A reinforcement learning algorithm to improve scheduling search heuristics with the SVM

Kai Gersmann and Barbara Hammer
Research Group LNM
Department of Mathematics/Computer Science
University of Osnabrück
49069 Osnabrück, Germany
E-mail: {kai,hammer}@informatik.uni-osnabrueck.de

***Abstract*— The Regret-Based Biased Random Sampling Scheme (RBRS) is a simple but powerful priority-rule based method to solve the Resource Constrained Project Scheduling Problem (RCPSP), a well-known NP-hard benchmark problem. We present a generic machine learning method to improve results of RBRS. The rout-algorithm of reinforcement learning is combined with the support vector machine (SVM) to learn an appropriate value function which guides the search strategy given by RBRS. The specific properties of the SVM allow to reduce the size of the training set and show improved results even after a short period of training as demonstrated for benchmark instances of the RCPSP.**

I. INTRODUCTION

Real life scheduling instances usually possess a large amount of problem dependent structure which is not tackled by formal descriptions of the respective problem and hence not taken into account by general problem solvers. Machine learning offers a natural way to adapt initial strategies to the specific setting and often allows to achieve better results. Starting with the work presented in [20] where reinforcement learning has been used to improve scheduling of NASA space shuttle payload processing, various approaches combine mostly TD(λ) or Q-learning and feedforward networks to adapt simple scheduling strategies to the typical demands in real life scenarios, see e.g. [2], [15], [18]. A preliminary application of support vector machine (SVM) methods to improve search strategies (only in batch mode where full information of a part of the search space is given) can be found in [4]. Most approaches, however, tackle small benchmark problems from computer science such as maze problems or they address problem instances from real life for which no comparison with standard operations research methods exist.

The resource constraint project scheduling problem (RCPSP) constitutes a well-studied NP-hard scheduling problem with various benchmark instances and automatic problem generator [12]. It captures a general production scenario where jobs with precedence constraints are to be processed requiring several limited resources. Classical solutions of RCPSP are often based on local search algorithms or integer linear programming [1], [14]. Due to the large number of parameters of the problem, it shows considerable structure even for artificial instances and it is therefore interesting to investigate the possibility to apply machine learning tools for these type problems in general.

In [8], [9] we considered the capability of reinforcement learning to improve a simple greedy strategy for general RCPSP instances, describing RCPSP as iterative repair problem analogous to [20]. The approach worked well on smaller instances but the time complexity did not allow an application to large-scale domains. In this paper we combine reinforcement learning with an iterative construction procedure which requires just a linear number of construction steps and is thus quite efficient. We demonstrate how reinforcement learning together with value function approximation with the SVM can improve the classical search strategy offered by the regret-based biased random sampling scheme (RBRS). RBRS is a priority-rule based search heuristic which was introduced in [5] and [6] for the RCPSP. It was found to outperform other priority-rule based approaches to this scheduling problem in several studies [13], [16]. In RBRS, jobs are sequentially scheduled following a probabilistic scheme guided by a heuristic priority rule. Since this problem can be interpreted as acyclic search problem in the search space of partial schedules, we can apply the particularly efficient rout algorithm of reinforcement learning [3]. As already mentioned, an SVM is used for approximating the value function [11], achieving a sparse representation and good generalization ability.

We will now first introduce the RCPSP and define the classical iterative RBRS heuristic. Afterwards, a solution of this problem with the rout algorithm of reinforcement learning is proposed whereby an SVM serves as value function approximator. Finally, experiments for RCPSP benchmark instances demonstrate the capability of improving the classical RBRS heuristic with adaptive models.

II. THE RCPSP AND PRIORITY RULE-BASED SCHEDULING

We will consider the following variant of the RCPSP: m jobs and an acyclic graph of precedence constraints are given with edge $i \to j$ indicating that job i is to be finished before job j can be started. Each job j is assigned a duration d_j it takes to process the job and the amount m_{ji} of resource $i = 1, \ldots, n$ the job requires. The resources are limited, i.e. at most c_i units of resource i are available at each time step. A schedule consists in a (possibly partial) allocation of the jobs to certain time slots. The jobs must not be interrupted during processing. Thus, we can characterize a schedule by the time points t_j at which the job j starts for all scheduled jobs j. We call a schedule *complete* if all jobs are allocated to time points $t_j \geq 0$. A schedule is *feasible* if it does neither violate precedence constraints, i.e.

$$t_i + d_i \leq t_j$$

holds for all $i \rightarrow j$ which are scheduled, nor resource restrictions, i.e.

$$\sum_{j:t\in[t_j,t_j+d_j)} m_{ji} \leq c_i$$

holds for all time points t of the schedule and resources i. The *makespan* of a schedule is the earliest time point when all jobs are completed, i.e. the value

$$\max_j \{t_j + d_j\}.$$

The goal is to find a feasible and complete schedule with minimum makespan, in general a NP-hard problem [12]. RCPSP formalizes a typical problem as it occurs in production processes or project planning.

Priority-rule based scheduling refers to a classical iterative greedy method. Starting from the empty schedule $s_\emptyset$, iteratively a candidate list of jobs is computed, a job from this candidate list is chosen, and it is added to the schedule.[1] A popular strategy chooses the respective candidate list as the set of so-called *active jobs*, i.e. jobs all predecessors of which in the precedence graph have already been scheduled. The chosen job is scheduled as early as possible such that neither precedence constraints nor resource constraints are violated in the resulting schedule. After m iterations, m denoting the number of jobs, a complete and feasible schedule is obviously obtained. It can be shown that an optimum schedule can be achieved in this way if the job from the candidate list is chosen appropriately in each step [17]. Naturally, it is not known which job of the set of active jobs has to be chosen, the RCPSP being an NP-hard problem. Priority-rule based scheduling assigns a heuristic priority value $v(j)$ to each job j which indicates the benefits for job j to be scheduled early. A priority value which showed very robust and good behavior in connection with serial scheduling assigns each job j its tail [13]. Thus, $v(j)$ denotes the length of a longest path in the precedence graph starting from job j if edges $i \rightarrow k$ are weighted by the duration of job k. This value constitutes a lower bound on the minimum processing time required for the successors of job j. Consequently, scheduling jobs with heavy tails first mirrors the intuition to minimize this lower bound of the makespan.

[1]This is the so-called serial method, in which one job is scheduled at a time. We do not consider parallel methods which schedule more than one job at a time. Typically, parallel methods are faster if each job requires a large set of resources; however, they search in the set of so-called non-delay schedules which need not contain an optimum [13], [17].

Deterministic (single pass) priority-rule based scheduling chooses the job j from the set of active jobs for which $v(j)$ yields the best (largest) value. This deterministic approach, however, often leads to suboptimal or bad schedules. The method can be improved by adding multi-pass sampling to the process. The regret based biased random sampling scheme (RBRS) obeys the following selection rule: for each job j of the candidate list C the *regret* is defined as the value

$$\text{reg}(j) = v(j) - \min_{j'\in C} v(j').$$

The regret of a job measures the worst-case consequence that might arise from selecting another job with respect to the heuristic priority value v. For this randomized selection, job j is selected with probability

$$p_{\text{reg}}(j) = (\text{reg}(j) + \epsilon)^\alpha / \sum_{j'\in C} (\text{reg}(j') + \epsilon)^\alpha$$

where $\epsilon > 0$ and $\alpha > 0$. These parameters modify the regret function in order to avoid zero probability and to balance the difference between high and low priorities, respectively. During experiments in this paper, we set $\epsilon = 1$ and $\alpha = 2$. As priority rule we choose $v(j)$ as the size of the tail of job j in the precedence graph.

We will in the following investigate the possibility to improve the RBRS search strategy by adaptation with reinforcement learning.

III. Learning the search strategy with the SVM

We have formulated the RCPSP as an iterative decision problem: starting from the empty schedule, single jobs are iteratively added until a complete schedule is reached. Note that this decision process is acyclic and those decisions are optimum which finally lead to a complete schedule with minimum makespan. The final makespan, however, is in general unknown. A well-established heuristic method to guide the search process in this decision problem is offered by simple priority rules, as introduced above. Reinforcement learning offers a natural way to improve initial heuristics by problem dependent strategies learned from data [18]. The goal of reinforcement strategies is to optimize the overall reward, in our case this refers to the minimization of the final makespan of the found schedule. Since the search domain is acyclic, we use the rout algorithm as proposed in [3] as a particularly efficient algorithm for acyclic domains.

Before describing the algorithm in detail, we introduce a generalization of the makespan to possibly infeasible schedules in analogy to [20]. The purpose of this definition is to also measure the violations of resource constraints apart from the makespan. For a possibly infeasible schedule s the *total resource utilization index* TRUI(s) is defined as the value

$$\text{TRUI}(s) = \sum_{t,k} \max\{1, \sum_{j:t_j\leq t<t_j+d_j} m_{jk}/c_k\}\,.$$

This measures the overallocation over the whole scheduling period and all resources. It yields the makespan for feasible schedules. Define by s_0 the complete but possibly infeasible schedule which arises if each job is scheduled as early as possible satisfying precedence constraints but possibly violating resource restrictions. s_0 can easily be computed, and properties of s_0 may be used to assess the complexity of the problem. The *resource dilation factor* RDF(s) is given as

$$\text{RDF}(s) = \text{TRUI}(s)/\text{TRUI}(s_0).$$

RDF is just a scaled version of TRUI to account for different size instances. Since TRUI is equal to the makespan for feasible schedules, RDF may serve as a scaled version of the makespan which depends on the complexity of the problem but not its size. We will later use RDF as a feature to learn the value function and, in addition, as the value function which shell be learned by the reinforcement approach.

Denote by S the set of (possibly incomplete and infeasible) schedules. We write $s \succ s'$ if schedule s' can be obtained from s by extending s by one job j from the candidate list as described above for the iterative construction method. Thereby, j need not be chosen according to a priority rule $v(j)$ but it can be an arbitrary job from the list of candidates. We write $s \succ^* s'$ if s' can be reached from s in multiple extension steps by iterating this process. Note that the set of schedules S can be interpreted as a directed graph which connections are given by edges $s \succ s'$. Paths in the graph correspond to tuples $s \succ^* s'$. Priority rule based iterative scheduling aims at finding a path within this graph starting from the empty schedule $s_\emptyset$ which leads to a complete schedule with minimum makespan, i.e. minimum RDF. An optimum search strategy can be found if we always extend partial schedules to the successor s for which the value

$$\begin{aligned}&\text{opt}_{RDF}(s) :=\\ &\max\{1/\text{RDF}(s') \mid s \succ^* s', s' \text{ is complete}\}\end{aligned}$$

is maximum. We choose this function opt_{RDF} as value function of reinforcement learning which has to be learned by our algorithm for relevant regions of the search space.

This value function obeys the Bellman equality

$$\text{opt}_{RDF}(s) = 1/\text{RDF}(s)$$

for complete schedules s and

$$\text{opt}_{RDF}(s) = \max\{\text{opt}_{RDF}(s') \mid s \succ s'\}$$

for incomplete schedules s'. Using this equality, rout tries to approximate the value function opt_{RDF} by a function f_{RDF} starting from the frontier states in the graph of schedules. We will use an SVM to realize f_{RDF} and we will explain its form and training method later. First, we introduce the global procedure of the rout algorithm adapted to our case. Define

$$f_{\text{step}}(s) := \begin{cases} 1/\text{RDF}(s) & \text{if } s \text{ is complete} \\ \max\{f_{\text{RDF}}(s') \mid s \succ s'\} & \text{otherwise} \end{cases}$$

Rout consists in the following steps:

```
initialize f_RDF and the training set T;
repeat: p_out := hunt_frontier_state(s_∅, []);
        add the last k patterns of the returned path
        p_out to T and retrain f_RDF;
```

whereby $[\,]$ denotes the empty path. The procedure hunt_frontier_state(s, p) takes as input a schedule s and a path p from $s_\emptyset$ to the schedule s together with the values under the (approximated) value function, and it returns a path in the graph which leads to a frontier state for which the Bellman equality is violated together with the values of the schedules along the path under the (approximated and corrected) value function. The procedure is defined as follows

```
hunt_frontier_state(s, p_0)
{   repeat h times:
      for all s ≻ s':
         generate a path p from s' to a complete
         schedule in the graph and compute the value
         f_RDF for all schedules along the path;        (*)
         if |f_RDF(s'') − f_step(s'')| > ε for some s'' ∈ p:
               choose the last such s'' in p;
               denote by p_s'' the concatenation of p_0
               and the prefix of p which ends in s'';
               hunt_frontier_state(s'', p_s'');
               exit;
      set the value of s in p_0 to f_step(s);
      set the value of all predecessors s' of s in p_0 to
      max{f_RDF(s'), f_step(s)};
      return(p_0);  }
```

This procedure searches for frontier states in the search graph for which the Bellman equality is not satisfied. Thereby, a small deviation ϵ is tolerated. We set $\epsilon = 0.02$ in our experiments. For efficiency, the test whether the Bellman equality holds is restricted to a part of the search space. Sampling checks up to h paths for each successor of s, whereby we choose $h = 20$ in our experiments.

The found frontier state together with the corrected function value is added to the training set for the value function. For efficiency, we also add some of the possibly corrected predecessors of the frontier state to the training set. The quantity k ranges from 1 to 6. It is controlled by the quality of the frontier state: a higher value of $f_{\text{step}}(s)$ increases k in order to encourage learning on the better results. Since frontier states are determined by sampling in $(*)$, a consistency check is done before adding new training patterns to the training set, deleting old examples which are contradictory to the new ones.

The sampling strategy in $(*)$ determines which part of the search space is explored. It is crucial to guide search towards promising regions of the search space and to find an appropriate balance of exploration and exploitation. We choose scheduling steps on p in $(*)$ based on a mixture of the initial RBRS heuristics and the learned value function. A job j of the candidate list of active jobs is added to the current schedule with probability

$$p(j) = \alpha \cdot p_{\text{rdf}}(j) + (1 - \alpha) \cdot p_{\text{reg}}(j)$$

As above, $p_{\text{reg}}(j)$ denotes the probability defined by regret based biased random sampling based on the heaviest-tail-first heuristic. p_{rdf} denotes the singular probability measure which assigns 1 to the job j from the candidate list which yields to the maximum value $f_{\text{RDF}}(s')$ whereby s' is the extension of the actual schedule by job j. The parameter α is linearly increased from 0 to 1 during training, hence starting with the search strategy given by RBRS and switching to the one-step-lookahead strategy based on the adapted value function f_{RDF} if enough exploration has been done.

For initialization, f_{RDF} is trained on a set of 150 frontier states and their predecessors obtained via search according to RBRS. Retraining of f_{RDF} only takes place each time after a set of 20 new frontier states and predecessors has been added to T. Moreover, a SVM is used as approximator f_{RDF} in our case. The SVM is already uniquely determined by a subset of the training set, the support vectors. We can therefore reduce T to the support vectors after training, hence allowing us to keep the size of the training set T nearly constant.

As already mentioned, f_{RDF} is given by a SVM trained on T. The SVM constitutes a universal learning algorithm for functions between real vector spaces with polynomial training complexity [10], [11], [19]. Since the SVM aims at minimizing the structural risk directly, we can expect very good generalization ability even for few training patterns. Moreover, solutions can be represented by sparse expansions of the support vectors. In order to use a standard SVM, schedules are represented in a finite dimensional vector space by characteristic features of the schedule and the specific search strategy as follows:

- The current partial schedule is described by characteristic statistical features: the percentage of jobs which are already scheduled, the mean and variance of the duration between the starting time of a job and the latest finish time of its predecessors (the idle time slacks), and the overall number of time windows (time windows are periods of the schedule where the set of jobs processed within this period does not change).
- In order to estimate the complexity to complete the actual partial schedule and to observe possible conflicts, we add characteristics of the trivially completed and possibly infeasible schedule which schedules all remaining jobs as early as possible respecting precedence constraints but not resource restrictions. The statistical quantities taken into account are: the mean and variance of free resource capacities, the RDF, the number of overallocated resources compared to the number of overallocated resources in the trivial schedule s_0, and various characteristics which describe the number of time windows of constant job allocation which violate resource constraints.
- To account for the specific search method, we finally add characteristics of the schedule obtained by completion of the actual schedule according to the deterministic priority rule. We describe this schedule by the mean and variance of free resource capacities, and the mean and variance of idle time slacks.
- In addition, we use the TRUI of s_0 to include information on the complexity of the problem.

Note that this representation is largely independent of the number of jobs and resources.

We use a real-valued SVM for regression with ϵ-insensitive loss function and ANOVA-kernel as provided e.g. in the publicly available SVM-light program by Joachims [11]. We could, of course, use alternative proposals of SVM for regression such as least squares SVM [19]. The final (dual) optimization problem for the Lagrange variables α_i, α_i^* of the SVM with ϵ-insensitive loss, given a set of training patterns $(x_i, y_i) \in \mathbb{R}^n \times \mathbb{R}$ reads as follows:

$$\begin{aligned}
&\text{minimize}\\
&\epsilon \cdot \textstyle\sum_i(\alpha_i + \alpha_i^*) - \sum_i y_i(\alpha_i - \alpha_i^*)\\
&\;+ 0.5 \cdot \textstyle\sum_{ij}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)k(x_i, x_j)\\
&\text{such that}\\
&\textstyle\sum_i(\alpha_i - \alpha_i^*) = 0, \quad 0 \le \alpha_i, \alpha_i^* \le C \text{ for all } i
\end{aligned}$$

where $\epsilon > 0$ defines the approximation accuracy, $C > 0$ the tolerance with respect to errors, and $k(a,b) = (\sum_i \exp(-\gamma(a_i - b_i)))^d$ defines the ANOVA kernel for input vectors a and b with components a_i resp. b_i. f_{RDF} can be derived from the dual variables as $f_{\text{RDF}}(x) = \sum_i(\alpha_i - \alpha_i^*)k(x, x_i) + \alpha_0$, where $\alpha_i, \alpha_i^* \neq 0$ holds only for a sparse subset of training points x_i, the support vectors, and the bias α_0 can be obtained from the equation $f_{\text{RDF}}(x_i) = y_i \pm \epsilon$ for support vectors x_i with $\alpha_i, \alpha_i^* < C$. In our experiments, we choose C as 0.05, ϵ as 0.02, γ as 0.1, and d as 3.

IV. Experiments

We apply reinforcement learning in comparison to RBRS with the heaviest-tail-first heuristics to 10 benchmark instances with 60 jobs and 4 resources from [12]. The instances are numbered j6014_1.sm ... j6014_10.sm. They constitute middle sized problems generated randomly with the problem generator as described in [12] with the following characteristics: the mean number of successors per job in the precedence graph is 1.5, all jobs require all resources, and the capacity of each recourse in comparison to the minimum needed capacity (i.e. the maximum requirement m_{ij} for all jobs i), and the maximum needed capacity (the maximum accumulated resource utilization for all jobs in s_0 taken over all time steps) is quantified by 0.5. This latter index refers to the fact that an optimum schedule lies half-way between a 'linear' schedule which assigns only a single job to each time step, and the 'compact' schedule s_0. Thus these instances constitute reasonably hard problems.

We compare the results obtained by sampling with RBRS after iterative random construction of N complete schedules, $N \in \{100, 400, 1000, 2000, 5000\}$, and the results obtained by the rout algorithm after initial training with 150 frontier states and N calls of the procedure hunt_frontier_state where $N \in \{100, 400\}$. The procedure generates in the mean two search paths after initial training (for the initial training set, only one path is generated), such that the number of 100 calls of hunt_frontier_state are comparable to 350 iterations of RBRS, 400 calls of the procedure are comparable to 950

	1	2	3	4	5	6	7	8	9	10
RBRS										
100	67	70	71	74	65	69	75	**88**	64	83
400	66	70	69	72	63	68	74	**88**	64	81
1000	66	68	69	**71**	63	67	74	**88**	64	81
2000	66	**67**	69	**71**	63	67	73	**88**	**63**	**80**
5000	**65**	**67**	69	**71**	63	67	73	**88**	**63**	**80**
steps	2184	1571	137	551	733	463	1167	7	1286	3982
ROUT										
100	66	**67**	**68**	72	63	67	73	**88**	64	82
400	**65**	**67**	**68**	**71**	**62**	**65**	**69**	**88**	**63**	**80**
opt	61	65	62	65	59	65	69	88	61	73

TABLE I

COMPARISON OF THE MAKESPAN OF THE BEST SCHEDULE FOUND BY THE REINFORCEMENT LEARNING APPROACH AND RBRS FOR 10 DIFFERENT INSTANCES WITH 60 JOBS PER INSTANCE, TAKEN FROM [12]. THE RESPECTIVE BEST FOUND VALUE IS DENOTED IN BOLDFACE. THE NUMBERS IN THE LEFTMOST COLUMN REFER TO THE NUMBER OF CONSTRUCTED PATHS FOR RBRS AND THE NUMBER OF CALLS OF THE PROCEDURE HUNT_FRONTIER_STATE FOR ROUT, RESPECTIVELY. THE LINE DENOTED BY 'STEPS' REFERS TO THE EXACT NUMBER OF PATHS CONSTRUCTED UNTIL RBRS FOUND THE RESPECTIVE BEST VALUE. IN EIGHT CASES, ROUT IMPROVES THE RESULT OF RBRS FOUND AFTER 1000 ITERATIONS. THE RESULTS OF ROUT IMPROVE THE SOLUTION COMPARED TO RBRS RUN FOR 5000 ITERATIONS STILL IN FOUR OF THE CASES. THE LAST LINE OF THE TABLE DENOTES THE VALUES OF OPTIMUM SCHEDULES FOR THESE INSTANCES AS GIVEN IN [12]. THIS OPTIMUM IS FOUND BY ROUT IN THREE CASES, BUT ONLY ONCE FOR RBRS.

iterations of RBRS with respect to the number of generated paths.

The obtained results are depicted in Table I. The solutions found by ROUT after 400 iterations is better than the solution found by RBRS after 5000 iterations in four cases, and it is better than the solution found by RBRS after 1000 iterations in eight cases. For three cases, ROUT could find the optimum schedule as reported in [12], RBRS found the optimum in only one case even after 5000 iterations. We also report the number of iterations after which RBRS found the respective best schedule (row 'steps'). In most cases, the best schedule of RBRS is found after at most 2000 iterations such that we cannot expect to find improved schedules using this strategy in a reasonable amount of time. ROUT adapts its strategy to the search space. Correspondingly, the results after a short training time, 100 iterations, are comparable or slightly better than the solutions found by RBRS after 400 steps. The solution found by RBRS after 400 iterations are in all but two cases better than the RBRS solutions after 1000 steps. Moreover, these solutions are still better than RBRS solutions after 5000 steps in four cases. Thus, ROUT allows to improve the search strategy by adaptation to the training data.

V. CONCLUSIONS

We have investigated the possibility to improve initial heuristics for scheduling with machine learning techniques for classical benchmark instances of the RCPSP, a well studied problem in operations research. We thereby compared a learned strategy to regret based biased random sampling, a simple and powerful iterative construction heuristic proposed for the RCPSP. We formulated this problem as acyclic search problem and proposed an adaptation of the rout algorithm of reinforcement learning together with the SVM as function approximator for the task. The approach could improve the results found by RBRS without additional value function on several medium sized benchmark instances.

Directions of further work could deal with problems with different characteristics (e.g. more easy instances), and larger sized problems. In addition, no feature optimization of the representation of partial schedules has been done so far. This could address two problems: a more expressive representation of the actual search state and features which are more efficient to compute. As an alternative, SVMs working directly on graphs would be interesting, see e.g. [7] for an overview about this topic. In [9], alternative value functions have also been tested such as an approach which just approximately gives the correct order of successor states $s \succ s'$ corresponding to opt_{RDF}. It would be interesting to evaluate whether these alternatives lead to improvements and, in particular, speed-up of the search strategy proposed in this article.

REFERENCES

[1] T.Baar, P.Brucker, and S.Knust, Tabu-search algorithms and lower bounds for the resource-constraint project scheduling problem, Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization, 1-18, Kluwer, 1998.

[2] A.G.Barto and R.H.Crites, Improving elevator performance using reinforcement learning. NIPS 8, 1017-1023, MIT Press, 1996.

[3] J.A.Boyan and A.W.Moore, Learning evaluation functions for large acyclic domains, Proc.ICML, 14-25, 1996.

[4] T.G.Dietterich and X.Wang, Batch value function approximation via support vectors, In: T.G.Dietterich, S.Becker, Z.Ghahramani, Z. (eds.), Advances in Neural Information Processing Systems 14, MIT Press, Cambridge, MA.

[5] A.Drexl, Scheduling of project networks by job assignment, Management Science 37, 1590-1602, 1991

[6] A.Drexl and J.Grünewald, Nonpreemptive multi-mode resource-constrained project scheduling, IEE Transactions 25, 74-81, 1993.

[7] T.Gärtner, A survey of kernels for structured data, ACM SIGKDD Explorations Newsletter 5(1):49-58, 2003.

[8] K.Gersmann and B.Hammer, Improving iterative repair strategies for scheduling with the SVM, Proc.ESANN, 235-240, 2003.

[9] K.Gersmann and B.Hammer, Improving iterative repair strategies for scheduling with the SVM, submitted to Neurocomputing.

[10] B.Hammer and K.Gersmann, A note on the universal approximation capability of SVMs, Neural Processing Letters 17:43-53, 2003.

[11] T.Joachims, Learning to Classify Text Using Support Vector Machines, Kluwer, 2002.

[12] R.Kolisch and A.Sprecher, PSBLIB – a project scheduling library, European Journal of Operational Research 96, 205-219, 1996.

[13] R.Kolisch and A.Drexl, Adaptive search for solving hard project scheduling problems Naval Research Logistics 43, 23-40, 1996.

[14] A.Mingozzi, V.Maniezzo, S.Ricciardelli, L.Bianco, An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation, Management Science 44, 714-729, 1998.

[15] S.Riedmiller and M.Riedmiller, A neural reinforcement learning approach to learn local dispatching policies in production scheduling, Proc.IJCAI, 1074-1079, 1999.

[16] A.Schirmer and S.Riesenberg, Advanced biased random sampling in serial and parallel scheduling, Research Report, Universität Kiel, 1997.

[17] A.Sprecher, R.Kolisch, and A.Drexl. Semi-active, active, and non-delay schedules for the resource-constraint project scheduling problem. European Journal of Operational Research, 80:94-102, 1995.

[18] R.Sutton and A.Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.
[19] J.A.K.Suykens, T.Van Gestel, J.De Brabanter, B.De Moor, and J.Vandewalle, Least Squares Support Vector Machines, World Scientific Pub. Co., 2002.
[20] W.Zhang and T.G.Dietterich, A reinforcement learning approach to job-shop scheduling, Proc.IJCAI, 1114-1120, 1995.