Reinforcement Learning for Schedule Optimization*

Nikolai Antonov[13], Jan Hůla[12], Mikoláš Janota[1], and Přemysl Šůcha[1]

[1] Czech Technical University in Prague, Prague, Czech Republic
[2] University of Ostrava, Ostrava, Czech Republic
[3] Kazan Federal University, Kazan, Russia

Problem formulation and related work. In this paper we use machine learning to optimize a specific problem in integer linear algebra, which is practically motivated as job scheduling. Let us have a machine (system) capable of doing some work divided into a sequence of jobs. The machine follows three basic assumptions. First, it can do only one job a time. Secondly, a started job cannot be interrupted: a job must be completed before starting a next one. Third, the machine cannot idle: having finished one job, it immediately moves onto the next one until all the jobs assigned to the machine are finished. We are given a set of jobs $N = \{1, 2, ..., n\}$ with *processing times* p_i and *due dates* d_i for all $i \in N$. We assume that p_i and d_i are positive integers and $p_i \leq d_i$ for all $i \in N$. Additionally, each job has a *weight* (or *cost*), which is a positive integer w_i, $i \in N$ representing how valuable a particular job is. Assume that all the jobs are available form the very beginning (time moment 0) and executed one by one in the order specified by a permutation P of N. Let C_i^P denote the completion time of i-th job executed according to permutation P and define a set $S = \{i \in N \mid C_i^P \leq d_i\}$. The goal is to find a permutation P^* maximizing the weighted number of jobs that will be completed no later than the specified due date, i.e. maximize $f(P) = \sum_{i \in S} w_i$. We formulate the problem in satisfiability modulo theories (SMT). We want to find an integer vector

$$(s_1, s_2, ..., s_n) \geq \mathbf{0}$$

maximizing

$$\sum \sigma(C_i^P, d_i) w_i, \text{ where } \sigma(x, y) = 1 \text{ if } x \leq y \text{ else } 0$$

subject to

$$i < j \implies (s_i + p_i \leq s_j) \vee (s_j + p_j \leq s_i), \ i \in N, \ j \in N$$

The formulation does not explicitly specify that there must not be idling time, because at the post-processing stage any solution can be easily adjusted to eliminate any idling. In this work, we use reinforcement learning (RL) to solve this maximization problem, without guaranteeing optimality—approximate optima are also practically interesting. We remark that a decision version of the problem is obtained by bounding the objective function by some integer K.

The problem is proven to be NP-hard [4]—Knapsack is a special case when all jobs have the same due date. Due to its practical importance, the problem has been studied extensively in scheduling and OR communities: Potts and Van Wassenhove [6] gave a branch and bound algorithm for solving instances with up to 1,000 jobs; M'Hallah and Bulfin [5] propose an exact algorithm capable of handling instances with up to 2,500 jobs; Baptiste et al. [1] developed an algorithm solving up to 50000 jobs instances of particular type; Hejl et al. [3] investigated strongly-correlated instances and achieved a progress of solving 5000 jobs within one hour.

To the best of our knowledge, the considered problem was not studied in the ML community. However, number of combinatorial optimization problems tackled by reinforcement learning and other ML approaches is growing every year; we refer the reader to a survey by Bengio et al. [2].

*Results of this project No. LL1902 were supported by the Ministry of Education, Youth and Sports within the programme ERC CZ

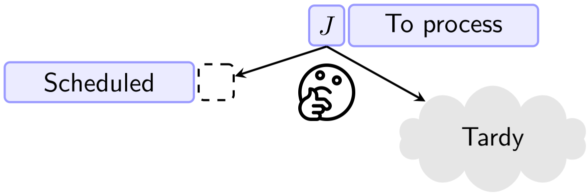

Figure 1: Decision-making of the RL agent for job J

Approach. We solve the considered problem using *deep reinforcement learning* [8, 7]. A sketch of the approach is provided in the Figure 1. Initially, all the jobs are sorted in ascending order by due date. At each moment of time, the agent observes a set of jobs that have not been completed yet and one of these jobs that the agent has to decide about (the unprocessed job with the earliest due date). The featurization of the unprocessed jobs is based on the distribution of their weights/proc. time/due date (represented as histograms). The agent has two possible actions: (1) schedule the first unprocessed job immediately and therefore it will be *on time*; (2) mark the job as *tardy* and move it to the end of schedule. One step of the agent corresponds to a decision regarding one job from a given instance. During the training phase, the agent receives a reward whenever the decision is right, i.e. if the job turned out to be the same (on time or late) as the agent predicted it to be in the optimal permutation that we have as a label. During the validation phase, the agent is only rewarded at the very end. The reward is equal to the ratio of the cost obtained by following the policy to the cost of the optimal solution.

Evaluation and conclusions. To make a fair comparison with actual results, we generate data according to the algorithm presented in [1] and [3]. Weights and durations are random integers from $[1; 100]$ and every due date is random integer from $[0.3S; 0.7S]$, where $S = \sum_{i \in N} p_i$. We compare to greedy heuristics MAX COST, MAX COST/DUR and MAX COST/DUE, which process the jobs in ascending order based on w_i, ratios $\frac{w_i}{p_i}$, and $\frac{w_i}{d_i}$, respectively. Terms $\mu(n)$ and $\sigma(n)$ stand for mean and standard deviation of optimality gap, obtained on the instances with n jobs. An optimality gap is defined as $\frac{v^* - v}{v^*}$, where v^* is the optimal value of the instance and v is the cost obtained by following the policy. The results show that our approach achieves much lower optimality gap than any of the greedy-heuristic approaches. This indicates that reinforcement learning is a viable approach to optimization of the linear integer algebra problems studied in this paper. We believe that this work would inspire further research on the use of reinforcement learning on more general problems or on probabilistic decision procedures.

	$\mu(100)$	$\sigma(100)$	$\mu(250)$	$\sigma(250)$
MAX COST	0.14	0.03	0.14	0.02
MAX COST/DUR	0.09	0.02	0.09	0.01
MAX COST/DUE	0.09	0.02	0.09	0.01
DRL model	0.065	0.027	0.015	0.01

	$\mu(500)$	$\sigma(500)$	$\mu(1000)$	$\sigma(1000)$
MAX COST	0.14	0.01	0.14	0.01
MAX COST/DUR	0.08	0.01	0.08	0.01
MAX COST/DUE	0.09	0.01	0.09	0.01
DRL model	0.017	0.01	0.009	0.006

References

[1] Philippe Baptiste, Federico Della Croce, Andrea Grosso, and Vincent T'kindt. Sequencing a single machine with due dates and deadlines: an ILP-based approach to solve very large instances. *J. Sched.*, 13(1):39–47, 2010.

[2] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *Eur. J. Oper. Res.*, 290(2):405–421, 2021.

[3] Lukás Hejl, Premysl Sucha, Antonín Novák, and Zdenek Hanzálek. Minimizing the weighted number of tardy jobs on a single machine: Strongly correlated instances. *Eur. J. Oper. Res.*, 298(2):413–424, 2022.

[4] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[5] Rym M'Hallah and R. L. Bulfin. Minimizing the weighted number of tardy jobs on a single machine. *Eur. J. Oper. Res.*, 145(1):45–56, 2003.

[6] Chris N. Potts and Luk N. Van Wassenhove. A branch and bound algorithm for the total weighted tardiness problem. *Oper. Res.*, 33(2):363–377, 1985.

[7] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning.* Adaptive Computation and Machine Learning series. Bradford Books, Cambridge, MA, 2 edition, November 2018.