ference on Artificial Intelligence '99
IJCAI'99, Stockholm, Sweden, August 1999

A neural reinforcement learning approach to learn local dispatching policies in production scheduling

Simone Riedmiller
Institut für Werkzeugmaschinen und Betriebstechnik (wbk)
University of Karlsruhe
D-76128 Karlsruhe
simone.riedmiller@mach.uni-karlsruhe.de

Martin Riedmiller
Computer Science Department (ILKD)
University of Karlsruhe
D-76128 Karlsruhe
riedml@ira.uka.de

Abstract

Finding optimal solutions for job shop scheduling problems requires high computational effort, especially under consideration of uncertainty and frequent replanning. In contrast to computational solutions, domain experts are often able to derive good local dispatching heuristics by looking at typical problem instances. They can be efficiently applied by looking at few relevant features. However, these rules are usually not optimal, especially in complex decision situations. Here we describe an approach that tries to combine both worlds. A neural network based agent autonomously optimizes its local dispatching policy with respect to a global optimization goal, defined for the overall plant. On two benchmark scheduling problems, we show both learning and generalization abilities of the proposed approach.

1 Introduction

Production scheduling is the allocation of limited resources to tasks over time, while one or more objectives have to be optimized. Many variants of the basic problem formulation exist, and most of them are NP-hard to solve [Pinedo, 1994], meaning that exact solution algorithms suffer from a non-polynomial increase of computation time. This constitutes a problem not only if the problem to solve surmounts a certain size, but also in moderately complex domains, where the occurrence of new or unexpected events - the arrival of new jobs or the breakdown of machines - makes frequent replanning necessary. Even more, technological changes like semiconductor fabrication or thin film production are posing additional challenges, since new problem structures - like conditional loops in the production process - occur, for which conventional optimization techniques may not be applicable.

An alternative and far less time-consuming way is the application of simple heuristic dispatching rules that select the job to process next on an idle resource depending on the current situation. However, these dispatching rules only reflect heuristic knowledge and do not guarantee to lead to the optimal behaviour of the overall system. Even for experienced human experts it may become arbitrarily difficult to decide which dispatching rule to apply and how to tune it in a certain scenario, since the effects on the dynamics of the overall system can hardly be predicted.

1.1 General Idea

Here we propose an alternative way that allows to combine the desire for (nearly) optimal solutions with a time-efficient computation, provided by resource-coupled dispatching rules. The idea is to have learning agents, that are associated to each resource and determine the local dispatching policy. This policy is not fixed, but instead is autonomously *learned* by getting feedback of the *overall* dynamic behaviour of the production system. In contrast to common dispatching rules which typically only consider few characteristic features of the current situation to make the decision, a learning agent can deal with more state information and therefore figure out more sophisticated policies, which are better tailored to the process. The appearance of the proposed training method does not depend on the optimization goal or constraints posed by the production process. Therefore, it is applicable to a wide range of problem instances. As an example, in this article we focus on job shop scheduling problems with the goal to reduce the summed tardiness. The following summarizes the main characteristics of the proposed approach:

- autonomous acquisition of (approximately) optimal dispatching policies (including the adaptation to the plant structure and the inherent consideration of constraints)

- reusability of acquired knowledge
- fast situation dependent decision making, ability to do reactive scheduling

1.2 Related Work

Recently, several reinforcement learning approaches have been proposed to solve certain aspects of scheduling problems. They vary in the type and their view of the scheduling problem and - as a consequence - the type of control decisions. Zhang and Dietterich [Dietterich and Zhang, 1995] propose an RL approach that learns a neural value-function to guide a repair-based scheduler. An action in this approach is the decision for a certain repair operation. Schneider, Boyan and Moore [Schneider *et al.*, 1998] present a value-function-based approach for the problem of demand based scheduling. The learning scheduler decides over a set of possible factory configurations to maximize expected production profit in the presence of varying demand curves. In contrast to these global approaches, a local multi-agent view of a production scheduling problem is taken in [Brauer and Weiss, 1998]. Each machine does make a local decision which job to process next based on the estimated completion time of the candidate jobs. The learning rule is based on the propagation of these estimates along the production line of a job, similar to the Q-Routing algorithm [Boyan and Littman, 1994]. An application of average-reward RL is presented by Mahadevan and Theocharous [Mahadevan and Theocharous, 1998] for the control of a transfer-line.

Our approach follows the idea of local decision making. The neural network based agent considers the current situation, represented by a set of relevant (local) features, to make its decision which job to process next. It learns a local decision policy (similar to heuristic dispatching rules) with the ability to adapt to the user-defined optimization goal. This is done autonomously, meaning that the policy is self-improved by repeating a certain number of typical training cases.

1.3 Task description

The task considered in the following is to process a set of m jobs on the n resources of a factory. Each job $j \in 1 \ldots m$, consists of a certain number of l_j basic operations $o_{j,1}, \ldots o_{j,l_j}$. A basic operation must be performed on a certain resource $k \in 1 \ldots n$ and has a certain processing time. A job is finished after completion of its last operation. If the completion time C_j of job j is larger than a certain due-date d_j, then the job is said to be *tardy*. The tardiness T_j of a job is zero, if it is finished before or at its due-date; otherwise it is $T_j = C_j - d_j$. In general scheduling objectives to be optimized all relate to the completion time of the jobs. Various variants of the basic optimization problem exist which can be classified within a complexity hierarchy [Pinedo, 1994]. In the following, we look at the problem of minimizing the summed tardiness over all jobs, $T = \sum_j T_j$. Solving this deterministic problem subsumizes also the total completion time problem as well as the maximum lateness or maximum makespan problems. Being NP-hard, it is not possible to solve this problem with a polynomial time algorithm.

2 Description of the solution approach

2.1 Optimization problem and decision making

The global production scheduling problem can be described as a Markov Decision Process (MDP): the system's state $s(t)$ is described by the current situation of the n resources and the processing state of the m jobs, a decision $a(t)$ describes which job is processed next on a waiting resource. The goal of scheduling is to find an optimal policy π^* such that production costs $R(s, a, t)$ accumulated over time are minimized

$$\min_{\pi} \sum_{t=1}^{t^f} R(s, a, t) \qquad (1)$$

where t^f denotes the time after which the last job is finished. Costs $R(s, a, t)$ may depend both on the current situation and on the selected decision and express the desired optimization goal. For example, costs may arise due to the tardiness of a job, due to a resource waiting idle, or due to costs caused by the need to change a tool before continuing processing, and so on.

In our approach, the global decision $a(t)$ is a vector of single decisions $a_k(t)$ made by distributed agents each associated with one of the n resources. If a resource is ready to process a new operation, then its agent chooses one job out of a set A_k of jobs, where A_k is the set of jobs for which the next operation o_{ji} must be processed on resource k. The resource is then occupied for the duration of o_{ji}. After this time, a new decision is made.

2.2 Learning Algorithm

Each agent makes its decision based on a local view $s_k(t) = s_k(s, t)$ of the global plant situation at time t. s_k represents the local view of the agent. It can be thought of compressing the huge amount of global state information into features that are relevant for making the local decision at resource k. Learning here means iteratively improving the decision policy with respect to the optimization of the *global* costs (1). This is done by a *Q-learning* related learning-rule adapted to the local decision process:

$$Q_k^{new}(s_k(t), a_k(t)) \leftarrow \\ R(s, a, t, \triangle_t) + \min_{a \in \mathcal{A}_k(t+\triangle_t)} Q_k(s_k(t+\triangle_t), a)$$

The learning rule relates the local decision process as experienced by agent k to the global optimization goal by considering the *global* direct costs $R(.)$. Since the time between the decisions varies depending on the duration of the currently processed operation, the 'reinforcement term' $R(s, a, t, \triangle_t)$ accumulates the global costs between t and $t + \triangle_t$:

$$R(s, a, t, \triangle_t) := \sum_{\tau=t}^{t+\triangle_t} R(s(\tau), a(\tau))$$

Accordingly, if Q_k converges, then the optimal local value function Q_k^* predicts the estimated accumulated global costs, if in situation s_k the job denoted by a_k would be processed next. A policy greedy with respect to Q_k^*, thus will choose that job next, that will lead to an optimization of the performance of the *overall* plant. The policy of the agent is determined by greedily exploiting the value function,

$$a_k(t) = \pi(s_k, A_k, t) = \min_{a \in A_k(t)} Q_k(s_k(t), a)$$

During learning, a random exploration strategy is performed that deviates from time to time from the current policy.

The agent's learning rule varies from the Q-learning assumptions in two important issues: a) no central global decision is made, but instead, the global decision is composed by individual decisions of time-varying policies. Unless all policies are stable, this makes the process seen by the local agent non-stationary (in contrast, a single Q-learning agent assumes to experience a stationary environment). b) an agent does only use compressed information of the complete state. With an agent's local state information, the observed system behaviour may become stochastic or even unpredictable. However, there is empirical evidence that Q-learning works in this scenario [Barto and Crites, 1996]. In the experiments in section 3 the problems mentioned in a) are circumvented, since in this paper only situations with a single local learning agent are examined.

2.3 Choice of $R(.)$

Since it is our objective to minimize the summed tardiness of all jobs, we have to choose $R(.)$ such, that the minimization of (1) is equivalent to the minimization of the tardiness T. First, $R(.)$ is the sum of the costs $r_j(.)$ associated with the jobs

$$R(s, a, t) = \sum_{j=1}^{m} r_j(s, a, t)$$

Two formulations of r_j are possible. The first is to compute the tardiness *after* the job is finished

$$r_j(t) := \begin{cases} T_j\text{,if job } j \text{ is finished at time } t \\ 0\text{,else} \end{cases} \tag{2}$$

The second possibility is to have costs in each time step *during* processing, if the job is currently too late but not ready yet:

$$r_j(t) := \begin{cases} 1\text{,if job } j \text{ is late \& not finished at } t \\ 0\text{,else} \end{cases} \tag{3}$$

Although both formulations are equivalent with respect to the general problem formulation, the latter choice has the advantage, that the cost function directly reflects the tardiness when it actually occurs, which may help the learning system. Therefore it is used in the experiments in section 3.

2.4 Learning System

We choose a multilayer-perceptron neural network for representing the value function $Q(.)$ for two reasons: a) the state space is continuous and therefore no finite scheme (like a lookup-table) can represent all states. b) we want to exploit the generalization ability of the neural network to find general policies, i.e. the Q function should generalize to unknown situations. Input for the function approximator is an adequate description of the current decision situation by a feature vector. The features have to comply with the following requirements: They should relate to the future expected costs and should be characteristic for the present situation. Features should represent characteristics of typical problem classes rather than of individual instances, such that the acquired knowledge can be generalized to new problem instances. For this reason, few attributes are considered, mainly describing the local situation at the machine. With respect to practical applicability, features should be computable out of data available from common commercial PPC-systems. In order to keep the network input small, redundant information should be avoided. Dealing with a time-dependent problem, besides static properties dynamic features should be concerned. They are not only dependent on the job or resource properties (e.g. processing time) but also on time progress (e.g. slack). Input to the neural network are local state features $s_k(t)$ plus features coding the available decision $a \in A_k(t)$. Possible features are:

state features $s_k(t)$:

- describing general characteristics of the problem: 'tightness' τ (1) and 'distribution' R (2) of the jobs with respect to their due-dates [Pinedo, 1994]
- describing the current situation: estimated tardiness, estimated makespan C (3), average slack

decision features/ job characteristics:

- describing characteristics of job j with respect to the present situation in A_k: e.g. due date index (EDD) (4), relative slack, slack index (MS)(5), relative waiting time (FIFO) (6), relative processing time, processing time index (SPT, LPT)
- describing the immediate consequences, i.e. the properties of the remaining operations if job j was selected: e.g. average remaining slack
- describing the relationship job/ operation, i.e. the significance of operation o_{ij} for job j: e.g. relative work in process, relative buffer

3 Empirical Evaluation

3.1 Research objectives

The following experiments show the behaviour of the proposed learning approach in comparison to heuristic dispatching rules. In detail, we examine the following issues:

- how does the choice of input features influence the optimality of the policy found by the learning system?
- is the learned policy general, i.e. does the policy show good performance when applied to untrained situations?
- does the proposed learning scheme work, i.e. is it possible to improve the decision policy of the local learning agent with respect to the global goal autonomously?
- how does the learned policy perform compared to heuristic dispatching rules?

3.2 Description of the experiments

We examine two cases: a single resource case, as a demonstration for the principle working and performance capabilities, and a multi resource case, to show the capability of the agent to work within a multi-agent scenario. In both cases, 10 different production scenarios are used during the training phase. Each production scenario has a random number of jobs (5 to 8) with different processing times and different due-dates. Experiments were based on a random generation of problems with different problem characteristics (number of jobs, loads, tightness of jobs, due-date-range, ...). In the single resource case, each job has one operation, whereas in the multi-resource case, each job has a random number of basic operations. Each operation has a random duration and must be processed on a certain resource. In the multi-resource case we also allow circles - i.e. a job may have to visit one resource multiple times - which constitutes an additional difficulty for conventional solution algorithms. The mean process duration and the mean due-date were chosen such that 'interesting' scenarios are created, i.e. that arbitrary policies are not likely to produce acceptable solutions. As mentioned above, the production objective considered here is to reduce the overall tardiness of all the jobs. To test the generalization ability, 10 test scenarios were generated, which vary from the ones considered in the learning phase.

For learning, we used a multi-layer perceptron with up to 6 inputs, 10 hidden neurons and one output neuron. The learning rate was set to 0.1 (since it was not our goal to optimize the learning speed, not much effort was done to find an optimal parameter here). During learning, the jobs are selected randomly (exploration factor = 1). The performance is reported in terms of the average tardiness of the jobs when acting greedily with respect to the current value function.

3.3 The one resource case

Table 1 shows the performance of some typical heuristic dispatching rules. The LPT-policy chooses the job with the longest processing time first, the minimum-slack (MS)-policy chooses the job with the minimum time between the expected termination and the due-date, and the EDD-policy chooses the job with the most urgent due-date. The average tardiness per job varies considerably on the 10 training scenarios. LPT performs worst with an average tardiness of 18.9, even worse than a random policy (12.9). MS works considerably better showing an average tardiness of 7.5, and EDD perfoms best with an average tardiness of 5.7.

	LPT	Random	MS	EDD
Training Set	18.9	12.9	7.5	5.7

Table 1: Average tardiness on the training set for different heuristic dispatching policies

To test the learning capability of the neural dispatching agent, several combinations of input features were tested (the feature numbers in table 2 correspond to the numbering in section 2.4). In general, with a sensitive choice of input features the performance of the learning system did improve considerably with the number of production runs (remember that the average tardiness of a random policy is 12.9). Not surprisingly, the final performance depends crucially on the provided input information. We observed that the performance of the system with features that are also considered by the EDD-rule has the same final performance as the EDD-rule (column 1). This means, that the learning system was able to extract this rule automatically out of the experience it made. Analogously, the same was true for the MS-rule (column 2). When we gave the combination of both features to the learning system, it was able to find a new policy, that is better than both EDD and MS (column 3). Actually, this is the effect we are expecting the learning agent to exploit: considering a combination of features that are of different importance in different situations and acquiring a new (probably very complicated) policy based on the input. However, adding more features not always improves the performance here (column 4).

features	1,2,3,4	1,2,3,4,5	1,2,3,4,6	1,2,3,4,5,6
Training set	5.7	7.5	**4.6**	5.0

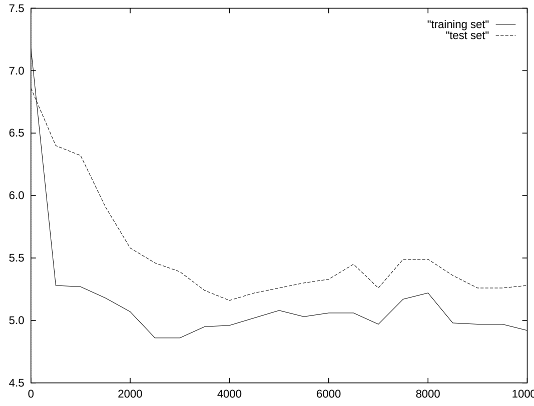
Table 2: Learning agent: Average tardiness on the training set for different input feature combinations

Figure 1 gives an impression of the learning process. The bold line shows the performance on the training set. After about 500 production runs, the system has a performance of 5.3 and thus already beats the EDD-policy (5.7). In course of learning, the performance is further improved.

3.4 Generalization ability

Besides the principle learning and optimization capabilities, one major effect we expect to observe is the generalization ability of the learning system. To test it, the trained neural agent is applied to situations not included in the training set. The results are shown in table 3.

Figure 1: Improvement of the performance of the learning system with an increasing number of production runs. The average performance of a random policy was 12.9, and the best heuristic policy, EDD, achieved an average tardiness of 5.5. The learning agent beats EDD after only 500 production runs (bold line).



	LPT	random	MS	SPT	EDD	Neural
Test Set	17.4	12.0	9.0	6.5	6.6	**5.2**

Table 3: Average tardiness for different dispatching policies on the test set (test for generalization)

Again, the neural agent shows the best performance 5.2 and beats the best heuristic rule considerably, which is the SPT-rule here with 6.5. This shows, that the learning agent is not only able to optimize its performance on a certain set of training cases, but also is able to generalize this knowledge to new, previously unknown cases *without retraining*. It may also be derived, that the selected features fulfill the requirement of problem independency.

3.5 The multi-resource case

In the multi-resource benchmark we consider a plant consisting of 3 resources. Here we examine the ability of one learning agent to adapt to the behaviour of a complex process. This behaviour is determined by the job profile and the structure of the plant, but in contrast to the previous scenario, also dispatching policies for the other resources play an important role. While we are examining the case of *one* learning agent and the other policies being fixed, in future experiments we will examine situations of multiple agents learning simultaneously.

In the training scenario, for example, when two fixed agents are acting according to the FIFO (First-in-First-Out) rule, than the performance of the third agent can be improved from 12.2 (the case of also acting according to the FIFO

	Training set			Test set		
	FIFO	EDD	MS	FIFO	EDD	MS
FIFO	12.2	-	-	10.9	-	-
EDD	-	2.65	-	-	3.17	-
MS	-	-	2.65	-	-	3.08
neural	**7.5**	**2.6**	**2.47**	**8.19**	**2.7**	**2.89**

Table 4: Average tardiness for different dispatching policies in the multi-resource case. The horizontal row denotes the fixed policies of resource 1 and 2, the vertical row compares the policy of resource 3 for a fixed policy and a neural learning policy

principle) to 7.5 (when a learning policy is applied). In order to get an optimal behaviour, the learning agent has to consider the future processing policy of a candidate job, too. As an additional difficulty, since circles may occur during the lifetime of a job, the current decision now also determines the future development of the candidate set. As can be seen in table 4, the learning agent is capable to deal with the described difficulties. In all training cases (left side) and all test cases (right side) the learning policy outperformed the fixed policy. The agent has autonomously acquired a local policy based on few relevant decision features, that is able to perform well in a complex environment with complex dynamics.

4 Conclusions

The paper describes a neural network based local learning approach to job shop scheduling problems. It is based on local learning agents, associated to a resource. The agent has a restricted view on the complete factory's state, representing the most important features that are needed for the local decision. The learning rule relates a local value function with costs depending on the overall performance of the global plant. Doing so, the acquired local decision policy is coordinated with the global optimization goal. The experiments on a one-resource and a three-resource production plant show the capability to learn local policies to optimize global behaviour from experience. Furthermore, the agent's policy can be generalized to unknown situations without retraining. Therefore, the learned policies are more tailored to the actual task than comparable heuristic dispatching rules, but still are general enough to be valid in a wide range of untrained situations. In case of major changes in the organizational structure the proposed learning architecture allows an easy reconfiguration of the reactive scheduling policy.

5 Acknowledgments

A major part of this work was carried out during a research visit at the Robotics Institute at Carnegie Mellon University, Pittsburgh, PA. The authors gratefully thank Stephen Smith, Andrew Moore and Jeff Schneider for valuable discussions. Special thanks to Dieter Spath, Hartmut Weule, Juergen Schmidt and Wolfram Menzel for giving us the opportunity and the support for this stay.

References

[Barto and Crites, 1996] A. G. Barto and R. H. Crites. Improving elevator performance using reinforcement learning. In M. E. Hasselmo D. S. Touretzky, M. C. Mozer, editor, *Advances in Neural Information Processing Systems 8*. MIT Press, 1996.

[Boyan and Littman, 1994] Justin Boyan and Michael Littman. Packet routing in dynamically changing networks - a reinforcement learning approach. In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, 1994.

[Brauer and Weiss, 1998] Wilfried Brauer and Gerhard Weiss. Multi-machine scheduling - a multi-agent learning approach. In *Proceedings of the 3rd International Conference on Multi-Agent Systems*, pages 42–48, 1998.

[Dietterich and Zhang, 1995] T. Dietterich and W. Zhang. A reinforcement-learning approach to job-shop scheduling. In *Proceedings of the 14.th International Joint Coference on Artificial Intelligence*, 1995.

[Mahadevan and Theocharous, 1998] S. Mahadevan and G. Theocharous. Optimization production manufacturing using reinforcement learning. In *Proceedings of the Eleventh International FLAIRS Conference*, pages 372 – 377. AAAI Press, 1998.

[Pinedo, 1994] M. Pinedo. *Introduction to Scheduling - Algorithms.* 1994.

[Schneider *et al.*, 1998] Jeff Schneider, Justin Boyan, and Andrew Moore. Value function based production scheduling. In *International Conference on Machine Learning*, 1998.