Solving a job-shop scheduling problem by an adaptive algorithm based on learning

Yuri N. Sotskov*, Omid Gholami, Frank Werner*****

** United Institute of Informatics Problems, Minsk, Belarus*
(Tel: 217-375-2842120; e-mail: sotskov@newman.bas-net.by).
***Islamic Azad University - Mahmudabad Branch, Iran*
(Tel: 217-375-2842125; e-mail: gholami@iaumah.ac.ir).
****Faculty of Mathematics, Otto-von-Guericke-University, Magdeburg, Germany*
(Tel: 49-391-67-12025; e-mail: frank.werner@ovgu.de)

Abstract: A learning stage of scheduling tends to produce knowledge about a benchmark of priority dispatching rules which allows a scheduler to improve the solution quality for a set of similar job-shop problems. Once trained on the sample job-shop problems (usually with small sizes), the adaptive algorithm solves a similar job-shop problem (with a moderate size or a large size) better than heuristics used as a benchmark at the learning stage of scheduling. Our adaptive algorithm does not guarantee to perform as an exact algorithm or better than a more sophisticated heuristic algorithm (like e.g. the shifting bottleneck one) which need a large running time. For an adaptive algorithm with a learning stage, the job-shop scheduling problem is modeled via a weighted mixed (disjunctive) graph with the conflict resolution strategy used for finding an appropriate schedule.

Keywords: Job-shop scheduling; Priority dispatching rules; Learning stage.

1. INTRODUCTION

Priority dispatching rules have been studied in the OR literature for several decades since they are widely used for different scheduling problems like the job-shop problem arising in real world: Haupt (1989); Muth and Thompson (1963); Panwalkar and Iskander (1977); Tanaev et al. (1994). However, the general conclusion from many years of academic and practical research is that no priority dispatching rule performs globally better than other ones tested for a wide class of scheduling problems which are NP-hard: Geiger et al. (2006); Gholami et al. (2012); Mouelhi-Chibani and Pierreval (2010); Shakhlevich et al. (1996). A priority dispatching rule may provide a good solution for a concrete scheduling problem but applied to another NP-hard problem, it may provide a bad solution the quality of which is far from that of an optimal schedule. To develop an efficient priority dispatching rule for a class of job-shop problems, which are binary NP-hard even for three jobs and three machines, Sotskov and Shakhlevich (1995), Brucker et al. (2007), takes a lot of research involving the implementation of different priority dispatching rules in a simulation. Therefore, several researchers developed tools to discover effective priority dispatching rules automatically: Abdolzadeh and Rashidi (2009); Gabel and Riedmiller (2007); Geiger et al. (2006); Dorndorf and Pesch (1995); Li and Shi (1994); Shakhlevich et al. (1996). Next, we review some scheduling approaches using a learning stage. A scheduling problem is denoted by a triplet $\alpha \mid \beta \mid \gamma$, as Lawler et al. (1993) have proposed.

A tabu search algorithm is a local search applied to various optimization problems, Glover (1989). Tabu search adopts local search with a memory implemented as a tabu list of moves which have been made in the past of the search, and which are forbidden (tabu) for certain iterations of the algorithm. A tabu move may be accepted if the solution obtained by the application of the move being better than the best solution previously obtained.

Gabel and Riedmiller (2007) adopted an alternative view on scheduling problems by modelling them as multi-agent reinforcement learning problems. They interpreted a job-shop problem as a sequential decision process and attach to each machine an adaptive agent that makes its job dispatching decisions independently of the other agents and improves its behaviour by trial and error employing a reinforcement learning stage. A multi-agent algorithm was developed, which combines data-efficient batch-mode reinforcement learning, neural network-based value function approximation and the use of an optimistic inter-agent coordination scheme.

Mouelhi-Chibani and Pierreval (2010) proposed an approach based on a neural network to select the most suited priority dispatching rule each time a machine becomes available. Contrary to the most learning approaches to select scheduling heuristics, no training set is needed. The parameters of the neural network are determined through a simulation. For the flow-shop problem, it was shown that the neural network can select the priority dispatching rule dynamically. Once trained offline, the resulting neural network can be used online in connection with a flexible manufacturing system.

Abdolzadeh and Rashidi (2009) developed an approach of cellular learning automata for the job-shop problem. In their approach, there were two types of action sets. These actions were generated in order to transfer cells into the best states by changing the position of operations of some jobs.

Geiger et al. (2006) proposed a genetic programming approach, which is capable of automatically discovering priority dispatching rules for several single machine scheduling problems, namely, for the problems $1||\sum C_i$ and $1||L_{\max}$ which are polynomially solvable and for the problems $1|r_i|\sum C_i$, $1|r_i|L_{\max}$, $1|r_i|\sum T_i$, (and problem $1||\sum T_i$) which are unary NP-hard (binary NP-hard). The mechanism for discovery a composite dispatching rule using a benchmark of the priority dispatching rules was based on a genetic algorithm. In contrast to a usual implementation of an evolutionary-based search for creating new schedules, a genetic algorithm was used for creating a new composite dispatching rule, Geiger et al. (2006). Shakhlevich et al. (1996) showed how to generate a composite dispatching rule using a learning stage. To solve a problem $J||C_{\max}$ heuristically, Dorndorf and Pesch (1995) used a genetic algorithm served as a strategy to guide the design of suitable sequences of the priority dispatching rules. They considered both sequences of priority dispatching rules for job scheduling and job sequences on one bottleneck machine in the sense of the shifting bottleneck algorithm, Adams et al. (1988).

In Section 4, we compare our adaptive algorithm with the shifting bottleneck algorithm, as one of the most efficient heuristics developed for the problem $J||C_{\max}$. This algorithm finds the best schedule for that single machine, which is currently a bottleneck, and calculates the throughput time for each job. The minimum lateness for each machine calculated by finding the paths through the machines that reduces the maximum lateness $L_{\max} = \max\{C_i - d_i : J_i \in J\}$ observed for all the jobs on the bottleneck machine. Hereafter, d_i and C_i denote the due date and the completion time of the job $J_i \in J$ respectively, where $J = \{J_1, J_2, ..., J_n\}$ is the set of jobs to be processed. The operations on the bottleneck machine are scheduled due to solution of the NP-hard problem $1|r_i, prec|L_{\max}$ of minimizing maximal lateness. Here r_i denotes the release time of the job J_i. After defining additional precedence constraints, the analysis for the remaining machines is continued. The above process is repeated until either all machines have been accounted for or the maximum lateness equals zero for all the remaining machines. As computational experiments showed, the shifting bottleneck algorithm runs in a reasonable CPU-time if the number m of machines is not much greater than the number n of jobs. Due to this reason, our adaptive algorithm was also compared with the algorithm Ordinal-ECT developed by Gholami et al. (2012). The latter algorithm runs faster than the shifting bottleneck one in the case of $m > n$ while the quality of the solution obtained by the algorithm Ordinal-ECT is usually close to that obtained by the shifting bottleneck algorithm.

2. A MIXED GRAPH MODEL FOR THE JOB-SHOP

There are n jobs $J = \{J_1, J_2, ..., J_n\}$, which need to be processed on m different machines $M = \{M_1, M_2, ..., M_m\}$. The machine (technological) route $O_i = (O_{i1}, O_{i2}, ..., O_{in_i})$ of each job $J_i \in J$ through the machines M is fixed. The machine routes O_i may be different for different jobs $J_i \in J$. The time p_{ij} for processing operation O_{ij} of job $J_i \in J$ on the corresponding machine $M_v \in M$ is known. The objective of the problem $J||C_{\max}$ is to minimize the makespan $C_{\max} = \max\{C_i : J_i \in J\}$, which denotes the minimization of the time when all the jobs J have been processed.

The job-shop problem $J||C_{\max}$ may be described using a weighted mixed (or equivalently, disjunctive) graph $G = (Q, A, E)$, which is an appropriate model for constructing various exact and heuristic algorithms: Adams et al. (1988), Gholami et al. (2012), Shakhlevich et al. (1996), Lawler et al. (1993), Tanaev et al. (1994). In the mixed graph G, the vertex set Q is the set of operations: $Q = \{O, O_{1,1}, O_{1,2}, ..., O_{1n_1}, ..., O_{n1}, O_{n2}, ..., O_{nn_n} O_*\}$ including a dummy source operation O preceding all the other vertices in the digraph $(Q, A, \emptyset)$ and a dummy sink operation O_*. Each vertex of the set $Q \setminus \{O_*\}$ proceeds to vertex O_* in the digraph $(Q, A, \emptyset)$.

The arc set A defines all precedence constraints, and the edge set E defines all machine constraints: at any time a machine from the set M can process at most one job from the set J. A weight p_{ij} is prescribed to operation $O_{ij} \in Q$ of each job $J_i \in J$ and each stage $j \in \{1, 2, ..., n_i\}$. Hereafter, p_{ij} is equal to the time needed to process the operation O_{ij} on the corresponding machine $M_v \in M$. There exists a one-to-one correspondence between all semi-active schedules and all circuit-free digraphs Δ generated by the mixed graph G via orienting all edges from the set E. A schedule is called semi-active if no operation $O_{ij} \in Q$ can start earlier without delaying the processing of some other operation from the set Q or without altering the processing sequence of the operations on any of the machines M. For any regular criterion, at least one optimal schedule is semi-active. An optimal semi-active schedule is defined by an optimal digraph $G_s = (Q, A \cup A_s, \emptyset)$ from the set Δ, Tanaev et al. (1994). One of the general scheduling approaches, based on the mixed graph model, is the conflict resolution strategy, which may be used to develop either exact or heuristic scheduling algorithms. At an elementary step of such an algorithm, the conflict resolution strategy means to deal with one conflict edge of the mixed graph G and to decide which of the two orientations of the conflict edge has to be chosen for inclusion into the desired digraph $G_s = (Q, A \cup A_s, \emptyset)$. An edge $[O_{ij}, O_{uv}] \in E$ is called a conflict edge if both of its orientations (O_{ij}, O_{uv}) and (O_{uv}, O_{ij}) lead to an increase either of the starting time s_{uv} of the operation O_{uv} or of the starting time s_{ij} of the operation O_{ij}.

3. AN ADAPTIVE SCHEDULING ALGORITHM

An adaptive algorithm schedules the jobs in a job-shop using the knowledge on a benchmark of the priority dispatching rules tested at a specific learning stage.

At a learning stage via solving small or moderate job-shop problems optimally, the adaptive scheduler would be trained with several characteristics based on priority dispatching rules while constructing an optimal schedule. To this end, the characteristics of the operations from the conflict edges are calculated and stored in a learning database along with the decision made in the optimal scheduling.

At the examination stage, the learning database is used for scheduling the jobs appropriately in a job-shop with a moderate size or with a large size by an adaptive scheduler as a pattern. Our adaptive scheduler includes several modules and two databases. The general scheme of the adaptive scheduler can be seen in Fig. 1. In Subsections 3.1 – 3.4, the main modules of the adaptive scheduler are discussed in more detail as the page limit allows.

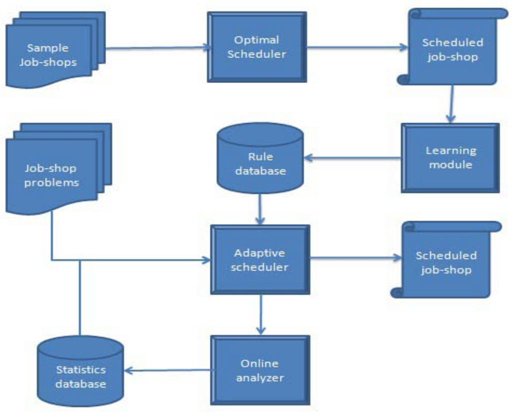

Figure 1: Scheme of the adaptive scheduler

3.1. The learning stage

For the learning stage, an optimal scheduler for the job-shop problem, like a branch-and-bound method, is needed to solve exactly instances with sufficiently restricted sizes. Then the data achieved from the learning stage may be used at the examination stage. A few small samples solved at the learning stage reduce the accuracy of the learning data, while a lot of samples or big samples solved increase the running time needed to find their optimal schedules at the learning stage. At the learning stage, the information about a successful orientation of the conflict edges is stored in the database. The learning table is analogous to those used in the theory of pattern recognition and it describes which orientation of a conflict edge is preferable while an optimal digraph (and an optimal schedule) is constructed.

The learning database is filled by the data as it is depicted in Table 1, where the learning data are filled in the first column, in the columns $X_1, X_2, ..., X_r$, and in the last column. The last column of Table 1 shows the decision made by an optimal scheduler to resolve a conflict edge $[O_{ij}^k, O_{uv}^k]$ from the set E of the mixed graph $G = (Q, A, E)$. Hereafter, a superscript k in the notation of the edge $[O_{ij}^k, O_{uv}^k]$ is used to distinguish different edges either for the same mixed graph $G = (Q, A, E)$ or for different mixed graphs modelling job-shop problems $J \| C_{\max}$ used at the learning stage. For simplicity of the notation, all conflict edges have same subscripts *ij* and *uv* in the first column of Table 1.

If in the optimal digraph $G_s = (Q, A \cup A_s, \varnothing)$, a conflict edge $[O_{ij}^k, O_{uv}^k] \in E$ was substituted by the arc $(O_{ij}^k, O_{uv}^k) \in A_s$, then we set $\Omega^k = \Omega_1$. If in the optimal digraph $G_s = (Q, A \cup A_s, \varnothing)$, a conflict edge $[O_{ij}^k, O_{uv}^k] \in E$ was substituted by the arc $(O_{uv}, O_{ij}) \in A_s$, then we set $\Omega^k = \Omega_2$.

Table 1: Conflict resolutions in optimal schedules

Conflict edge	X_1	X_2	**…**	X_r	**Ω class**
$[O_{ij}^1, O_{uv}^1]$	g_1^1	g_2^1	…	g_r^1	Ω^1
$[O_{ij}^2, O_{uv}^2]$	g_1^2	g_2^2	…	g_r^2	Ω^2
⋮	⋮	⋮	⋮	⋮	
$[O_{ij}^w, O_{uv}^w]$	g_1^w	g_2^w	…	g_r^w	Ω^w
$[O_{ij}, O_{uv}]$	g_1	g_2	…	g_r	?

For each conflict edge $[O_{ij}^k, O_{uv}^k], k \in \{1,2,...,w\}$, which was treated while branching in a branch-and-bound method, the characteristics corresponding to the priorities of the operations O_{ij}^k and O_{uv}^k on the corresponding machine $M_v \in M$ have to be calculated and stored in the columns $X_1, X_2, ..., X_r$ of Table 1. To be more precise: for each priority dispatching rule from the database used in the learning stage, a priority π_{ij}^k of operation O_{ij}^k and priority π_{uv}^k of operation O_{uv}^k are calculated. Respecting the priority dispatching rule, the operation with a larger priority has to be processed on the corresponding machine $M_v \in M$ before processing the operation with a smaller priority. The characteristic X_t of the conflict edge $[O_{ij}^k, O_{uv}^k] \in E$ corresponding to the priority dispatching rule is defined as the relative difference of the priorities π_{ij}^k and π_{uv}^k of the operations O_{ij}^k and O_{uv}^k as follows:

$$g_t^k = \frac{\pi_{ij}^k - \pi_{uv}^k}{\max\{\pi_{ij}^k, \pi_{uv}^k\}}.$$

The sign of the value g_t^k shows which of the operations O_{ij}^k or O_{uv}^k has priority to be processed first on the machine $M_v \in M$. The absolute value of g_t^k shows how much the superiority of the operation with the larger priority is?

There are a lot of priority dispatching rules that are used in a variety of heuristic algorithms for scheduling the jobs $J_i \in J$ in a job-shop: Haupt (1989), Muth and Thompson (1963), Panwalkar and Iskander (1977). Some characteristics of the priority dispatching rules are gathered before scheduling while some other characteristics are post scheduling ones. As an example of a priority dispatching rule, let us consider the Earliest Completion Time rule (ECT–rule, for short) that recommends to process first the operation with the earlier completion time. Let the earliest completion time for operation O_{ij}^k (for operation O_{uv}^k) be equal to 90 (to 73, respectively). Then one can calculate the value of the corresponding characteristic X_t as follows:

$$g_t^k = \frac{\pi_{ij}^k - \pi_{uv}^k}{\max\{\pi_{ij}^k, \pi_{uv}^k\}} = \frac{90-73}{90} = 0.18 .$$

A positive value g_t^k indicates that operation O_{uv}^k has to be processed before operation O_{ij}^k respecting the ECT–rule.

3.2. The examination stage

At the examination stage, a job-shop problem $J \| C_{\max}$ with a moderate or with a large size has to be solved using a natural principle of precedence: it is reasonable in a new conflict situation to adopt a decision which has lead to success (i.e., to an optimal schedule in our case) at the learning stage in a similar conflict situation.

Let the weighted mixed graph $G = (Q, A, E)$ model a job-shop problem $J \| C_{\max}$ to be solved at the examination stage by an adaptive scheduler. In the scheduling process for each conflict edge $[O_{ij}, O_{uv}] \in E$ of the mixed graph G which was met by the adaptive scheduler, the characteristic vector $(g_1, g_2, ..., g_r)$ has to be calculated and then compared with the characteristic vectors $(g_1^k, g_2^k, ..., g_r^k)$, $k \in \{1,2,...,w\}$, of the conflict edges stored in Table 1 during the learning stage.

Let the characteristic vector $(g_1^e, g_2^e, ..., g_r^e)$ (here the index $e \in \{1,2,...,w\}$ is fixed) be the closest to the vector $(g_1, g_2, ..., g_r)$ among all vectors $(g_1^k, g_2^k, ..., g_r^k)$ (here the index $k \in \{1,2,...,w\}$ varies) presented in Table 1. Then, to resolve a conflict edge $[O_{ij}, O_{uv}] \in E$ in the mixed graph $G = (Q, A, E)$, an adaptive scheduler uses the same decision as in the class Ω^e stored in Table 1.

It should be noted that sometimes the characteristic vectors appear so close to each other, that the heuristic fundamental of the algorithm makes a mistake implying a circuit appearance in a digraph $G_s = (Q, A \cup A_s, \varnothing)$ constructed by the adaptive scheduler. A circuit in the digraph $G_s = (Q, A \cup A_s, \varnothing)$ means a deadlock in machine binding, and to prevent from this phenomenon, a circuit test procedure has been applied. For example, if the adaptive scheduler decides to introduce the arc (O_{ij}, O_{uv}) in the digraph generated by the mixed graph G, a circuit tester examines while a path from the vertex O_{uv} to the vertex O_{ij} exists in the already constructed subgraph of the digraph $G_s = (Q, A \cup A_s, \varnothing)$. If such a path exists, adding the arc (O_{ij}, O_{uv}) to the constructed subgraph will generate a circuit. Therefore, the symmetric arc (O_{uv}, O_{ij}) has to be added to this subgraph of the digraph $G_s = (Q, A \cup A_s, \varnothing)$.

3.3. Procedures for monitoring variable parameters

It should be noted that each time a scheduler wants to make a decision, different job characteristics like the operation starting time, completion time, due date, processing time and some other parameters are needed for the right decision. Some of these data like the operation processing time are unchangeable, while others like the operations completion time may be changed due to adding a new arc in the digraph $G_s = (Q, A \cup A_s, \varnothing)$.

Thus, at each such time point, it is necessary to refresh the characteristics of the jobs which are in conflict. It is clear that such recalculations are mass and therefore, time-consuming. To get rid of this computation problem, we developed special procedures for monitoring variable parameters. These procedures are used both at the learning and at the examination stages of the adaptive algorithm.

The main duty of these procedures is to control on-line the digraph $G_s = (Q, A \cup A_s, \varnothing)$ while it is constructing. At every time point when an arc is added to the digraph, the side effect of adding this arc is analyzed. If a new arc has an influence on the corresponding parameters of the subsequent jobs, then the data about those jobs have to be updated. Procedures for monitoring variable parameters prevent a scheduler to regenerate the whole set of parameters each time when they are needed for decision-making.

As an example, the following procedure shows how to update the starting times of the operations when a scheduler decided to add an arc (O_{ij}, O_{uv}) in the digraph $G_s = (Q, A \cup A_s, \varnothing)$.

```
IF $s_{uv} < s_{ij} + p_{ij}$ THEN { $s_{uv} = s_{ij} + p_{ij}$ ; $Buffer := Add(O_{uv})$ }
WHILE NOT ( $empty(Buffer)$ ) DO
        { $node := remove(Buffer)$
        FOR each subsequent of (node) as sub DO
                IF ( $s_{sub} < s_{node} + p_{node}$ ) THEN
                { $s_{sub} = s_{node} + p_{node}$ $Buffer := Add(sub)$ }}
```

Other parameters needed for a scheduler to make the right decision for conflict resolution may be monitored similarly.

3.4. Three strategies for considering a set of conflict edges

One challenge in an adaptive algorithm is how to find a conflict edge $[O_{ij}, O_{uv}] \in E$ and compare the characteristics of the jobs $J_i \in J$ and $J_u \in J$, which are waiting for processing on the same machine. It is clear that the order, in which the conflict edges will be resolved, will influence the quality of the objective function value in the constructed schedule. There are several strategies for scanning the job requests for processing. Some algorithms, like the shifting bottleneck one, determine which machine is currently the bottleneck. This is realized via considering the times p_{ij} needed for processing the jobs $J_i \in J$ on the machines, which are involved in the routes O_i, the release times of the jobs $J_i \in J$ on these machines, and the due dates of the jobs on these machines. As our experiments showed, if the machine number is considerably larger than the job number, such an algorithm is time-consuming.

An algorithm of another type looks for a critical job (i.e., a job with the largest total processing time on the machines involved in the job route) and tries to process first the operations of the critical job. Then the algorithm tries to process next the operations of the second critical job, and so on. Another algorithm, which could also be considered, tries to process first the operations O_{i1} of jobs $J_i \in J$, then the second operations O_{i2} of the jobs $J_i \in J$, then the third operations O_{i3} of the jobs $J_i \in J$, and so on until the last operations O_{in_i} of the jobs $J_i \in J$ are processed.

We have treated three strategies of scanning the conflict edges in order to choose the most effective strategy to schedule the jobs in the adaptive algorithm. The first strategy uses the critical path method and sorts the jobs $J_i \in J$ by decreasing the total job processing times and starts with processing a job having the largest total processing time. In Fig. 2, this version of the adaptive algorithm is called Max-PT (Maximum Processing Time first). The second strategy makes the same as the first one but sorts the jobs oppositely: in increasing order of the total job processing times. In Fig. 2, this adaptive algorithm is called Min-PT (Minimum Processing Time first). The third strategy processes first the operations O_{i1} of the jobs $J_i \in J$, then the operations O_{i2} of the jobs $J_i \in J$, and so on. This adaptive algorithm is called Ordinal in Fig. 2. The evaluation of a wide set of randomly generated job-shop problems by these three strategies shows that the Ordinal algorithm performs better with respect to the objective function $C_{\max}$ than the two other strategies. So, the Ordinal strategy was used in the adaptive algorithm to resolve the set of conflict edges.

4. COMPUTATIONAL RESULTS

For the computational experiments, we used a laptop computer: Intel®, core^{TM} 2 Duo, CPU T6400, 2.00 GHz and 2GB Internal Memory, Windows 7, Ultimate 32 bit. Only one instance of a job-shop problem with 6 jobs, 6 machines and the equality $n_i = 6$ for all jobs $J_i \in J$, given by Muth and Thompson (1963) was used at the learning stage. This example is named as MT-6. The benchmark of the priority dispatching rules includes the following four rules: Earliest Due Date rule (EDD–rule), First Come First Served rule (FCFS–rule), Longest Processing Time rule (LPT–rule) and Shortest Processing Time rule (SPT–rule).

In Table 2, the effectiveness of the adaptive algorithm is compared with the Ordinal-ECT algorithm; Gholami et al. (2012), the Shifting bottleneck algorithm, Adams et al. (1988), and with four heuristic algorithms based on a single priority dispatching rule from the benchmark used at the learning stage.

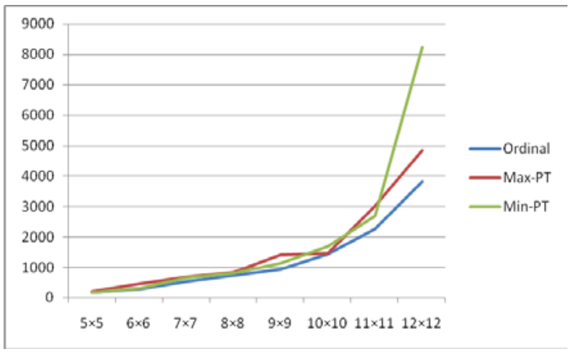

Figure 2: Comparison of the $C_{\max}$ values obtained by the three strategies to process the operations $O_{ij} \in Q$

Table 2 shows the values of $C_{\max}$ obtained for the benchmark problems $J \| C_{\max}$, namely: for problem MT-6; for the famous problem MT-10 with 10 jobs and 10 machines, given by Muth and Thompson (1963); for another problem with 10 jobs and 10 machines; for two problems with 10 jobs and 5 machines, and for a problem with 18 jobs and 5 machines. The CPU-times used for running the adaptive algorithm are given in the last two figures.

In Fig. 3, the number of jobs is fixed, $n = 10$, and the number of machines is increased from 10 to 40. In Fig. 4, the number of machines is fixed, $m = 10$, and the number of jobs is increased from 10 to 40. Thus, one can see that the adaptive algorithm is both effective and efficient.

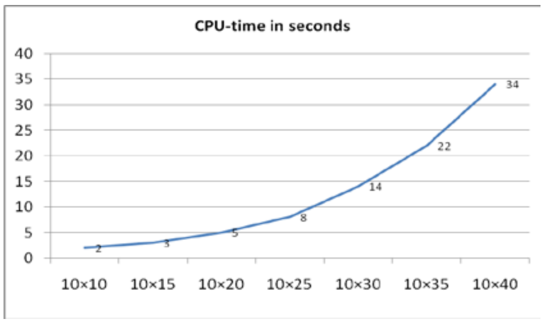

Figure 3: Average CPU-times used by the adaptive algorithm for solving a set of problems with a fixed number of jobs

Our experience showed that sample job-shop problems having different patterns of operation topology and parameters (e.g., the range of the job processing times) have a different impact on the training of the adaptive algorithm at the learning stage. As only the optimal MT-6 solution was used for learning, the adaptive algorithm could not show its ability in the presented experiments. To get better objective function values, it is desirable to train the adaptive algorithm on different optimal solutions, which are closer to the instances that have to be solved at the examination stage. Furthermore, it is desirable to have a faster exact algorithm for the job-shop problems when it is necessary to solve large job-shop problems very well. The exact algorithm may be used to solve a part of a large problem, e.g., a sub-problem with size 6×6 or 7×7, and the obtained optimal schedules may be used at the learning stage. It is clear that, if the adaptive algorithm is trained on the original problem, it can produce better results.

Table 2: Makespan values for six benchmark instances $J \| C_{\max}$ calculated by seven heuristic algorithms

Benchmark problems $J \| C_{\max}$	**Ordinal-ECT algorithm**	**Shifting bottleneck**	**EDD-rule**	**FCFS-rule**	**LPT-rule**	**SPT-rule**	**Adaptive algorithm**
MT-6 (6×6)	59	58	63	65	67	73	58
MT-10 (10×10)	1252	1094	1246	1184	1168	1338	1167
Job-shop-10 (10×10)	82	94	122	87	86	118	86
Job-shop-18 (18×5)	1419	1220	1263	1462	1393	1451	1370
SGW-10-1 (10×5)	662	564	629	638	627	762	564
SGW-10-2 (10×5)	714	620	673	650	638	807	620

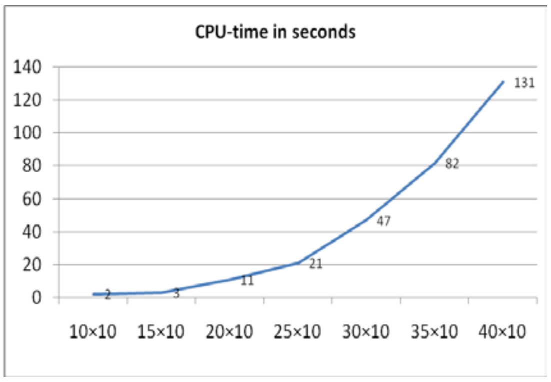

Figure 4: Average CPU-time used by the adaptive algorithm for solving a set of problems with a fixed number of machines

5. CONCLUSION

An adaptive scheduling algorithm was developed to solve the unary NP-hard problem $J \| C_{\max}$ with a large size. Due to the learning stage, it could schedule the jobs with a good value of the objective function $C_{\max}$ (Table 2) in reasonable CPU-time (Fig. 3 and Fig. 4). Using a pattern recognition technique to find the best answer for conflict resolution helps the adaptive scheduler to obtain schedules with good values of the objective function. By tuning the adaptive algorithm on different optimal solutions of such instances, which are close to those that have to be solved, the quality of the results will be improved without any side effect on the running time of the adaptive algorithm at the examination stage.

REFERENCES

J. Adams, E. Balas, D. Zawack, The shifting bottleneck procedure for jobshop scheduling. *Management Science*, 34(3): 391–401, 1988.

M. Abdolzadeh, H. Rashidi, An approach of cellular learning automata to job shop scheduling problem. *International Journal of Simulation: Systems, Science and Technology*, 11(2):56–64, 2010.

P. Brucker, Y.N. Sotskov, F. Werner, Complexity of shop-scheduling problems with fixed number of jobs: A survey. *Mathematical Methods of Operations Research*, 65(3): 461–481, 2007.

U. Dorndorf, E. Pesch, Evoluation based learning in a job shop scheduling environment. *Computers & Operations Research*, 22(1): 25–40, 1995.

T. Gabel, M. Riedmiller, Adaptive reactive job-shop scheduling with learning agents. *International Journal of Information Technology and Intelligent Computing*, IEEE Press, 2(4), 2007.

C.D. Geiger, R. Uzsoy, H. Aytug, Rapid modelling and discovery of priority dispatching rules: an autonomous learning approach. *Journal of Scheduling*, 9:7–34, 2006.

O. Gholami, Y.N. Sotskov, F. Werner, Job-shop problems with objectives appropriate to train scheduling in a single-track railway. *SIMULTECH 2012 - Proceedings of 2nd International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, 425–430, 2012.

F. Glover, Tabu search – part 1. *ORSA Journal on Computing*, 1(2):190–206, 1989.

R. Haupt, A survey of priority rule-base scheduling. *OR Spectrum*, 11(1): 3–16, 1989.

E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys, Sequencing and Scheduling: Algorithms and Complexity. In: *Handbooks in Operations Research and Management Science, Volume 4: Logistic of Production and Inventory*, Edited by Graves, S.C., Rinnooy Kan, A.H.G., Zipkin, P., North-Holland, 445–522, 1993.

D.-C. Li, I.-S. Shi, Using unsupervised learning technologies to induce scheduling knowledge for FMSs. *International Journal of Production Research*, 32(9): 2187–2199, 1994.

W. Mouelhi-Chibani, H. Pierreval, Training a neural network to select dispatching rules in real time. *Computers & Industrial Engineering*, 58:249–256, 2010.

J.F. Muth, G.L. Thompson, Industrial Scheduling. *Prentice-Hall*, Englewood Cliffs, N.J., 1963.

S.S. Panwalkar and W. Iskander, A survey of scheduling rules, *Operations Research*, 25(1):45–61, 1977.

N.V. Shakhlevich, Y.N. Sotskov, F. Werner, Adaptive scheduling algorithm based on mixed graph model. *IEE Proceedings: Control Theory and Applications*, 143(1): 9–16, 1996.

Y.N. Sotskov, N.V. Shakhlevich, NP-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59: 237–266, 1995.

V.S. Tanaev, Yu.N. Sotskov, V.A. Strusevich, Scheduling Theory: Multi-Stage Systems. *Kluwer Academic Publishers*, Dordrecht, Netherlands, 1994.