be easy to compute.

Due to space restrictions, we omit a detailed explanation of all features used, but provide a description on a more abstract level. State features depict the current situation of the resource by describing its processing state and the set A_k of jobs currently waiting at that resource. That job set characterization includes the resource's current workload, an estimation of the earliest possible job completion times, or the estimated makespan. Furthermore, we capture characteristics of A_k by forming relations between minimal and maximal values of certain job properties over the job set (like operation duration times or remaining job processing times). Action features

characterize single jobs a_k from A_k currently selectable by k. Here, we aim at describing makespan-oriented properties of individual jobs (like processing time indices), as well as immediate consequences to be expected when processing that job next, viz the properties of the job's remaining operations (e.g. the relative remaining processing time). Apart from that, action features cover the significance of the next operation $o_{j_i,next}$ of job j_i (e.g. its relative duration).

B. Issues of Value Function Approximation

Since an agent's value function Q_k has an infinite domain, we need to employ some function approximation technique to represent it. In this work, we use multilayer perceptron neural networks to represent the state-action value function. In particular, we aim at exploiting the generalization capabilities of neural networks yielding general dispatching policies, i.e. policies which are not just tuned for the situations encountered during training, but which are general enough to be applied to unknown situations, too. Input to a neural net are the features (cf. Section III-A) describing the situation of the resource as well as single waiting jobs[2]. Thus, the neural network's output value $Q_k(s_k, a_k)$ directly reflects the priority value of the job corresponding to action a_k depending on the current state s_k.

Training Data Utilization: During the learning phase, a set $\mathcal{S}_L$ of scheduling problem instances is given—these problems are processed on the plant repeatedly, where the agents are allowed to schedule jobs randomly with some probability, obtaining new experiences that way. In principle, it is possible to perform an update on the state-action value function according to Equation 4 after each state transition. However, to foster fast improvements of the learned policy by exploiting the training data as efficiently as possible, we revert to fitted Q iteration.

Fitted Q iteration denotes a batch (also termed off-line) reinforcement learning framework, in which an approximation of the optimal policy is computed from a finite set of four-tuples [8]. The set of four-tuples $\mathbb{T} = \{(s^i, a^i, c^i, \overline{s}^i) | i = 1, \ldots, p\}$ may be collected in any arbitrary manner and corresponds to single "experience units" made up of states s^i, the respective actions a^i taken, the immediate costs c^i incurred, as well as the successor states $\overline{s}^i$ entered. The basic algorithm takes $\mathbb{T}$, as well as a regression algorithm as input, and after having initialized $\tilde{Q}$ and a counter q to zero, repeatedly processes the following three steps until some stop criterion becomes true:

1) increment q
2) build up a training set $\mathbb{F}$ for the regression algorithm according to: $\mathbb{F} := \{(in^i, out^i) | i = 1, \ldots, p\}$ where $in^i = (s^i, a^i)$ and $out^i = c^i + \gamma \min_{b \in A(s^i)} \tilde{Q}^{q-1}(\overline{s}^i, b)$
3) use the regression algorithm and the training set $\mathbb{F}$ to induce an approximation $\tilde{Q}^q : S \times A \to \mathbb{R}$

Subsequently, we consider neural fitted Q iteration (NFQ, [9]), a realization of fitted Q iteration where multi-layer neural networks are used to represent the Q function. NFQ is an effective and efficient RL method for training a Q value function that requires reasonably few interactions with the scheduling plant to generate dispatching policies of high quality. We will discuss an adaptation of NFQ to be used in the scope of this work in Section III-C.

Convergence Problems: A critical question concerns the convergence of the learning technique to a (near-)optimal decision policy when used in conjunction with value function approximation. Using neural networks as function approximators, the risk of diverging learning results arises. There are, however, several methods for coping with the danger of non-convergent behavior of a reinforcement learning method. For more details on that topic we refer to [10] and [11].

In order to be able to safely apply our learning approach to reactive scheduling for complex benchmark problems, we rely on *policy screening*, a straightforward, yet computationally intensive method for selecting high-quality policies in spite of oscillations eventually occurring during learning (suggested by Bertsekas and Tsitsiklis [12]): We let the policies generated undergo an additional evaluation (by processing problems from a separate set of screening scheduling problems $\mathcal{S}_S$), which takes place in between single iterations of the NFQ learning algorithm. So, we can determine the actual performance of the policy represented by the Q function in each iteration and, finally, detect and return the best policy created.

C. Issues of Inter-Agent Coordination

In the literature on multi-agent learning, a distinction between joint-action learners and independent learners is made. The former can observe their own, as well as the other agents' action choices. Consequently, in that case the multi-agent MDP can be reverted to a single-agent MDP with an extended action set and be solved by some standard method. Here, however, we concentrate on independent learners because:

1) We want to take a fully distributed view on multi-agent scheduling. The agents are completely decoupled from one another, get local state information, and are not allowed to share information via communication.
2) Decision-making shall take place in a distributed, reactive manner. Hence, no agent will be aware of the jobs being processed next on other resources.
3) The consideration of joint-action learners with global view on the plant would take us nearer to giving all agents the ability to, e.g., construct a disjunctive graph for the scheduling problem at hand and use some classical solution method to solve it. With respect to 1) and 2), this is exactly what we intend to avoid.

We are, of course, aware that the restrictions that we impose on our learning agents depict a significant problem aggravation when compared to the task of finding an optimal schedule with full problem knowledge.

Given the fact that the scheduling benchmarks to which we intend to apply our approach are deterministic, we can employ a powerful mechanism for cooperative multi-agent learning during the learning phase where all agents learn in parallel. Lauer and Riedmiller [13] suggest an algorithm for

[2]In the experiments whose results we describe in Section V, we made use of seven state features and six action features, hence having 13 inputs to the neural network.

distributed Q learning of independent learners using the so-called *optimistic assumption* (OA). Here, each agent *assumes* that all other agents act optimally, i.e. that the combination of all elementary actions forms an optimal joint-action vector. Given the standard prerequisites for Q learning, it can be shown that the optimistic assumption Q iteration rule

$$Q_k(s,a) := \max\{Q_k(s,a), r(s,a) + \gamma \max_{b \in A(\overline{s})} Q_k(\overline{s},b)\} \quad (10)$$

to be applied to an agent-specific local Q function Q_k converges to the optimal $Q^\star$ function in a deterministic environment, if initially $Q_k \equiv 0$ for all k and if the immediate rewards $r(s,a)$ are always larger or equal zero. The basic idea of that update rule is that the expected returns of state-action pairs are calculated in the value of Q_k by successively taking the maximum. For more details on that algorithm and on the derivation of agent-specific policies, we refer to [13].

For the problem settings we are considering in the scope of this paper, we have to take the following two facts into consideration: First, we must use the notion of non-negative costs instead of rewards, so that small Q values correspond to "good" state-action pairs incurring low expected costs. Second, we perform batch-mode learning by first collecting a large amount of training tuples and then calculating updated values to the Q functions using the NFQ training method.

To comply with these two requirements, we developed a batch-mode reinforcement learning method that adapts and combines neural fitted Q iteration and the optimistic assumption Q update rule. First note that in a deterministic environment where scheduling scenarios from a fixed set $\mathcal{S}_L$ of problems are repeatedly processed, the probability of entering some state s_k more than once is larger than zero. If in s_k a certain action $a_k \in A(s_k)$ is taken repeatedly, it may eventually incur very different global costs because of different elementary actions selected by other agents.

So, the integration of the optimistic assumption with a fitted Q iteration learning algorithm can be achieved by modifying the way the training set $\mathbb{F}$ is constructed (see algorithm sketch in Section III-B): Instead of considering all experience tuples from $\mathbb{T}$, only those are utilized for generating $\mathbb{F}$ that have resulted in minimal expected costs. This implies that we *assume* that all other agents have taken optimal dispatching decision that—in combination with the elementary action a_k taken by the considered agent k—are most appropriate for the current state. Accordingly, the *optimistic* target value out^l for some state action pair $in^l = (s^l, a^l)$ (with $s^l = s_k(t)$, $a^l = a_k(t)$ for some t) can be calculated as follows

$$out^l := \min_{\substack{(s^i,a^i,c^i,\overline{s}^i)\in\mathbb{T},\\ (s^i,a^i)=in^l}} \left(c^i + \gamma \min_{b \in A(\overline{s}^i)} Q_k(\overline{s}^i, b) \right) \quad (11)$$

Thus, the target value out^l for some state-action pair $in^l = (s^l, a^l)$ is the minimal sum of the immediate costs and discounted costs to go over all tuples $(s^l, a^l, \cdot, \cdot) \in \mathbb{T}$.

After having constructed the training set $\mathbb{F}$, any suitable neural network training algorithm (e.g. backpropagation) can be employed for the regression task at hand.

We stress that using a neural value function approximation to represent Q and providing agents with local view information only, neither the convergence guarantees for certain types of fitted Q iteration algorithms [8], nor the convergence proof of the OA Q learning algorithm [13] endure. Nevertheless, it is possible to obtain convincing empirical results despite the approximations we employ, as we will show in Section V.

D. Open Issues

Our approach to model the scheduling task as a sequential decision problem and to make reactive scheduling decisions features the disadvantage that currently the resulting schedules correspond to solutions from the set of non-delay schedules, only: If a resource has finished processing one operation and has at least one job waiting, the dispatching agent immediately continues processing by picking one of the waiting jobs.

From scheduling theory, however, it is well-known that for certain scheduling problem instances the optimal schedule may be a delay schedule. In fact, the following subset inclusion holds for three sub-classes of non-preemptive schedules

$$\mathbb{S}_{nondelay} \subsetneq \mathbb{S}_{active} \subsetneq \mathbb{S}_{semiactive} \subsetneq \mathbb{S} \quad (12)$$

where $\mathbb{S}$ denotes the set of all possible schedules [7]. The optimal schedule for a particular problem, however, is always within $\mathbb{S}_{active}$, but not necessarily within $\mathbb{S}_{nondelay}$. As a consequence, our approach will fail to find the optimal solution for many problem instances, but is capable of generating near-optimal schedules from $\mathbb{S}_{nondelay}$. An extension of our learning approach towards delay schedules depicts an important open issue for future work.

IV. Related Work

Job-shop scheduling has received an enormous amount of attention in the research literature. Traditionally, a distinction between predictive production scheduling (also called analytical scheduling or offline-planning) and reactive scheduling (or online control) is made [14]. While the former assumes complete knowledge over the tasks to be accomplished and aims at achieving global coherence in the process of local decision-making, the latter is concerned with making local decisions independently to react to unexpected events or when a global control cannot be instantiated. Most of the approaches that utilize computational intelligence to solve scheduling problems belong to the realm of predictive scheduling.

Classical approaches to solve job-shop scheduling problems cover, for instance, disjunctive programming, branch-and-bound algorithms [15], or the shifting bottleneck heuristic [16]—a thorough overview is given in [17]. Moreover, there is a large number of local search procedures to solve job-shop scheduling problems. These include simulated annealing [18], tabu search [19], as well as squeaky wheel optimization [20]. With a higher degree of relevance to CI, various different search approaches have been suggested based on evolutionary techniques and genetic algorithms (e.g. [21] or [22]).

In contrast to these analytical methods yielding to search for a single problem's best solution, our RL-based approach

belongs to the class of reactive scheduling techniques. Most relevant references for reactive scheduling cover simple as well as complex dispatching priority rules (see [23] or [24]). Focusing on job-shop scheduling with blocking and no-wait constraints, in [25] the authors develop heuristic dispatching rules (such as AMCC, cf. Section V) that are suitable for online control, but that benefit from having a global view on the entire plant when making their dispatch decisions.

Using our reactive scheduling approach, the finally resulting schedule is not calculated beforehand, viz before execution time. Thus, our RL approach to job-shop scheduling is very different from the work of Zhang et al. [26] who developed a repair-based scheduler that is trained using the temporal difference RL algorithm and that starts with a critical-path schedule and incrementally repairs constraint violations.

V. Empirical Evaluation

In this section, we report on the results our adaptive scheduling approach achieves when it is applied to several job-shop scheduling benchmark problems of varying sizes. Problems abz5-9 were generated by Adams et al. [16], problems orb01-09 were generated by Applegate and Cook [15], and finally, problems la01-20 are due to Lawrence [27]. Although these benchmarks are of different sizes, they have in common that no recirculation occurs and that each job has to be processed on each resource exactly once ($v_{j_i} = m, i \in \{1, \ldots, n\}$).

A. Initial Experiment

To start with, we consider the notorious problem ft10 proposed by Fisher and Thompson [28], that had remained unsolved for more than twenty years. Here, during the learning phase, $\mathcal{S}_L = \{p_{ft10}\}$ is processed repeatedly on the simulated plant, where the agents associated to the ten resources follow ε-greedy strategies ($\varepsilon = 0.5$) and sample experience while adapting their behaviors using neural fitted Q iteration with optimistic assumption and in conjunction with the policy screening method (we set $\mathcal{S}_L = \mathcal{S}_S$, i.e. the screening set contains the same problems as the training set).

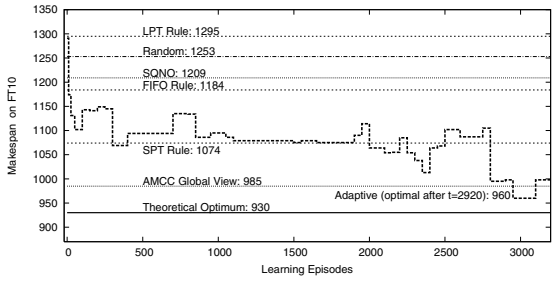

Fig. 1. Learning process for the notorious ft10 problem.

We compare the performance of five different types of dispatching policies:

1) a purely random dispatcher
2) three basic dispatching rules (LPT/SPT choose operations with longest/shortest processing times first, FIFO considers how long operations had to wait at some resource.)
3) two more sophisticated dispatching rules (SQNO is a simplistic heuristic violating the local view restriction by considering information about the queue lengths at the resources where the waiting jobs will have to be processed next. AMCC is a heuristic to avoid the maximum current C_{max} based on the idea of repeatedly enlarging a consistent selection, given a general alternative graph representation of the scheduling problem [25].)
4) our adaptive agent-based approach to reactive scheduling
5) the theoretical optimum ($C_{max,opt} = 930$)

The best solution found by the learning approach was discovered after 2920 repeated processings of $\mathcal{S}_L$ (see Figure 1). The makespan $C_{max} = 960$ of the corresponding non-delay schedule thus has a relative error of 3.2% compared to the optimal schedule. We note that we have detected the optimal learnt dispatching policy (represented by the agents' neural networks representing their Q functions) by means of the policy screening method described in Section III-B.

B. Benchmark Results

Next, we studied the effectiveness of our agent-based scheduling approach for a large number of different-sized benchmark problems, ranging from job-shops with 5 resources and 10 jobs to 15 resources and 20 jobs. We allowed the agents to sample training data tuples in an ε-greedy manner for maximally 25000 processings of $\mathcal{S}_L$ with $\mathcal{S}_L = \mathcal{S}_S$ and permitted intermediate calls to NFQ with optimistic assumption ($N = 20$ iterations of the Q iteration loop) in order to reach the vicinity of a near-optimal Q function as quickly as possible.

In Table I, we compare the capabilities of three different groups of algorithms to the theoretical optimum. Simple dispatching priority rules (group 1) consider only the local situation at the resource for which they make a dispatching decision. The same holds for our adaptive agents approach (3) whose results are given in the table's last two columns. Moreover, two examples of more sophisticated heuristic rules are considered (group 2) that are not subject to that local view restriction. 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 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. Note that the "Remaining Error" of our learning approach is also calculated with respect to the theoretical optimum.

For the 5×15 (la6-10) and 5×20 (la11-15) benchmark problems, the optimal solution can be found by our learning approach in all cases, and for the 5×10 (la1-5), 10×10 (la16-20, orb1-9) sets, only a small relative error of less than ten percent compared to the optimal makespan remains ($3.4/4.0/7.1\%$). As to be expected, dispatching rules, even those disposing of more than just local state information (like

Name & Size	(1)			(2)		Theor.	(3)	Rem.
	FIFO	LPT	SPT	SQNO	AMCC	Optimum	Adaptive	Error
ft6 6×6	65	77	88	73	55	*55*	**57**	3.64
ft10 10×10	1184	1295	1074	1209	985	*930*	**960**	3.23
ft20 5×20	1645	1631	1267	1476	1338	*1165*	**1235**	6.01
abz5 10×10	1467	1586	1352	1397	1318	*1234*	**1293**	4.78
abz6 10×10	1045	1207	1097	1124	985	*943*	**981**	4.03
abz7 15×20	803	903	849	823	753	*667*	**723**	8.40
abz8 15×20	877	949	929	842	783	*670*	**741**	10.60
abz9 15×20	946	976	887	865	777	*691*	**779**	12.74
Avg. abz	1033.6	1124.2	1022.8	1010.2	923.2	*841.0*	**903.4**	8.11
la01 5×10	772	822	751	988	666	*666*	**666**	0.00
la02 5×10	830	990	821	841	694	*655*	**687**	4.89
la03 5×10	755	825	672	770	735	*597*	**648**	8.54
la04 5×10	695	818	711	668	679	*590*	**611**	3.56
la05 5×10	610	693	610	671	593	*593*	**593**	0.00
Avg. $la_{5\times10}$	732.4	829.6	713.0	787.6	673.4	*620.2*	**641.0**	3.40
la06 5×15	926	1125	1200	1097	926	*926*	**926**	0.00
la07 5×15	1088	1069	1034	1052	984	*890*	**890**	0.00
la08 5×15	980	1035	942	1058	873	*863*	**863**	0.00
la09 5×15	1018	1183	1045	1069	986	*951*	**951**	0.00
la10 5×15	1006	1132	1049	1051	1009	*958*	**958**	0.00
Avg. $la_{5\times15}$	1003.6	1108.8	1054.0	1065.4	955.6	*917.6*	**917.6**	0.00
la11 5×20	1272	1467	1473	1515	1239	*1222*	**1222**	0.00
la12 5×20	1039	1240	1203	1202	1039	*1039*	**1039**	0.00
la13 5×20	1199	1230	1275	1314	1161	*1150*	**1150**	0.00
la14 5×20	1292	1434	1427	1438	1305	*1292*	**1292**	0.00
la15 5×20	1587	1612	1339	1400	1369	*1207*	**1207**	0.00
Avg. $la_{5\times20}$	1277.8	1396.6	1343.4	1373.8	1222.6	*1182*	**1182.0**	0.00
la16 10×10	1180	1229	1156	1208	979	*945*	**996**	5.40
la17 10×10	943	1082	924	955	800	*784*	**793**	1.15
la18 10×10	1049	1114	981	1111	916	*848*	**890**	4.95
la19 10×10	983	1062	940	1069	846	*842*	**875**	3.92
la20 10×10	1272	1272	1000	1230	930	*902*	**941**	4.32
Avg. $la_{10\times10}$	1085.4	1151.8	1000.2	1114.6	894.2	*864.2*	**899.0**	3.95
orb1 10×10	1368	1410	1478	1355	1213	*1059*	**1154**	8.97
orb2 10×10	1007	1293	1175	1038	924	*888*	**931**	4.84
orb3 10×10	1405	1430	1179	1378	1113	*1005*	**1095**	8.96
orb4 10×10	1325	1415	1236	1362	1108	*1005*	**1068**	6.27
orb5 10×10	1155	1099	1152	1122	924	*887*	**976**	10.03
orb6 10×10	1330	1474	1190	1292	1107	*1010*	**1064**	5.35
orb7 10×10	475	470	504	473	440	*397*	**424**	6.80
orb8 10×10	1225	1176	1107	1092	950	*899*	**956**	6.34
orb9 10×10	1189	1286	1262	1358	1015	*934*	**996**	6.64
Avg. orb	1164.3	1226.1	1142.6	1163.3	977.1	*898.2*	**962.7**	7.13
Overall Avg.	1054.2	1137.6	1037.3	1080.7	932.9	*874.6*	**908.9**	4.17

TABLE I

LEARNING RESULTS ON OR JOB-SHOP BENCHMARK PROBLEMS.

AMCC or SQNO), are clearly outperformed. For the mixed abz benchmarks involving also instances with 15 resources and 20 jobs per problem, the average relative error increases to 8.1%, yet the rule-based schedulers are surpassed, still.

C. Generalization for Unknown Problems

Some analytical search procedure (like a tabu search) finds a suitable schedule for one specific problem instance. By contrast, our learning approach—after having learned for a set of one or more training problems—will have yielded dispatching policies that are generally applicable. To empirically investigate to which extent the acquired dispatching policies are appropriate for unknown problems, we designed a further experiment. Here, the learning agents were presented three sets of scheduling problems: (a) the training set $\mathcal{S}_L$ for the learning phase, (b) the screening set $\mathcal{S}_S$ for intermediate policy screening rollouts (as before $\mathcal{S}_L = \mathcal{S}_S$), and (c) an application set $\mathcal{S}_A$ containing independent scheduling scenarios to evaluate the quality of the learning results on problem instances the agents have not seen before ($\mathcal{S}_L \cap \mathcal{S}_A = \emptyset$).

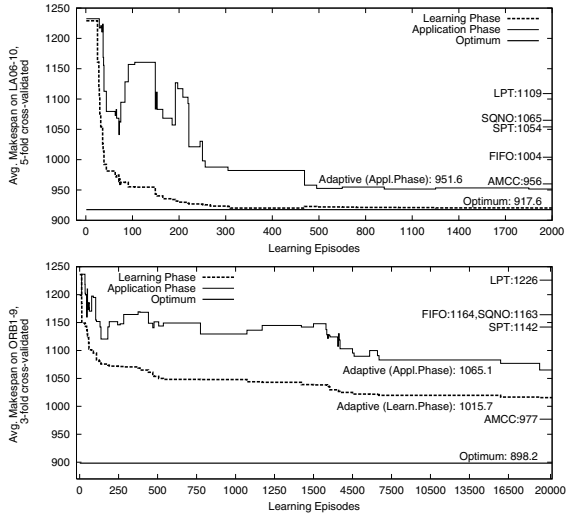

Fig. 2. Using ν-fold cross-validation, the adaptive agents' dispatching policies are trained on the $\mathcal{S}_{la}^{5\times15}$ (top) and $\mathcal{S}_{orb}^{10\times10}$ (bottom) benchmark suites, respectively, and, during the application phase, are evaluated on independent test scenarios. At the chart's right hand side, the average makespans achieved by several static dispatching rules are given for comparison.

Of course, it would be unrealistic to expect the dispatching policies, that were trained using, for instance, a training set with 5×15 problems, to bring about reasonable scheduling decisions for very different problems (e.g. for 10×10 benchmarks). Therefore, we have conducted experiments for benchmark suites $\mathcal{S}$ consisting of problems with identical sizes that were provided by the same authors. From an applicatory point of view, this assumption is appropriate and purposeful, because it reflects the requirements of a real plant where usually variations in the scheduling tasks to be solved occur according to some scheme and depending on the plant layout, but not in an entirely arbitrary manner.

Moreover, since $|\mathcal{S}|$ is rather small under these premises, we performed ν-fold cross-validation on $\mathcal{S}$: We disjoint $\mathcal{S}$ into $\mathcal{S}_L$ and $\mathcal{S}_A$, train on $\mathcal{S}_L$, assess the performance of the learning results on $\mathcal{S}_A$, and repeat that procedure ν times.

Figure 2 illustrates the learning process and the learning results for a benchmark suite of 5×15 problems $\mathcal{S}_{la}^{5\times15} = \{p_{la06}, \ldots, p_{la10}\}$ (top), as well as for the more challenging suite of 10×10 problems $\mathcal{S}_{orb}^{10\times10} = \{p_{orb1}, \ldots, p_{orb9}\}$. For the former, our learning approach succeeds in entirely capturing the characteristics of the training problems in $\mathcal{S}_L$ during training: If the learned dispatching policies process the instances from $\mathcal{S}_L$ the theoretic optimum is reached, i.e. schedule decisions resulting in minimal makespan are yielded. More importantly, even on the independent problem instances from $\mathcal{S}_A$ (5-fold cross-validation) that were not experienced during training, excellent results are achieved: With an average

makespan of 951.6 during the application phase, the acquired policies outperform not just simple dispatching rules, but even those that have full state information (like AMCC). Further, the gap in performance compared to the theoretically best schedules is only 3.7% in terms of average C_{max}.

The local dispatching rules obtained for the $\mathcal{S}_{orb}^{10\times10}$ benchmark suite feature a remaining relative error of 18.6% compared to the theoretic optimum in terms of minimal makespan. Although for these more intricate benchmark problems the results are less impressive, they allow us to draw two empiric conclusions: First, traditional dispatching priority rules that solely employ local state information regarding the respective resource (just as our learning approach does) are clearly outperformed. And, second, the resulting dispatching policies acquired during training feature generalization capabilities and, hence, can effectively be applied to similar, yet unknown, scheduling problem instances.

VI. Conclusion

Job-shop problems are NP-hard. We have pursued an alternative approach to scheduling where each resource is assigned a decision-making agent that decides which job to process next, based on its partial view on the production plant. We use neural reinforcement learning to enable the agents to learn a dispatching policy from repeated interaction within the plant and to adapt their behavior to the environment. This way, we obtain a reactive scheduling system, where the final schedule is not calculated beforehand, viz before execution time, where online dispatching decisions are made, and where the local dispatching policies are aligned with the global optimization goal. Hence, not just the adaptation of the agents' behavior during learning is decentralized, but also decision-making during application proceeds without a centralized control.

The focus of this paper is on necessary enhancements to the basic approach—concerning task modelling, value function representation, and inter-agent coordination—that are required to render our learning approach suitable for larger scale problem sizes. The empirical evaluation on such large-scale benchmark problems leads to the conclusion that problems of current standards of difficulty can very well be effectively solved by the learning method we suggest: In many cases the resulting schedules are optimal with respect to the optimization goal, whereas in other cases the remaining gap in performance is extremely small. Notwithstanding the inherent difficulties in facing partial state observability and agent-independent learning, the dispatching policies acquired do also generalize to unknown situations without retraining.

A disadvantage of our reactive scheduling approach is the fact that only non-delay schedules can be produced, which in many cases prohibits finding optimal schedules. An extension beyond that sub-class of active schedules depicts an important and promising aspect for future work.

Acknowledgment

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

References

[1] S. Riedmiller and M. Riedmiller, "A Neural Reinforcement Learning Approach to Learn Local Dispatching Policies in Production Scheduling," in *IJCAI 99*, Stockholm, Sweden, 1999, pp. 764–771.

[2] T. Gabel and M. Riedmiller, "Reducing Policy Degradation in Neuro-Dynamic Programming," in *Proceedings of ESANN2006*, Bruges, Belgium, 2006, pp. 653–658.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning. An Introduction*. Cambridge, USA: MIT Press/A Bradford Book, 1998.

[4] C. Watkins and P. Dayan, "Q-Learning," *Machine Learning*, vol. 8, 1992.

[5] A. Barto and R. Crites, "Improving Elevator Performance Using Reinforcement Learning," in *Advances in Neural Information Processing Systems 8 (NIPS)*, Denver, USA, 1995, pp. 1017–1023.

[6] H. Kamaya, H. Lee, and K. Abe, "Switching Q-learning in Partially Observable Markovian Environments," in *Proceedings of IROS 2000*. Takamatsu, Japan: IEEE Press, 2000, pp. 1062–1067.

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

[8] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-Based Batch Mode Reinforcement Learning," *Journal of Machine Learning Research*, 2005.

[9] M. Riedmiller, "Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method," in *Machine Learning: ECML 2005*. Porto, Portugal: Springer, 2005.

[10] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro Dynamic Programming*. Belmont, USA: Athena Scientific, 1996.

[11] R. Munos, "Error Bounds for Approximate Policy Iteration," in *Proceedings of the International Conference on Machine Learning 2003*, 2003.

[12] D. Bertsekas, M. Homer, D. Logan, and S. Patek, "Missile Defense and Interceptor Allocation by Neuro-Dynamic Programming," in *Transactions on Systems, Man, and Cybernetics*, vol. 30, 2000, pp. 42–51.

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

[14] J. Blazewicz, K. Ecker, G. Schmidt, and J. Weglarz, *Scheduling in Computer and Manufacturing Systems*. Berlin: Springer, 1993.

[15] D. Applegate and W. Cook, "A Computational Study of the Job-Shop Scheduling Problem," *ORSA Journal on Computing*, vol. 3, pp. 149–156, 1991.

[16] J. Adams, E. Balas, and D. Zawack, "The Shifting Bottleneck Procedure for Job Shop Scheduling," *Management Science*, vol. 34, pp. 391–401, 1988.

[17] E. Pinson, "The Job Shop Scheduling Problem: A Concise Survey and Some Recent Developments," in *Scheduling Theory and Applications*, P. Chretienne, E. Coffman, and J. Lenstra, Eds., 1995, pp. 177–293.

[18] P. van Laarhoven, E. Aarts, and J. Lenstra, "Job Shop Scheduling by Simulated Annealing," *Operations Research*, vol. 40, pp. 113–125, 1992.

[19] E. Nowicki and C. Smutnicki, "A Fast Taboo Search Algorithm for the Job Shop Problem," *Management Science*, vol. 42, pp. 797–813, 1996.

[20] D. Joslin and D. Clements, "Squeaky Wheel Optimization," *Journal of Artificial Intelligence Research*, vol. 10, pp. 353–373, 1999.

[21] J. Bean, "Genetics and Random Keys for Sequencing and Optimization," *ORSA Journal of Computing*, vol. 6, pp. 154–160, 1994.

[22] B. Ombuki and M. Ventresca, "Local Search Genetic Algorithms for the Job Shop Scheduling Problem," *Applied Intelligence*, vol. 21, pp. 99–109, 2004.

[23] S. Panwalkar and W. Iskander, "A Survey of Scheduling Rules," *Operations Research*, vol. 25, pp. 45–61, 1977.

[24] K. Bhaskaran and M. Pinedo, "Dispatching," in *Handbook of Industrial Engineering*, G. Salvendy, Ed., 1977, pp. 2184–2198.

[25] A. Mascis and D. Pacciarelli, "Job-Shop Scheduling with Blocking and No-Wait Constraints," *European Journal of Operational Research*, vol. 143, pp. 498–517, 2002.

[26] W. Zhang and T. Dietterich, "A Reinforcement Learning Approach to Job-Shop Scheduling," in *Proceedings of the International Joint Conference on Artificial Intellience (IJCAI'95)*, 1995, pp. 1114–1120.

[27] S. Lawrence, "Supplement to Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques," Carnegie Mellon University, Pittsburgh, USA, Tech. Rep., 1984.

[28] J. Muth and G. T. (Eds.), *Industrial Scheduling*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1963.