Improving iterative repair strategies for scheduling with the SVM

Kai Gersmann, Barbara Hammer

Research group LNM, Department of Mathematics/Computer Science, University of Osnabrück, Germany

Abstract

Resource constraint project scheduling (RCPSP) is an NP-hard benchmark problem in scheduling which takes into account the limitation of resources' availabilities in real life production processes and subsumes open-shop, job-shop, and flow-shop scheduling as special cases. We here present an application of machine learning to adapt simple greedy strategies for the RCPSP. Iterative repair steps are applied to an initial schedule which neglects resource constraints. The rout-algorithm of reinforcement learning is used to learn an appropriate value function which guides the search. We propose three different ways to define the value function and we use the support vector machine (SVM) for its approximation. The specific properties of the SVM allow to reduce the size of the training set and SVM shows very good generalization behavior also after short training. We compare the learned strategies to the initial greedy strategy for different benchmark instances of the RCPSP.

Key words: RCPSP, SVM, reinforcement learning, ROUT algorithm, scheduling

1 Introduction

Resource constraint project scheduling (RCPSP) is the problem to schedule a number of jobs on a given number of machines such that the overall completion time is minimized. Thereby, precedence constraints of the jobs are to be taken into account and the jobs require different amounts of (renewable) resources of which only a certain amount is available at each time step. Problems of this type occur frequently in industrial production planning or project management, for example. As a generalization of job-shop scheduling, the RCPSP constitutes an NP-hard optimization problem [6]. Thus exact solutions serve merely as benchmark generators

Email address: {kai,hammer}@informatik.uni-osnabrueck.de (Kai Gersmann, Barbara Hammer).

rather than efficient problem solvers for realistic size problems. Most exact solvers rely on implicit enumeration and backtracking such as branch and bound methods as proposed in [12,15,31]. Alternative approaches have been based on dynamic programming [13] or zero-one programming [34]. Exact approaches, however, may lead to valuable lower bounds [11]. A variety of heuristics has been developed for the RCPSP which can also solve realistic problems in reasonable time. The proposed methods can roughly be differentiated into four paradigms: priority based scheduling, truncated branch and bound methods, methods based on disjunctive arcs, and metaheuristics [23]. Thereby, priority based scheduling iteratively expands partial schedules by candidate jobs for which all predecessors have already been scheduled. This might be done in a single pass or multiple passes and it relies on different heuristics which job to choose next [22,27,36]. Truncated branch and bound methods perform only a partial exploration of the search tree constructed by branch and bound methods whereby the exploration is guided by heuristics [1]. As alternative method, precedence constraints can be enlarged by disjunctive arcs which make sure that the resource constraints are met, i.e. technologically independent jobs which cannot be processed together because of their resource requirements are taken into account [4]. Metaheuristics for the RCPSP include various local search algorithms such as simulated annealing, tabu search, genetic algorithms, or ant colony optimization [2,17,25,30,33,38]. The critical part of iterative search strategies is thereby the representation of instances and the definition of the neighborhood graph [26]. Apart from its widespread applicability in practical applications, the RCPSP is an interesting optimization problem because a variety of well studied benchmarks is available. A problem generator which provides different size instances depending on several relevant parameters such as the network complexity or the resource strength is publicly available on the web [24]. At the same site, benchmark instances together with the best lower and upper bounds found so far can be retrieved.

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. The specific structure, however, might allow to find better heuristics of the problem. Often, humans can solve instances of theoretically NP-complete scheduling tasks in a specific domain in short time based on their experience on previous examples; i.e., humans use their implicit knowledge about typical problem settings in the domain. Machine learning offers a natural way to adapt initial strategies to a specific setting based on examples. Thus it constitutes a possibility to improve general purpose problem solvers for concrete domains. Starting with the work of [3,47], machine learning has successfully been applied to various scheduling problems [8]. The approach [47] thereby uses TD(λ), a specific reinforcement learning method, together with feedforward networks for an approximation of the value function to improve initial greedy heuristics for scheduling of NASA space shuttle payload processing. The trained strategies generalize to new instances of similar type such that an efficient solution of typical scheduling problems within this domain is possible based on the learned heuristics. The approaches [7,32] are also based on TD(λ), but they use simple regression models for value-function approximation. The application area is here the improvement of local search methods for various theoretical NP-hard optimization problems including bin packing, the satisfiability problem, and the traveling salesperson problem. Again, local search methods could successfully be adapted to specific problem instances. [41] combines a lazy learner with a variant of TD(0) for problems in production scheduling and reports promising results. The work [40] includes comparisons to an alternative reinforcement learner for the same setting, the rout algorithm, which can only be applied to acyclic domains but which is guaranteed to converge [9]. Further machine learning approaches to scheduling problems include: an application to schedule program blocks for different programming languages including C and Fortran [28]; simulated annealing in combination with machine learning to learn placement strategies for VLSI chips [43]; Q-learning, another reinforcement strategy, in combination with neural networks to learn local dispatching heuristics in production scheduling [37]; distributed learning agents for multi-machine scheduling [10] or network routing [46], respectively; and a direct integration of case based reasoning to scheduling problems [39]. Thus machine learning is capable of improving simple scheduling strategies for concrete domains. However, the reported approaches mostly use concrete problem settings from practical applications or instances specifically generated for the given problem. Thus, it is not clear whether machine learning yields improvements also for standard benchmarks widely used in the operations research literature.

The RCPSP possesses a large number of problem parameters. Thus, 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. We will consider the capability of reinforcement learning to improve a simple greedy strategy for solving RCPSP instances. Thereby, we will test the approach on the benchmarks provided by the generator [24]. To apply machine learning, we formulate the RCPSP problem as iterative repair problem with a number of repairs limited by the size of the respective instance. Since this problem can be interpreted as acyclic search problem, we can apply the rout algorithm of reinforcement learning [9] which is guaranteed to converge if the approximation of the value function is sufficiently close. For value function approximation, the support vector machine (SVM) is chosen since the SVM provides excellent generalization also if only few examples are available and if input data is high dimensional [14]. In addition, the SVM yields sparse representations such that we can work with reduced training sets. We thereby consider three different ways to assess the value function: a function which results from the Bellman equation, a rank based approach, and a related fast heuristic. We demonstrate the ability of the approach to improve the initial greedy strategy even after few training steps, and we investigate the generalization capability of learned strategies to new RCPSP instances in several experiments.

We will now first introduce the RCPSP and formulate iterated repair steps as a Markov decision process for which reinforcement learning can be applied. We then discuss function approximation by means of the SVM and evaluate the algorithm in several experiments with different size RCPSP instances.

2 Resource constraint project scheduling

We consider the following variant of the RCPSP: m jobs and n resources are given. An acyclic graph specifies the precedence constraints of the jobs, 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 an allocation of the jobs to certain time slots in which they are processed and it can be characterized by the time points $s(j)$ at which the jobs j start, since we do not allow interruption of the jobs. I.e. a list $s = (s(1), \ldots, s(m))$ stands for the schedule in which job i is started at time point $s(i)$ and it takes until time point $s(i) + d_i$ to be completed. A job is said to be *active* at the interval $[s(i), s(i) + d_i)$ in a given schedule. A *feasible schedule* does neither violate precedence constraints nor resource restrictions, i.e. the constraints

$$s(i) + d_i \leq s(j) \text{ for all } i \to j$$

and

$$\sum_{j : t \in [s(j), s(j)+d_j)} m_{ji} \leq c_i \text{ for all time points } t \text{ and resources } i$$

hold. The *makespan* of a schedule is the earliest time point when all jobs are completed, i.e. the value

$$\max_j \{s(j) + d_j\}.$$

The goal is to find a feasible schedule with minimum makespan, in general a NP-hard problem [6]. Note that this formulation is a conceptual one since the starting times $s(j)$ occur in the range of the sum. An alternative formulation which allows to apply mixed integer programming techniques can be found in [35]. More general formulations of the RCPSP which also take into account a time-cost tradeoff, multiple execution modes, time-lacks, or alternative objectives are possible [23].

A lower bound for the minimum achievable makespan is given by the possibly infeasible schedule which schedules each job as early as possible taking the precedence constraints into account but possibly violating resource restrictions. This initial schedule can obviously be computed in polynomial time by adding the durations d_j along paths in the precedence graph. We refer to this initial possibly infeasible schedule by s_0 in the following. In [47] an objective called the resource dilation factor (RDF) is defined which is related to the makespan and takes resource violations into account thus generalizes the makespan to infeasible schedules: given a schedule s, define the *total resource utilization index* $\mathrm{TRUI}(s)$ as

$$\sum_{t,i} \max\{1, \textstyle\sum_{j:t\in[s(j),s(j)+d_j)} m_{ji}/c_i\}$$

where t enumerates the time steps in the schedule and i the resources. Note that the summands indicate the amount of overallocation of resource i at time t, hence $\mathrm{TRUI}(s)$ gives n times the makespan for feasible schedules. The *resource dilation factor* $\mathrm{RDF}(s)$ is defined as the normalization

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

whereby s_0 is the possibly infeasible schedule which allocates all jobs at the earliest possible time step respecting precedence constraints but violating resource constraints. The normalization of $\mathrm{TRUI}(s)$ by $\mathrm{TRUI}(s_0)$ has the effect that the value of the objective $\mathrm{RDF}(s)$ is roughly in the same range for RCPSP problems of different sizes. Thus $\mathrm{RDF}(s)$ is a possible general objective to be learned by machine learning tools. Since $\mathrm{RDF}(s)$ differs from the makespan of s by a constant factor for feasible schedules, we can alternatively state our objective as the task to find a feasible schedule s with minimum $\mathrm{RDF}(s)$.

We now formulate this problem as iterative repair problem: Starting from the possibly infeasible schedule s_0, a feasible schedule can be obtained by repair steps. We consider the following possible repair steps of a given schedule s: for the earliest time point violating a resource constraint, one job i which is active at this time point is chosen with starting time $s(i)$. The job and its successors in the precedence graph are rescheduled. The following two possibilities are considered:

(1) $s(i)$ is either increased by one,
(2) or $s(i)$ is set to the earliest time point such that job i does not lead to resource constraint violations. I.e., $s(i)$ is set to the earliest time point such that for all resources j and all time points $t \in [s(i), s(i) + d_i)$ the constraint $\sum_k m_{kj} \leq c_j$ is fulfilled whereby the sum is over all jobs k which are active at time point t and which are not successors of job i.

All successors of i are then scheduled at the earliest possible time for which the precedence constraints are fulfilled disregarding resource constraints. We denote $s \succ s'$ if s' can be obtained from s by one repair step. $s \succ^* s'$ denotes the fact that s' can be obtained from s by a number of repair steps, whereby this number is arbitrary (possibly 0). Note the following:

- For all schedules s with $s_0 \succ^* s$ all precedence constraints are fulfilled by defi-

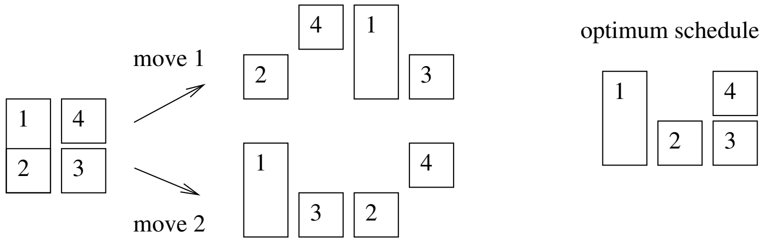

Fig. 1. A simple example for an instance where repair steps (2) yield to suboptimal solutions. An optimal schedule is depicted at the right side.

nition.

- The directed graph with vertices s and edges $s \succ s'$ is acyclic.
- For all paths in this graph which start from s_0, a feasible schedule is found after a polynomial number of repair steps. This is obvious, since precedence constraints are respected for all schedules in such a path, and in each step the earliest possibl time point with resource conflicts is improved.
- At least one path starting from s_0 leads to an optimum schedule.

Note that option (2), rescheduling jobs such that additional conflicts are avoided, yields reasonable repair steps and promising search paths. However, we cannot guarantee to reach optimum schedules starting from s_0 solely based on repair steps of type (2) and thus have to also include (1). (2) is similar in spirit to so-called priority based scheduling with parallel priority rules, a greedy strategy which constructs schedules from scratch scheduling each job as early as possible taking into account precedence and resource constraints [21]. It is well known that parallel priority rules only yield so-called non-delay schedules, which need not contain an optimum schedule [21,42]. Since we start from s_0, we get also other schedules. However, an optimum may not be reachable from s_0 using only (2) as the following example shows: consider a RCPSP instance with 4 jobs and 2 machines. All jobs have unit duration time, and the precedence constraints are given by $1 \to 3$ and $2 \to 4$. The resource constraints are $c_1 = c_2 = 1$. Jobs 1, 2, and 3 require one unit of resource 1, jobs 1 and 4 require one unit of resource 2. Fig. 1 shows the initial schedule s_0 and the two schedules obtained when applying repair steps 2. Both schedules are longer than the optimum schedule, which is also depicted in Fig. 1.

If repair steps (1) are integrated, optimum schedules can be reached from s_0 as can be seen as follows: note that the starting times $s(i)$ in s_0 constitute a lower bound on the starting times of every feasible schedule. In addition, the jobs are scheduled at the earliest possible time with respect to precedence constraints. One can iteratively apply repair steps (1) to s_0 such that the following two properties are maintained for the resulting schedule s:

- For a given fixed optimum feasible schedule s^* the inequality $s(i) \leq s^*(i)$ holds.
- Denote by t the earliest time point in s where resource constraints are violated. Then all jobs which are successors of jobs active at time points $\geq t$ are scheduled as early as possible with respect to precedence constraints (ignoring resource

constraints).

This can be achieved if we choose a job i in a repair step (1) for which $s(i) < s^*(i)$ holds. Such job i exists because s^* would otherwise not be feasible. For the new schedule s', $s'(i) \leq s^*(i)$ is still valid. All successors of this job are scheduled at the earliest possible time steps, and all other jobs are not rescheduled. Thus the above two properties hold, because s^* also respects precedence constraints. Note that the first property implies that s, if feasible, is itself an optimum schedule.

We can thus solve the RCPSP by iterative search in this acyclic graph starting from s_0. Efficient strategies rely on heuristics which parts of the graph should be explored. Assume a value function

$$f_{\text{val}} : s \mapsto f_{\text{val}}(s) \in \mathbb{R}$$

is given which evaluates the possibly heuristic preference to consider the (possibly infeasible) schedule s. Any given evaluation function f_{val} can be integrated into a simple *one-step lookahead strategy* as follows:

$$t := 0; \;\; \text{compute } s_0;$$

$$\text{repeat until } s_t \text{ is feasible:}$$

$$s_{t+1} := \text{argmax}_{s_t \succ s'} f_{\text{val}}(s'); \;\; t := t + 1;$$

Of course, there cannot exist simple and general strategies of how to choose each repair step optimum, the RCPSP being an NP-hard problem. One simple greedy strategy which likely yields good schedules is to choose always that repair step $s \succ s'$ such that the local RDF s', is optimum among all schedules directly connected to s. I.e. we can choose f_{val} as

$$f_{\text{val}}^{\text{greedy}}(s) := 1/\text{RDF}(s)\,.$$

We refer to the feasible schedule obtained by this heuristic value function starting from s_0 as s_{greedy}. We will in the following investigate the possibility to improve this greedy strategy based on local RDF by adaptation with reinforcement learning.

3 Reinforcement learning and rout-algorithm

We have formulated the RCPSP as an iterative decision problem: starting from s_0, repair steps are iteratively applied until a feasible schedule is reached. Thereby, those decisions are optimum which finally lead to a feasible schedule with minimum RDF. We thus obtain an *optimum strategy* if we choose

$$f_{\text{val}}^{\text{opt}}(s) := \max_{s \succ^* s'} 1/\mathrm{RDF}(s')$$

This function is in general unknown. Reinforcement learning offers a possibility to learn this optimum strategy or a variant thereof based on examples [44]. The key issue is thereby the Bellman equality:

$$f_{\text{val}}^{\text{opt}}(s) = \begin{cases} 1/\mathrm{RDF}(s) & \text{if } s \text{ is feasible} \\ \max_{s \succ s'} f_{\text{val}}^{\text{opt}}(s') & \text{otherwise} \end{cases}$$

Note that this equality uniquely determines the optimum strategy. Popular reinforcement strategies include TD(λ) to learn the value function based on the Bellman equation, and Q-learning which directly adapts policies for which a similar equation holds [20]. The algorithms are guaranteed to converge for discrete spaces [5]. If the value function is approximated e.g. by a linear function or a neural network, however, which is learned during exploration of the search space, problems might occur and convergence is in general not guaranteed [29]. In our case acyclic domains are given. We can thus use the rout algorithm as proposed in [9]. The rout algorithm tries to enlarge the training set only by valid training examples. It first adds the last schedules on a given path to the training set for which the Bellman equality is not fulfilled and thus the value function not yet learned correctly. Some function approximator is repeatedly trained on the stored training examples until the Bellman equality is valid for all states. Rout is guaranteed to converge if a sufficiently close approximation of the value function can be found.

Using the Bellman equality, rout tries to learn the value function $f_{\text{val}}^{\text{opt}}$ by a function f_{rout} starting from the frontier states in the repair graph. Denote by f_{step} the function

$$f_{\text{step}}(s) = \begin{cases} 1/\mathrm{RDF}(s) & \text{if } s \text{ is feasible} \\ \max_{s \succ s'} f_{\text{rout}}(s') & \text{otherwise} \end{cases}$$

Note that $f_{\text{rout}} = f_{\text{step}}$ implies that f_{rout} is the optimum strategy $f_{\text{val}}^{\text{opt}}$. Rout consists in the following steps:

initialize f_{rout} and training set T;

repeat: hunt_frontier_state(s_0);

add the returned pattern to T and retrain f_{rout};

where

hunt_frontier_state(s)

{ repeat h times: for all $s \succ s'$:

generate a repair path p from s' to a feasible schedule; (*)

if $|f_{\mathrm{rout}}(s'') - f_{\mathrm{step}}(s'')| > \epsilon$ for some $s'' \in p$:

hunt_frontier_state(s'') for the last such $s'' \in p$; exit;

return the pattern $(s, f_{\mathrm{step}}(s))$; }

I.e., this procedure finds a frontier state, a last state in the repair graph for which the Bellman equality is not fulfilled for the learned approximation of the value function. This is tested by sampling, for efficiency. Thereby, we typically choose $h = 20$ and we allow small deviations from the exact Bellman equality, setting ϵ to a small positive value.

It is essential to guarantee that promising regions of the search space are covered and the value function is closely approximated in these regions. At the same time, it has to be ensured that the whole search space is covered to such a degree that all relevant regions are detected. To make a reasonable compromise between exploration and exploitation, we choose repair steps on the search path p in $(*)$ based on the following heuristic: the successor of a schedule s'' is chosen as s''' with $s'' \succ s'''$ and

$$s''' = \begin{cases} \mathrm{argmax}_{s'''} f_{\mathrm{val}}^{\mathrm{greedy}}(s''') & \text{with probability } p, \\ \mathrm{argmax}_{s'''} f_{\mathrm{rout}}(s''') & \text{with probability } (1-p)^2, \\ \text{a random successor} & \text{with probability } p(1-p). \end{cases}$$

Thereby, $p \in [0,1]$ is linearly decreased from 1 to 0 during training. Search first explores regions of the search space for which the initial heuristic given by $f_{\mathrm{val}}^{\mathrm{greedy}} = 1/\mathrm{RDF}$ is promising. Once the value function has been learned, it might yield to better solutions and thus the probability $(1-p)^2$ is increased in later steps of the algorithm. Since frontier states are determined by sampling, invalid examples (non frontier states) might be added to the training set for which the maximum one-step-lookahead value is not yet correct. It is thus advisable to add a consistency check when adding new training examples to T, deleting inconsistent previous examples from the training set.

Because of the Bellman equality, it is obviously guaranteed that this algorithm converges if a sufficiently close approximation (better than ϵ) of the value function can be learned from the given training data and if sampling in (*) assigns nonzero probability to all successors. It can be expected that also before convergence of rout, an approximation of $f_{\text{val}}^{\text{opt}}$ is found which improves the initial search strategy. As already mentioned, various different regression frameworks have been combined with reinforcement learning including neural networks, linear functions, and lazy learners. For rout, a sufficiently powerful approximator is to be chosen to guarantee an exploration of the whole space.

4 Approximation of the value function

We use a support vector machine (SVM) for the approximation of the value function [14]. The SVM constitutes a universal learning algorithm for functions between real vector spaces with polynomial training complexity [16,18,45]. Since the SVM aims at minimizing the structural risk directly, we can expect very good generalization ability even for few training patterns.

4.1 Standard SVM

In a first approach, we train a standard SVM to learn the optimum decision function $f_{\text{val}}^{\text{opt}}$ which measures the optimum achievable RDF. In order to use SVM, schedules are represented in a finite dimensional vector space adapting features as proposed in [47] to our purpose. Recall that m denotes the number of jobs, n the number of resources, d_i the duration and m_{ij} the requirements of resource j of job i, and c_j the available amount of resource j at each time step. For a schedule s, $s(i)$ denotes the starting point of job i. The makespan of the schedule is referred to by K. For any real number x, denote $x^+ = \max\{x, 0\}$. The following list of features is used:

- Mean and standard deviation of free resource capacities:

$$m_{\text{free}} = \frac{1}{Kn} \sum_{i=1}^{n} \sum_{k=1}^{K} \left(c_i - \sum_{s(j) \leq k < s(j)+d_j} m_{ji} \right)^+$$

  and

$$s_{\text{free}} = \left(\frac{1}{Kn} \sum_{i=1}^{n} \sum_{k=1}^{K} \left(\left(c_i - \sum_{s(j) \leq k < s(j)+d_j} m_{ji} \right)^+ - m_{\text{free}} \right)^2 \right)^{1/2}$$

- Mean and standard deviation of minimum and average slacks between a job and its predecessors:

$$m_{\text{sl}} = \frac{1}{m} \sum_{j=1}^{m} s(j) - \max\{s(i) + d_i : i \to j\}$$

$$s_{\mathrm{sl}} = \left(\frac{1}{m}\sum_{j=1}^{m}(s(j) - \max\{s(i) + d_i : i \to j\} - m_{\mathrm{sl}})^2\right)^{1/2}$$

$$m_{\mathrm{asl}} = \frac{1}{m}\sum_{j=1}^{m} P(j)^{-1}\sum_{i\to j} s(j) - (s(i) + d_i)$$

$$s_{\mathrm{asl}} = \left(\frac{1}{m}\sum_{j=1}^{m}\left(P(j)^{-1}\sum_{i\to j} s(j) - (s(i) + d_i) - m_{\mathrm{asl}}\right)^2\right)^{1/2}$$

where $P(j) = \max\{1, \text{number of predecessors for } j\}$. Remember that only schedules which are valid with respect to the precedence constraints occur in our case, such that the slacks are always nonnegative.

- The RDF of s and, in addition, a second feature which gives the RDF for feasible schedules and which is zero for infeasible schedules.
- The overallocation index

$$o_{\mathrm{ind}} = \frac{\sum_{i=1}^{n}\sum_{k=1}^{K}\left(\sum_{s(j)\le k< s(j)+d_j} m_{ji} - c_i\right)^+}{\sum_{i=1}^{n}\sum_{k=1}^{K_0}\left(\sum_{s_0(j)\le k< s_0(j)+d_j} m_{ji} - c_i\right)^+}$$

  Here, K_0 denotes the makespan of the initial schedule s_0.
- The percentage of windows with constraint violations. A window is thereby a maximum time period where the set of active jobs does not change.
- The overall number of windows, W.
- The percentage of constraint violations in the first 10 windows after the first constraint violation.
- The percentage of time steps that contain a constraint violation.
- The first violated window index: $(W - w_0)/W$ where W is the total number of time windows and w_0 the index of the first window with constraint violation.
- The total resource utilization index of the start schedule s_0.

These features measure potentially relevant properties of schedules including the feasibility, denseness of the scheduled jobs, etc. Note that this representation allows to transfer the trained value function f_{rout} to new instances even with a different number of jobs and resources, since (almost) scale-free quantities are measured. 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 [18]. We could, of course, use alternative proposals of SVM for regression such as least squares SVM [45]. The final (dual) optimization problem for SVM with

ϵ-insensitive loss, given pattern $(x_i, y_i) \in \mathbb{R}^n \times \mathbb{R}$ reads as follows:

minimize

$$\epsilon \sum_i(\alpha_i + \alpha_i^*) - \sum_i y_i(\alpha_i - \alpha_i^*) + 0.5 \sum_{ij}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)k(x_i, x_j)$$

such that $\quad \sum_i(\alpha_i - \alpha_i^*) = 0,\ 0 \le \alpha_i, \alpha_i^* \le C$ for all i

where $\epsilon > 0$ defines the approximation accuracy, i.e. the size of the ϵ-tube within which deviation from the desired values is tolerated. $C > 0$ regulates the tolerance with respect to errors. $k(a, b) = (\sum_i \exp(-\gamma(a_i - b_i)))^d$ is here chosen as the ANOVA kernel. a_i resp. b_i denote the components of a and b. The regression function f_{rout} can be derived from the dual variables as $f_{\text{rout}}(x) = \sum_i(\alpha_i - \alpha_i^*)k(x, x_i) + b$, where $\alpha_i, \alpha_i^* \neq 0$ holds only for a sparse subset of training points x_i, the support vectors, and the bias b can be obtained from the equation $f_{\text{rout}}(x_i) = y_i \pm \epsilon$ for support vectors x_i with $\alpha_i, \alpha_i^* < C$.

Note that the SVM is uniquely determined by the support vectors, i.e. the points (x_i, y_i) for which $\alpha_i \neq 0$ or $\alpha_i^* \neq 0$ holds. These points constitute a sparse subset of T. We can thus speed up the learning algorithm by deleting all points but the support vectors from the training set after training the SVM.

4.2 Ranked SVM

Rout in combination with the standard SVM algorithm learns the optimum value function $f_{\text{val}}^{\text{opt}}$. However, this is more than we actually need. Any value function f_{val} which fulfills the conditions

$$f_{\text{val}}(s) \ge f_{\text{val}}(s') \iff f_{\text{val}}^{\text{opt}}(s) \ge f_{\text{val}}^{\text{opt}}(s')$$

yields the same solution as an optimum strategy. Such possibly simpler functions can be learned by the *ranking* SVM algorithm which has been proposed by Joachims [19]. We here propose a combination of rout with the approach, which just learns the (potentially simpler) ranking induced by $f_{\text{val}}^{\text{opt}}$. We consider a special case of the algorithm as introduced in [19]. Suppose we are given input vectors x_i with values y_i and a feature map Φ which maps the x_i into a potentially high dimensional Hilbert space. A linear function in the feature space, parameterized by the weights w, ranks the data points according to the ranking induced by the output values y_i, iff

$$\forall i, j : y_i > y_j \iff \langle w, \Phi(x_i)\rangle > \langle w, \Phi(x_j)\rangle\,,$$

$\langle\cdot,\cdot\rangle$ denoting the dot product in the feature space. Thus the ranked SVM tries to find a classifier with optimum margin such that these constraints are fulfilled. To account for potential errors, slack variables are introduced as in the standard SVM case. Thus we achieve an optimization problem very similar to the standard formulation of SVM:

$$\begin{array}{ll} \text{minimize} & \\ \quad \langle w,w\rangle/2 + C\sum_{y_i>y_j}\xi_{ij} & \\ \text{subject to} & \forall y_i > y_j : \langle w,\Phi(x_i)\rangle \geq \langle w,\Phi(x_j)\rangle + 1 - \xi_{ij} \\ & \forall i,j : \xi_{ij} \geq 0 \end{array}$$

This optimization problem is convex, and it is in fact equivalent to the classical SVM problem in the feature space to classify the difference vectors $\Phi(x_i) - \Phi(x_j)$ for $y_i > y_j$ positive; thus, it can be transformed into a dual version which allows us to use kernels:

$$\begin{array}{ll} \text{maximize} & \\ \quad \sum_{y_i>y_j}\alpha_{ij} & \\ \quad -0.5\sum_{y_i>y_j,y_k>y_l}\alpha_{ij}\alpha_{kl}(k(x_i,x_k) - k(x_j,x_k) - k(x_i,x_l) + k(x_j,x_l)) & \\ \text{subject to} \quad \forall i,j : 0 \leq \alpha_{ij} \leq C & \end{array}$$

As beforehand, the classifier can be formulated in terms of the support vectors $x \mapsto \sum_{y_i>y_j}\alpha_{ij}(k(x_i,x) - k(x_j,x))$. If we restrict to linear kernels, the problem can be further reduced to a classical SVM problem in the original space: classifiy the data points $x_i - x_j$ for all $y_i > y_j$ positive with an SVM without bias. In this approach, we use a ranking SVM to learn a function f_{rank} which induces the same ranking of schedules as $f_{\text{val}}^{\text{opt}}$. It can be expected that this learning task is easier than learning the exact optimum $f_{\text{val}}^{\text{opt}}$. We can apply the rout algorithm, as introduced beforehand, to learn f_{rank}. Thereby, the one-step lookahead f_{step} of f_{rank} used in the hunt-frontier-state procedure has to be adapted as follows: denote by T_{feasible} the feasible schedules already collected in the training set T. We set

$$f_{\text{step}}(s) = \begin{cases} \max_{s \succ s'} f_{\text{rank}}(s') & \text{if } s \text{ is infeasible} \\ \max_{s' \in T_{\text{feasible}}} f_{\text{rank}}(s') + 1 & \text{if } s \text{ is feasible and} \\ & \text{RDF}(s) < \text{RDF}(s') \text{ for all } s' \in T_{\text{feasible}} \\ f_{\text{rank}}(s') \text{ with} & \text{otherwise} \\ s' = \text{argmax}_{s' \in T_{\text{feasible}}}\{\text{RDF}(s') \leq \text{RDF}(s)\} & \end{cases}$$

This choice has the effect that the values of the learned ranking f_{rank} are propagated via the Bellman equality starting from frontier states. If we stored the RDF of feasible schedules, the Bellman equality need not hold for functions which just respect the ranking of $f_{\text{val}}^{\mathrm{opt}}$. Thus the function learned in this approach, f_{rank}, is simpler than $f_{\text{val}}^{\mathrm{opt}}$ and potentially simpler SVMs can produce appropriate value functions. However, this training algorithm uses a quadratic number of constraints for SVM training. In addition, we have to access all feasible schedules from the training set to compute f_{step} for feasible schedules. Thus, training is slower than for standard SVM.

4.3 A fast heuristic value function

As mentioned above, it is not necessary to strictly learn the value function $f_{\text{val}}^{\mathrm{opt}}(s)$. It suffices for a value function $f_{\text{val}}(s)$ to induce the same order as $f_{\text{val}}^{\mathrm{opt}}$ if successors of a schedule in the repair graph are ranked according to f_{val}. More precisely, only the *maxima* have to agree if a one-step lookahead is used as in our case, i.e. for all schedules s the condition

$$\text{argmax}_{s \succ s'} f_{\text{val}}(s') = \text{argmax}_{s \succ s'} f_{\text{val}}^{\mathrm{opt}}(s')$$

guarantees optimum decisions. The ranking SVM, as introduced beforehand, guarantees a correct global ranking [19]. However, this algorithm uses a quadratic number of constraints for SVM training, thus it is rather slow in our setting. A different approach is to focus on the weaker condition, that only the maxima have to coincide.

We are only interested in the best (or good) paths. Thus the overall ranking of regions with small value function $f_{\text{val}}^{\mathrm{opt}}$ need not be very precise. Rather, the learned evaluation should correctly rank paths which yield to the best value $f_{\text{val}}^{\mathrm{opt}}$ found so far. We therefore substitute the optimum value function $f_{\text{val}}^{\mathrm{opt}}$ by a direct heuristic function which only roughly approximates the ranking for small values, and which is more precise for good schedules. Since it is not clear before training, which values $f_{\text{val}}^{\mathrm{opt}}$ can be achieved, this function is built during training, focusing on the respective best values found so far. We assume that a sequence $r = (r_i)_{i=1}^{\infty}$ of real numbers is given, which correspond to the values $f_{\text{val}}^{\mathrm{opt}}(s_i) = 1/\mathrm{RDF}(s_i)$ of feasible schedules s_i found during training in the order of their appearance. We consider the subsequence of values which correspond to improvements, i.e. the strictly monotone subsequence $\bar{r}$ of r with values $\bar{r}_1 = r_1$, and $\bar{r}_i$ being the first value in the sequence r when deleting all values not larger than $\bar{r}_{i-1}$. We now project the range of possible values to a range corresponding to these improving steps, thereby stretching the actual best regions and compressing regions where the value function is low; define $g_r : \mathbb{R} \to \mathbb{N}$ by $g_r(x) = i$ if $x \in [\bar{r}_i, \bar{r}_{i+1})$ (whereby $\bar{r}_0 := 0$). Then,

$$g_r \circ f_{\mathrm{val}}^{\mathrm{opt}}$$

is a value function with compressed 'bad' range and expanded 'good ' range, which we try to learn. Since this function always ranks the best value higher than the remaining ones, it yields the same optimum one-step-lookahead strategy as $f_{\mathrm{val}}^{\mathrm{opt}}$. We can thereby use a large tolerance ϵ for the approximation accuracy since this ranking has to be approximated only roughly.

One problem occurs in this approach: as training examples are added to the training set T, the number of examples with small value $g_r \circ f_{\mathrm{val}}^{\mathrm{opt}}$ increases rapidly, whereas good (improving) values of this function are rare. Thus the training set becomes unbalanced. To account for this fact, examples with large values $g_r \circ f_{\mathrm{val}}^{\mathrm{opt}}$ are added to the training set more often. This is done by including a fraction of the entire search path towards a frontier state to the training set, whereby the size of the fraction depends on the value of the function $f_{\mathrm{val}}^{\mathrm{opt}}$ on the path. This has the additional effect that also examples which represent schedules after few repair steps are added to the training set at the beginning of training and thus the search space is better covered.

Thus, the rout algorithm is changed in the following way to learn an approximation f_{dir} of the function $g_r \circ f_{\mathrm{val}}^{\mathrm{opt}}$: define, as beforehand, the one-step lookahead corresponding to f_{dir} by

$$f_{\mathrm{step}}(s) = \begin{cases} g_r(1/\mathrm{RDF}(s)) & \text{if } s \text{ is feasible} \\ \max_{s \succ s'} f_{\mathrm{dir}}(s') & \text{otherwise} \end{cases}$$

The rout algorithm becomes:

initialize f_{dir} and training set T;

repeat: hunt_frontier_state(s_0, (s_0));

add the returned pattern s and $P(s)$ patterns of the returned path to T;

retrain f_{dir};

Thereby, $P(s)$ is larger the better the value of s. The frontier-state search is as follows:

hunt_frontier_state(s, p)

{ repeat h times: for all $s \succ s'$:

generate a repair path p' from s' to a feasible schedule;

if $|f_{dir}(s'') - f_{\text{step}}(s'')| > \epsilon$ for some $s'' \in p'$:

let p'' be the subpath of p' from s to s'';

hunt_frontier_state(s'',(p, p'')) for the last such $s'' \in p'$; exit;

return the pattern $(s, f_{\text{step}}(s))$ and the path p; }

As beforehand, we add a consistency check before enlarging the training set T by new patterns to account for potential non-frontier states in T. Note that the used values of f_{step} are well defined in this procedure, since they only depend on the values $\bar{r}_i$ of already visited feasible schedules.

5 Experiments

For all experiments, we use the publicly available SVM-light software of Joachims for SVM training [18]. We use the ANOVA kernel with $\gamma = 0.1$ and $d = 3$ for the standard SVM and the direct heuristic focusing on the optima. For the ranking SVM, we restrict to the linear kernel, such that the problem can be transferred to an equivalent classification problem in the original space with a quadratic number of examples. The capacity of SVM training is set to $C = 0.05$. In all experiments, the function is initially trained on a set T of 200 frontier states obtained via search according to local RDF and random selection. Retraining of the value function takes place each time after a set of 20 new training points has been added to T. Thereby, a consistency check is done for old, possibly non-frontier patterns. For the direct SVM-method and ranking SVM, we set the tolerance $\epsilon = 0.01$. For the heuristic variant, the larger value $\epsilon = 0.05$ is chosen.

5.1 Small instances

We first randomly generated 10 instances with 20 jobs and 4 resources with the generator described in [24]. We compare results achieved with one-step lookahead and the simple initial greedy heuristic, to schedules achieved with one-step lookahead and the value function learned with rout and standard SVM, rout with ranking function, and rout with direct heuristic which focuses on optima. To show the capability

instance	1	2	3	4	5	6	7	8	9	10
greedy	2.39	2.86	2.97	2.65	2.68	3.10	2.49	2.40	2.82	2.71
greedy$_T$	2.71	2.97	3.40	3.09	3.01	3.46	2.80	2.92	3.87	2.81
rout$_{\mathrm{short}}$	**3.48**	3.22	3.03	3.09	**3.67**	3.85	2.85	2.98	**4.13**	2.99
rout	**3.48**	3.36	3.48	3.15	**3.67**	**4.11**	3.09	3.43	**4.13**	3.06
rank$_{\mathrm{short}}$	3.38	3.40	3.89	3.15	**3.67**	**4.11**	**3.14**	3.35	**4.13**	**3.21**
dir$_{\mathrm{short}}$	3.38	**3.51**	3.89	**3.27**	**3.67**	**4.11**	**3.14**	3.43	**4.13**	**3.21**
dir	**3.48**	**3.51**	**4.09**	**3.27**	**3.67**	**4.11**	**3.14**	**3.51**	**4.13**	**3.21**
imp.(%)	45.6	22.7	37.7	23.4	36.9	32.6	26.1	46.2	46.4	18.4

Table 1

Improvement obtained by reinforcement learning with different objectives compared to a simple greedy strategy on 10 different RCPSP instances. The respective best value is denoted in boldface. The last line denotes the percentage of improvement of the best schedule compared to the initial greedy solution.

of our approach to improve simple repair strategies even after short training we thereby compare the solution provided by the respective value function after training on the initial training set with 200 instances, and after training on a total number of 1000 training patterns explored by the reinforcement learner. We report the inverse of the achieved RDF, multiplied by the number of resources, 4, in Table 1. Thereby, 'greedy' refers to the initial greedy strategy based on the RDF, 'greedy$_T$' refers to the best value found in the initial training set, i.e. found by probabilistic iterative search guided by the initial greedy strategy, 'rout$_{\mathrm{short}}$' refers to the schedule found by the standard SVM-approach after training on the initial training set T, 'rout' refers to the schedule found by the standard SVM approach after 1000 training examples have been seen, 'rank$_{\mathrm{short}}$' is the result provided by the ranked SVM trained on the initial 200 instances, 'dir$_{\mathrm{short}}$' denotes the result of the only shortly trained SVM using the direct heuristic, and 'dir' refers to the same approach trained on 1000 training examples. We do not report results for the ranked SVM trained on more examples because of the increased time complexity of this approach: initial training of these instances takes about 6 min CPU-time on a Pentium III (700 MHz) for all three settings; training in combination with reinforcement learning for up to 1000 training examples takes about 2 hours CPU-time for the standard SVM and the direct heuristic. For the ranked SVM, this expands to about 12 hours, which is due to the increased complexity due to a quadratic training set, and the larger number of support vectors thus much slower training and evaluation of the SVM because of the simpler kernel (linear instead of ANOVA). Note that no backtracking takes place when the final schedules as reported in Tab. 1 are constructed; rather, the learned value function is used to directly transform the initial schedule s_0 with repair steps guided by one-step lookahead to a feasible schedule.

The obtained values as reported in Table 1 indicate, that even after a short training time, improved schedules can be found with the learned strategy. The strategy rout$_{\text{short}}$ improves compared to greedy$_T$ in all but two cases, and rank$_{\text{short}}$ and dir$_{\text{short}}$ improve for all instances compared to greedy$_T$. Hence the models generalize nicely also based on only few training examples. In addition, the solutions found after only shortly training the respective SVM often already yield near-optimum schedules for the tested instances. The direct heuristic dir which focuses on optima and which has been trained on 1000 patterns yields the best solution for all tested instances, and it also yields the best found solution when only trained on the initial set of 200 instances in seven of the ten cases. The improvement compared to the schedule obtained by the simple initial greedy strategy thereby ranges from 18.4% to 46.4%. In absolute numbers, the makespan for schedule number 6, for example, decreases from 48 time steps for s_{greedy} to 40 time steps for rout$_{\text{short}}$, and 37 time steps for rout.

We next investigate the generalization capability of the learned strategies. For this purpose, we randomly disrupt instance number 6 as follows: a precedence constraint is added or removed, a resource demand is increased or decreased by about 20% of the total range, a job duration is changed by about 30%, a resource availability by about 10%. Thus we obtain 30 similar instances. For these instances, we evaluate the quality of schedules obtained by one-step lookahead using the value functions trained on the *original* (i.e. not disrupted) instance. The achieved values are reported in Table 2. We report the results achieved with the strategy rout, rout$_{\text{short}}$, and dir. The generalization performance of the other strategies lies between these reported values. For comparison, we report the result obtained by the greedy strategy according to the RDF, and the best schedule obtained when probabilistic search including backtracking guided by the RDF is considered, visiting 200 feasible schedules.

In all but one case, dir yields the optimum value which greatly improves the original greedy strategy. Thereby, the achieved quality is comparable to the quality obtained for the original instance. In all but one case, already the only shortly trained standard SVM improves the initial greedy strategy. Note that the original instance is disrupted in this experiment such that the optimum schedules for the resulting instances are different from the original schedule and they yield different inputs to the value function, as can already be seen by the large variance of the quality of s_{greedy}. Hence this experiment indicates the generalization capability of learned strategies to new, though similar instances.

5.2 Instances with 30 jobs

For the next experiment, we consider 10 benchmark instances with 30 jobs and 4 machines, taken from [24]. We train standard SVM and the direct heuristic on these instances, as beforehand. Due to the high computational costs, we do not consider

instance	1	2	3	4	5	6	7	8	9	10
greedy	2.63	2.79	2.86	2.93	2.58	2.62	2.11	2.54	3.46	3.10
greedy$_T$	3.47	3.54	3.64	3.81	3.62	3.71	3.62	3.47	3.46	3.26
rout$_{\mathrm{short}}$	3.81	3.12	4.00	3.18	3.62	3.80	3.81	3.81	3.24	3.24
rout	3.63	**4.08**	**4.11**	**4.12**	**4.11**	**4.11**	**4.11**	**4.12**	4.11	**4.11**
dir	**3.91**	**4.08**	**4.11**	**4.12**	**4.11**	**4.11**	**4.11**	4.11	**4.12**	**4.11**
imp.(%)	48.7	46.2	43.7	40.6	59.3	56.9	94.8	61.8	19.1	32.6
instance	11	12	13	14	15	16	17	18	19	20
greedy	2.40	2.46	3.64	2.54	2.82	2.53	2.30	3.03	3.29	2.65
greedy$_T$	3.57	3.46	3.64	3.39	3.63	3.53	3.53	3.45	3.60	3.68
rout$_{\mathrm{short}}$	**3.84**	3.63	3.82	3.73	3.82	3.79	3.80	3.79	**4.09**	**4.08**
rout	**3.84**	**4.12**	**3.92**	3.73	3.82	**4.10**	**4.10**	**4.10**	**4.09**	**4.08**
dir	**3.84**	**4.12**	**3.92**	**3.92**	**4.12**	**4.10**	**4.10**	**4.10**	**4.09**	**4.08**
imp.(%)	60.0	67.5	7.7	54.3	46.1	62.0	78.3	35.3	24.3	54.0
instance	21	22	23	24	25	26	27	28	29	30
greedy	2.57	2.62	3.10	2.43	3.07	2.37	3.16	3.10	2.63	2.65
greedy$_T$	3.44	3.62	3.62	3.60	3.34	3.50	3.62	3.71	3.47	3.20
rout$_{\mathrm{short}}$	3.44	3.80	3.80	3.97	3.76	3.86	**4.11**	3.46	3.82	2.90
rout	2.66	**4.11**	**4.11**	**4.08**	3.34	**4.17**	**4.11**	**4.11**	3.82	3.07
dir	**3.62**	**4.11**	**4.11**	**4.08**	**4.06**	**4.17**	**4.11**	**4.11**	**4.02**	**3.57**
imp.(%)	40.9	56.9	32.6	68.0	32.2	76.0	30.0	32.6	52.9	34.7

Table 2
Generalization capability of the learned strategies. The quality of the solutions for 30 similar instances obtained by the value functions trained for the original instance are depicted. The respective best values are depicted in boldface. The last line shows the improvement of the best found strategy compared to the value of s_{greedy}.

the ranked SVM for these instances. In addition, the number of training examples is reduced to 800. Initial training here takes about 10 min CPU-time, and training including rout for up to 800 training examples takes about 4 hours on a Pentium III (700 MHz). The achieved results are depicted in Table 3. Thereby, the notation is as beforehand. In addition, we report the values for optimum solutions for these instances as given in [24].

As beforehand, learning the evaluation function allows to improve the quality of found solution by 18.5% to 58.6% compared to s_{greedy}. The heuristic dir which focuses on the optima rather than the exact RDF yields better solutions than the

instance	1	2	3	4	5	6	7	8	9	10
greedy	2.54	2.74	2.77	2.53	2.53	2.39	2.83	2.18	3.51	2.50
greedy$_T$	3.39	3.00	3.40	2.85	2.95	2.73	2.96	3.05	3.85	2.95
rout$_{\text{short}}$	2.58	2.41	2.90	2.75	3.02	2.21	3.14	2.36	**4.16**	3.01
rout	3.15	2.92	3.45	1.81	2.16	2.92	2.87	2.50	**4.16**	2.50
dir$_{\text{short}}$	**4.03**	3.43	3.49	**3.48**	3.45	3.25	3.67	**3.39**	**4.16**	**3.61**
dir	**4.03**	**3.48**	**3.70**	**3.48**	**3.52**	**3.37**	**3.87**	**3.39**	**4.16**	**3.61**
imp.(%)	58.6	25.6	33.6	37.5	39.1	41.0	36.7	55.5	18.5	44.4
opt	4.13	3.92	4.12	3.94	4.20	3.91	4.05	4.03	4.16	3.97

Table 3
Improvement obtained by reinforcement learning with different objectives compared to a simple greedy strategy on 10 different RCPSP instances with 30 jobs per instance, taken from [24]. The respective best found value is denoted in boldface. The last but one line denotes the improvement (in %) of the best found schedule compared to the greedy solution s_{greedy}. The last line denotes the values of optimum schedules for these instances as given in [24].

standard SVM combined with rout. For four cases, already the shortly trained value function dir$_{\text{short}}$ yields the best achieved value using one-step lookahead. Since optimum solutions for these instances are available, we can also access the absolute quality of the found schedules. In one case, the optimum could be found. For the other cases, the found solution is 0.1 to 0.68 apart from the optimum achievable value in terms of the scaled inverse RDF. However, these results are obtained *without* backtracking, i.e. using the learned value function to generate only one path in the search tree.

Also for these larger instances, the generalization ability has been tested. For this purpose, instance 9 has been disrupted as beforehand to get 30 similar instances for which the value function trained for the original instance has been applied. The mean quality obtained over 30 instances is 4.16 for dir and 3.76 for rout. For comparison, the mean value of s_{greedy} is 3.03, and the mean value of the greedy strategy together with limited backtracking (200 frontier states) is 3.66. Thus the strategy found by dir generalizes well to new instances and also rout allows improvements.

6 Conclusions

We have investigated the possibility to improve iterative repair strategies for the RCPSP by means of machine learning. We thereby restricted to acyclic repair steps with the benefit of priorly limited runtime and the possibility to use the rout reinforcement learning algorithm together with the SVM for value function approximation. Thereby, three different possibilities to approximate an adequate value function have been proposed, direct approximation of the optimum decision function based on the final RDF, an approach which only approximates the induced ranking, and a direct, faster heuristic which approximates the ranking at the observed best regions of the search space. The learned value functions could improve the initial greedy strategy for artificially generated instances and 10 benchmark instances. The learned strategies thereby transfer to new instances as tested exemplarily in experiments. Improved schedules could be found within this method although no backtracking has been done based on the approximated value function. Thereby, the direct heuristic yields the best overall performance in reasonable time. Standard SVM also improves the initial heuristic but it gives worse result than dir. The ranked SVM also improves compared to the standard SVM, but it considerably increases the computational effort because of a quadratic number of constraints for training.

However, the found strategies have not yet been capable of developing strategies which give the best possible solutions in a one-step lookahead search without backtracking. It is, of course, not clear whether this is possible at all, since the computation time of the used one-step lookahead strategies is linear. It can be expected that the results could be further improved if this simple search is substituted by more complex stochastic backtracking methods based on the learned value function such that the approaches might becomes competitive even for large-scale scheduling problems.

References

[1] R.Alvarez-Valdéz and J.M.Tamarit. Heuristic algorithms for resource-constrained project scheduling: a review and an empirical analysis. In R.Słowiński and J.Węglarz (eds.), Advances in project scheduling, pages 113-134, Elsevier, Amsterdam, 1996.

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

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

[4] C.E.Bell and J.Han. A new heuristic solution method in resource-constrained project scheduling. Naval Research Logistics, 38:315-331, 1991.

[5] D.P.Bertsekas and J.Tsitsiklis. Neuro-Dynamic Programming. Athena Scientific, 1996.

[6] J.Błażewicz, J.K.Lenstra, and A.H.G.Rinnoy Kan, Scheduling subject to resource constraints: classification and complexity. Discrete Applied Mathematics, 5:11-24, 1983.

[7] J.A.Boyan. Learning evaluation functions for global optimization. PhD thesis, Carnegie Mellon University, 1998.

[8] J.Boyan, W.Buntine, and A.Jagota (eds.). Statistical machine learning for large-scale optimization. Neural Computing Surveys 3(1):1-58, 2000.

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

[10] W.Brauer and G.Weiss. Multi-machine scheduling – a multi-agent learning approach. Proceedings of the 3rd Internatipnal Conference on Multi-Agent Systems, pages 42-48, 1998.

[11] P.Brucker and S.Knust. Lower bounds for resource-constrained project scheduling problems. European Journal of Operational Research, 149: 302-313, 2003.

[12] P.Brucker, S.Knust, A.Schoo, and O.Thiele. A branch and bound algorithm for the resource-constraint project scheduling problem. European Journal of Operational Research, 107:272-288, 1998.

[13] J.A.Carruthers and A.Battersby. Advances in critical path methods. Operational Research Quaterly, 17(4):359-380, 1966.

[14] C.Cortes and V.Vapnik. Support vector networks. Machine Learning, 20(3):273-297, 1995.

[15] E.Demeulemeester and W.Herroelen. New benchmark results for the resource-constraint project scheduling problem. Management Science, 43(11):1485-1492, 1997.

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

[17] S.Hartmann. A competitive genetic algorithm for resource constrained project scheduling, Technical Report 451, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 1997.

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

[19] T.Joachims. Optimizing search engines using clickthrough data. Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD), ACM, 2002.

[20] L.P.Kaelbling, M.L.Littmann, and A.W.Moore. Reinforcement learning: a survey. Journal of Artificial Intelligence Research, 4:237-285, 1996.

[21] R.Kolisch. Efficient priority rules for the resource-constrained project scheduling problem. Journal of Operations Management, 14(3):179-192, 1996.

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

[23] R.Kolisch and R.Padman. An integrated survey of project scheduling. Technical Report 463, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 1997.

[24] R.Kolisch and A.Sprecher, PSBLIB – a project scheduling library, European Journal of Operational Research 96, 205-219, 1996. See also http://www.bwl.uni-kiel.de/Prod/psplib/

[25] J.-K.Lee and Y.-D.Kim. Search heuristics for resource constraint project scheduling. Jorunal of Operational Research Society, 47:678-689, 1996.

[26] V.J.Leon and B.Ramamoorthy. Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling. OR Spectrum, 17(2/3):173-182, 1995.

[27] H.E.Mausser and S.R.Lawrence. Exploiting block structure to improve resource-constraint project schedules. Technical report, University of Colorado, Graduate School of Business Administration, 1995.

[28] A.McGovern, E.Moss, and A.G.Barto. Building a basic block instruction scheduler with reinforcement learning and rollouts. Machine learning, 49(2/3):141-160, 2002.

[29] A.Merke and R.Schoknecht. A necessary condition of convergence for reinforcement learning with function approximation. Proceedings of ICML, Morgan Kaufmann, 2002.

[30] D.Merkle, M.Middendorf, and H.Schmeck. Ant colony optimization for resource-constrained project scheduling. To appear in IEEE Transactions on Evolutionary Computation.

[31] 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.

[32] R.Moll, A.G.Barto, T.J.Perkins, and R.S.Sutton. Learning instance-independent value functions to enhance local search, NIPS'98, 1998.

[33] K.S.Naphade, S.D.Wu, amd R.H.Storer. Problem space search algorithms for recource-constraint project scheduling. Annals of Operations Research, 70:307-326, 1997.

[34] J.H.Patterson and G.W.Roth. Scheduling a project under multiple resource constraints: a zero-one approach. AIIE Transactions, 8:449-455, 1976.

[35] A.A.B.Pritsker, L.J.Watters, and P.M.Wolfe. Multiproject scheduling with limited resources: a zero-one programming approach. Management Science, 16:93-107, 1969.

[36] B.Pollack-Johnson. Hybrid structures and improving forecasting and scheduling in project management. Journal of Operations Management, 12:101-117, 1995.

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

[38] S.E.Sampson and E.N.Weiss. Local search techniques for the generalized resource constrained project scheduling problem. Naval Research Logistics, 40:665-675, 1993.

[39] A.Schirmer. Case-based reasoning and improved adaptive search for project scheduling. Manuskripte aus den Instituten für Betriebswirtschaftslehre 472, Universität Kiel, Germany, 1998.

[40] J.G.Schneider, J.A.Boyan, and A.W.Moore. Value function based production scheduling. ICML'98, 1998.

[41] J.G.Schneider, J.A.Boyan, and A.W.Moore. Stochastic production scheduling to meet demand forecast. Proceedings of the 37th IEEE Conference on Decision and Control, Tampa, Florida, U.S.A. 1998

[42] 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.

[43] L.Su, W.Buntine, A.R.Newton, and B.S.Peters. Learning as applied to stochastic optimization for standard cell placement. Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers& Processors, pages 622-627, 1998.

[44] R.Sutton and A.Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.

[45] 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.

[46] D.H.Wolpert, K.Tumer, and J.Frank. Using collective intelligence to route internet traffic. Advances in Neural Information Processing Systems - 11, MIT Press, 1999.

[47] W.Zhang and T.G.Dietterich, A reinforcement learning approach to job-shop scheduling, Proc.IJCAI, 1114-1120, 1995.