Improving iterative repair strategies for scheduling with the SVM

Kai Gersmann, Barbara Hammer

University of Osnabrück, Department of Mathematics/
Computer Science, Albrechtstraße 28, 49069 Osnabrück, Germany,
e-mail: {kai,hammer}@informatik.uni-osnabrueck.de

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. We here present an application of machine learning to adapt simple greedy strategies. The rout-algorithm of reinforcement learning is combined with the support vector machine (SVM) for value function approximation. 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.

1. 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 [11] 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, 8, 9]. The RCPSP constitutes a well-studied NP-hard scheduling problem with various benchmark instances and automatic problem generator [6]. It captures a general production scenario where jobs with precedence constraints are to be processed requiring several limited resources. Heuristic solutions of RCPSP are often based on local search algorithms or integer linear programming [1, 7]. 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.

We here consider the capability of reinforcement learning to improve a simple greedy strategy for general RCPSP instances. We formulate the 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 are capable of using the particularly efficient rout algorithm of reinforcement learning [3] and SVM [5] for approximating the value function, achieving a very sparse representation of the respective training set and good generalization ability. 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.

2. Resource constraint project 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 $i = 1, \dots, n$ 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 and can be characterized by the time points t_j at which the jobs j start, since we do not allow interruption of the jobs. A feasible schedule does neither violate precedence constraints, i.e. $t_i + d_i \leq t_j$ for all $i \to j$, nor resource restrictions, i.e. $\sum_{j:t\in[t_j,t_j+d_j)} m_{ji} \leq c_i$ for all time points t 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 schedule with minimum makespan, in general a NP-hard problem [6].

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, s_0, can obviously be computed in polynomial time. In [11] an objective called the resource dilation factor (RDF) is defined which is related to the makespan and takes resource violations into account for infeasible schedules: given a schedule s, RDF(s) is defined as TRUI(s)/TRUI(s_0) where the total resource utilization index TRUI(s) is defined as $\sum_{t,i} \max\{1, \sum_{j:t\in[t_j,t_j+d_j)} m_{ji}/c_i\}$ 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 TRUI(s) gives n times the makespan for feasible schedules. We can therefore alternatively minimize TRUI or the scaled version RDF instead of the makespan.

Starting from s_0, a feasible schedule can be obtained by repair steps. We consider only the following repairs: for the earliest time point violating a resource constraint, one job i is chosen and the job and its successors in the precedence graph reschedules. t_i is either increased by one or set to the earliest time point not violating resource constraints. All successors of i are scheduled at the earliest possible time for which the precedence constraints are fulfilled disregarding resource constraints. $s_i \succ s_j$ denotes the fact that s_j can be obtained from s_i by one repair step. Note that the precedence constraints are fulfilled in all schedules obtained in this way starting from s_0. A feasible schedule is achieved after a number of time steps bounded in the size of the instance as soon as all resource constrained are fulfilled. Moreover, an optimum feasible schedule can be obtained in principle on a path starting from s_0 if the repair steps are chosen appropriately. Of course, there cannot exist a simple and general strategy 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 $s_i \succ s_j$ such that the local RDF, i.e. the RDF of s_j, is optimum among all schedules directly connected to s_i. We refer to the first feasible schedule obtained in this way 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. Rout-algorithm and SVM

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. Note that this decision process is acyclic and those decisions are optimum which finally lead to a feasible schedule with minimum RDF. A simple greedy strategy substituting the final RDF on the search path by the local RDF after each repair step and deciding based on this one step lookahead has been proposed. Reinforcement learning offers a natural way to improve this strategy by problem dependent strategies learned from data [9]. They try to optimize the overall reward, i.e. minimize the final RDF or maximize its inverse. The rout algorithm has been proposed as a particularly efficient algorithm for acyclic domains [3].

Denote by S the set of schedules. Rout aims at learning the value function $\text{opt}_{RDF} : S \to \mathbb{R}$, $s \mapsto \max\{1/\text{RDF}(s') \mid s'$ is feasible and connected to s by a finite number of repair steps$\}$ for relevant regions of the search space. Choosing repair steps $s \succ s'$ based on the one step lookahead given by $\text{opt}_{RDF}(s')$ yields an optimum solution, an approximation of opt_{RDF} still likely achieves an improvement compared to the solution s_{greedy}. Obviously, $\text{opt}_{RDF}(s) = 1/\text{RDF}(s)$ for feasible schedules and for infeasible schedule s the Bellman equality holds: $\text{opt}_{RDF}(s) = \max\{\text{opt}_{RDF}(s') \mid s \succ s'\}$. Using this equality, rout tries to approximate the value function opt_{RDF} by a function f_{RDF} starting from the frontier states in the repair graph. Define $f_{\text{step}}(s) := 1/\text{RDF}(s)$ if s is feasible and $\max\{f_{\text{RDF}}(s') \mid s \succ s'\}$ otherwise. Rout consists in the following steps:

initialize f_{RDF} and training set T;
repeat: hunt_frontier_state(s_0);
  add the returned pattern to T and retrain f_{RDF};

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 $|f_{\text{RDF}}(s'') - f_{\text{step}}(s'')| > \epsilon$ for some $s'' \in p$:
   hunt_frontier_state(s'') for the last such $s'' \in p$; exit;
 return($s, f_{\text{step}}(s)$); }

Thereby, $h = 20$ and $\epsilon = 0.01$. We choose repair steps on p in $(*)$ based on a compromise between exploration and exploitation: choose the successor s'' with optimum RDF(s'') with probability p, the successor with optimum $f_{\text{RDF}}(s'')$ with probability $(1-p)^2$, and randomly with probability $p(1-p)$. p is linearly decreased from 1 to 0, hence starting with the search strategy based on local RDF and switching to the learned strategy f_{RDF} if enough exploration has been done. f_{RDF} is initialized randomly and initially trained on a set T of 200 frontier states obtained via search according to local RDF and random selection. Retraining of f_{RDF} only takes place each time after a set of 20 new training points has been added to T. Moreover, a SVM is used as approximator f_{RDF} in our case, which is already uniquely determined by a subset of T, the support vectors. We can therefore reduce T to the support vectors after training. We perform a consistency check when adding new training patterns to T, deleting old patterns if they almost coincide with new ones.

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 [4, 5, 10]. Since the SVM aims at minimizing the structural risk directly, we can expect very good generalization ability even for few training patterns. Moreover, the SVM is determined by the support vectors which constitute a sparse subset of T, hence allowing us to keep the size of T nearly constant. In order to use SVM, schedules are represented in a finite dimensional vector space adapting features as proposed in [11] to our purpose: the mean and variance of free resource capacities, mean and variance idle time slacks between consecutive jobs, the RDF, number of overallocated resources compared to this number in s_0, and various characteristics describing the number of time windows of constant job allocation which violate resource constraints. This representation allows to transfer the trained value function f_{RDF} to new instances even with a different 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 [5]. We could, of course, use alternative proposals of SVM for regression such as least squares SVM [10]. 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 $\sum_i(\alpha_i - \alpha_i^*) = 0$, $0 \leq \alpha_i, \alpha_i^* \leq C$ for all i

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 with components a_i resp. b_i. f_{RDF} can be derived from the dual variables as $f_{\mathrm{RDF}}(x) = \sum_i(\alpha_i - \alpha_i^*)k(x, x_i) + \alpha_0$, 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 $f_{\mathrm{RDF}}(x_i) = y_i \pm \epsilon$ for support vectors x_i with $\alpha_i, \alpha_i^* < C$. We have chosen C as 0.05, ϵ as 0.01 γ as 0.1 and d as 3.

4. Experiments

We randomly generated 10 instances with 20 jobs and 4 resources with the generator [6]. To show the capability of our approach to improve simple repair strategies even after short training we compare the solution provided by greedy search based on RDF, s_{greedy}; the optimum solution found when initializing the training set T with 200 pattern, $s_{\mathrm{greedy\_noise}}$; the solution obtained with one-step lookahead based on the SVM trained on the initial set T, $s_{\mathrm{rout\_short}}$; and the solution based on the SVM trained on 1000 training pattern, s_{rout}. The inverse of the achieved RDF, multiplied by the number of resources, 4, is depicted in Fig. 1 and clearly indicates that even after short training time, improved schedules can be found with the learned strategy. s_{rout} often already constitutes a near-optimum schedule for the tested instances and provides in all but 1 cases an improved schedule. For all but 2 cases already the initially trained SVM shows better solutions than the optimum solution found when generating the initial training set, i.e. the generalization ability of the SVM allows to go

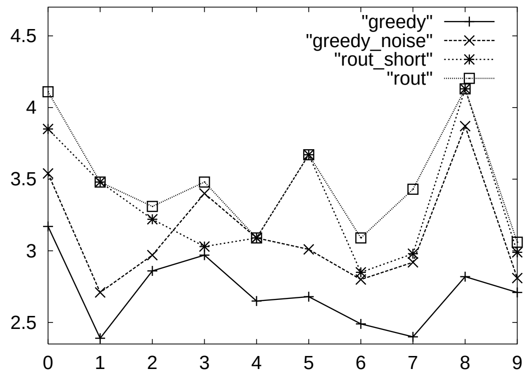

Figure 1: Improvement obtained by reinforcement learning compared to a simple (iterated) greedy strategy on 20 different RCPSP instances.

beyond information provided in the initial training set. In absolute numbers, the makespan for the leftmost schedule decreases from 48 time steps in s_{greedy} to 43 time steps in $s_{\text{greedy\_noise}}$, 40 time steps in $s_{\text{rout\_short}}$, and 37 time steps in s_{rout}. Note that no backtracking has been done for strategies based on f_{RDF}.

To investigate the generalization capability of the learned strategies, we randomly disrupt the leftmost instance 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 which s_{greedy} and $s_{\text{greedy\_noise}}$ are depicted in Fig. 2. In addition, the value function trained for the *original* instance on the initial set T and 1000 pattern, respectively, has been used for one-step lookahead for the disrupted instances, yielding schedules $s_{\text{rout\_short}}$ and s_{rout}. The benefit of this strategy is depicted in Fig. 2, the leftmost row presenting the values for the original instance used for training. 22 out of 30 instances are improved by the only shortly for the original instance trained version, and 21 out of 30 instances achieve a final schedule which quality is comparable to the final schedule obtained in the original problem. In only one case, f_{RDF} leads to a final schedule which does not significantly improve the initial schedule s_{greedy}. 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, as can already be seen by the large variance of the quality of s_{greedy}. Hence this experiment clearly points out the generalization capability of learned strategies to new, though similar instances.

5. Conclusion

We have demonstrated 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 efficient rout reinforcement learning algorithm together with the SVM for value function approximation. The approach could improve the initial greedy

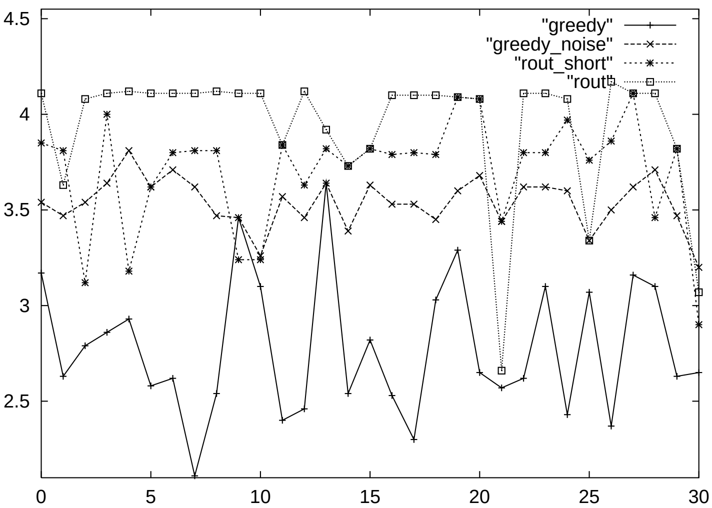

Figure 2: Improvement achieved by the strategy trained on the leftmost instance for 30 similar instances; f_{RDF} generalizes to more than 2/3 of the cases.

algorithm for artificially generated instances and the learned strategies transfer to new instances as demonstrated by experiments. Note that good schedules could be found within this method although no backtracking has been done based on the approximated value function. It can be expected that the results could be further improved if one-step lookahead based on the learned function f_{RDF} is combined with stochastic backtracking methods such that the approach becomes competitive even for large-scale scheduling problems.

References

[1] 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.

[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] B.Hammer and K.Gersmann, A note on the universal approximation capability of SVMs, to appear in Neural Processing Letters.

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

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

[7] 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.

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

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

[10] 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.

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