Reinforcement Learning of Dispatching Strategies for Large-Scale Industrial Scheduling

Pierre Tassel,[1] Benjamin Kovács,[1] Martin Gebser,[1 2] Konstantin Schekotihin,[1]
Wolfgang Kohlenbrein,[3] Philipp Schrott-Kostwein [3]
[1] University of Klagenfurt, Austria
[2] Graz University of Technology, Austria
[3] Kostwein Holding GmbH, Austria

Abstract

Scheduling is an important problem for many applications, including manufacturing, transportation, or cloud computing. Unfortunately, most of the scheduling problems occurring in practice are intractable and, therefore, solving large industrial instances is very time-consuming. Heuristic-based dispatching methods can compute schedules in an acceptable time, but the construction of a heuristic providing satisfactory solution quality is a tedious process. This work introduces a method to automatically learn dispatching strategies from just a few training instances using reinforcement learning. Evaluation results obtained on real-world, large-scale instances of a resource-constrained project scheduling problem taken from the literature show that the learned dispatching heuristic generalizes to unseen instances and produces high-quality schedules within seconds. As a result, our approach significantly outperforms state-of-the-art combinatorial optimization techniques in terms of solution quality and computation time.

Introduction

Many problems in various application domains, such as transportation, manufacturing, energy, or finance, require solving scheduling problems in an acceptable time. Unfortunately, most of these problems are proven to be *NP-complete* (Garey, Johnson, and Sethi 1976) or *NP-hard* (Sotskov and Shakhlevich 1995), i.e., no algorithms for efficient schedule computation are known. Therefore, domain-independent optimization techniques, such as Mixed Integer Programming (MIP) (Bénichou et al. 1971), Constraint Programming (CP) (Rossi, van Beek, and Walsh 2006), or Answer Set Programming (ASP) (Gebser et al. 2012), often cannot find suitable schedules for industrial-size problem instances within given time constraints. In such cases, researchers aim to develop fast (meta-)heuristic algorithms that compute satisfactory, but not necessarily optimal, schedules in a runtime appropriate for practical scenarios.

Over the past decades, various approximate scheduling methods have been suggested in the literature (Vaessens, Aarts, and Lenstra 1996; Potts and Strusevich 2009; Çalis and Bulkan 2015). Among them, dispatching rules constitute one of the most popular approaches (Grabot and Geneste 1994). Every such rule is a handcrafted heuristic designed by experts using past problem instances and personal experience in solving scheduling problems. Examples include the First-In-First-Out (FIFO) rule, which schedules jobs according to the time they arrive in a queue, or the Most-Total-Work-Remaining (MTWR) rule, prioritizing jobs with the most amount of processing still to be done (Blackstone, Phillips, and Hogg 1982).

Although algorithms based on dispatching rules have shown good runtime results when applied to real-world scheduling problems, the quality of obtained solutions is often considerably off from optimal schedules. In addition, these methods are not robust and require manual parametrization for a particular scheduling problem, which can be tedious and error-prone. Machine Learning (ML) arises as a natural tool for tuning and/or discovering such heuristics. The reutilization of experience gained by solving previous problem scenarios on new unseen instances has been a topic of significant interest in recent years, with a range of approaches suggested in the literature (Bengio, Lodi, and Prouvost 2021).

Contributions. In this paper, we propose a new *Reinforcement Learning* (RL) formulation for scheduling problems and an approach for training neural networks to learn dispatching strategies. Our contributions can be summarized as follows:

- **Efficient MDP Formulation.** We introduce an efficient Markov Decision Process (MDP) formulation for scheduling problems, where the state-transition function considers sequences rather than a fixed number of actions. This formulation is closely related to dispatching heuristics and allows us to represent the scheduling process in terms of single-agent decision-making. Moreover, the number of state transitions stays low and learned policies do not depend on the number of jobs or tasks in problem instances.
- **RL Training Approach.** Taking advantage of the specificities of our MDP formulation, we design an RL approach capable of training order- and size-invariant neural networks on instances of diverse scale. Our training method does not require a continuous reward function, thus eliminating the need for reward-shaping techniques (Laud 2004) and circumventing the injection of possibly biased prior knowledge into the learning environment.
- **Beam-search Solution Generation.** Finding action sequences minimizing the solution cost of an instance based on the generated probability distribution of the neural net-

work is hard. While simple and fast, greedy search can often lead to sub-optimal outcomes when applied to neural action selection. On the contrary, exhaustive search gets intractable very soon due to the size of the search space. Our use of beam search to generate solutions from a neural network, thus, constitutes a suitable compromise between solution quality and computational cost.

- **Empirical Evaluation.** We evaluate our method on large-scale, industrial instances of a *Resource-Constrained Project Scheduling* (RCPS) problem (Hartmann and Briskorn 2010) from the literature (Kovács et al. 2021), comparing our approach to previously developed CP and MIP models. In particular, we demonstrate the ability of our method to learn dispatching heuristics that generalize well to previously unseen RCPS instances, thus reusing the experience gained by solving a few training scenarios.

The paper is organized as follows. Section describes basic concepts of scheduling problems, introduces the RCPS problem, as well as state-of-the-art algorithms and heuristics to solve these problems. Section presents our RL approach and its implementation. Results of an extensive evaluation on an industrial scheduling problem are provided in Section . Section concludes the paper and discusses future work.

Background and Problem Statement

Scheduling problems appear in different domains and are ubiquitous in many industries such as manufacturing or transportation (Rinnooy Kan 2012). From the earliest approaches, most research focused on the assignment of jobs to machines, where each job has an arrival time, a due date, and a penalty associated with delays (Muth and Thompson 1963). Since then, practical applications have led to various scheduling problem definitions focusing on specific domain requirements (Pinedo 2016).

RCPS Problem. Different variants of the RCPS problem (Hartmann and Briskorn 2010; Cavalcante, Cardonha, and Herrmann 2013) deal with scenarios where limited resources ought to be utilized in the best possible way to accomplish the tasks of given jobs. A typical RCPS instance comprises a set J of jobs and renewable resources M (e.g., machines or personnel), which are represented by finite subsets of $\mathbb{N}^+$. Each job $j \in J$ corresponds to a (partially) ordered set of tasks (or operations) T_j with a deadline $d_j \in \mathbb{N}^+$ and an importance weight $w_j \in \mathbb{N}$. Every task $\tau \in T_j$ has a processing time $p_\tau \in \mathbb{N}^+$ and requires at least one resource in M. The execution of tasks is non-preemptive, i.e., a task cannot be interrupted once started. To respect the order of tasks in a job j, any task $\tau \in T_j$ can start only when all preceding tasks in T_j have been finished. Additional requirements on the starting time might include *coupling*, when τ must start immediately after all preceding tasks are finished, or *buffering*, enforcing a time buffer $b_\tau \in \mathbb{N}$ between τ and preceding tasks. The resources $m \in M$ are characterized by their capacities $q_{m,k} \in \mathbb{Q}^+ \cup \{0\}$ for time points $1 \leq k \leq h$. Note that $q_{m,k} = 0$ signals the unavailability of m at time k, e.g., due to maintenance or days off. The objective is to find a schedule that minimizes the total tardiness, i.e., the weighted sum of delays l_j over all jobs $j \in J$:

$$\sum_{j \in J, l_j > 0} w_j \cdot \big(\omega \cdot l_j + \omega'\big) \tag{1}$$

where coefficients ω and ω' indicate per time point and job-wise delay penalty, respectively.

Previous Approaches. The RCPS problem is *NP-hard* in the strong sense (Blazewicz, Lenstra, and Rinnooy Kan 1983) and has been extensively studied in the literature (Herroelen, De Reyck, and Demeulemeester 1998; Pellerin, Perrier, and Berthaut 2020). Given the complexity, exact methods like MIP or CP come to their limits as the size of problem instances grows. Over the past decades, considerable research focused on methods to compute high-quality solutions within an acceptable time. Examples include approximate MIP or CP models exploiting specific problem properties and incomplete approaches like (meta-)heuristic search.

Heuristic Approaches. A greedy algorithm can utilize various heuristics to construct a solution for a problem instance. Among these heuristics, *dispatching rules* that express expert knowledge in the form of "if-then" conditions are prevalent. In general, such methods determine a priority for each job based on its features and resource states at every time point when scheduling decisions must be made. The features may include the time left until a job's deadline, the number of yet unaccomplished tasks, or the importance weight of a job. Then, the greedy algorithm schedules tasks of highest-priority jobs to free machines such that problem constraints, e.g., resource capacities, are satisfied.

While handcrafted heuristics have an outstanding runtime performance and lead to solutions for very large problem instances, they incur several drawbacks: *(i)* the prioritization of jobs relies on the personal experience of domain experts and particular special cases encountered in the past; *(ii)* experts must be aware of features relevant for the job prioritization and combine them properly in a heuristic function; and *(iii)* even slightest modifications of the original scheduling problem or instance characteristics might cause a significant reduction of solution quality or even make a heuristic method unable to compute a feasible solution.

Metaheuristic Approaches. These techniques use sophisticated combinations of heuristics to explore the most promising regions of the search space thoroughly. Unlike complete optimization algorithms, metaheuristics do not guarantee the computation of a globally optimal solution when given unlimited runtime. Nevertheless, by traversing a range of feasible solutions, metaheuristics can often produce high-quality solutions with less computational effort than complete methods (Blum and Roli 2003). Moreover, in many cases, metaheuristic approaches provide better solutions than greedy heuristics, requiring just a modest increase in computation time. Most metaheuristics implement some form of stochastic optimization, incorporating random decisions to circumvent overly biased exploration of the search space and to escape local optima (Bianchi et al. 2009). The metaheuristic methods utilized for scheduling include simulated annealing, tabu search, artificial immune systems, bee and ant colony optimization, genetic algorithms, particle swarm optimization, and scatter search (Vaessens, Aarts, and Lenstra 1996; Ishibuchi and

Murata 1998; Ahire et al. 2000; Gao et al. 2019). A common issue of metaheuristics is lack of robustness, i.e., they can be successful on some problem instances but fail to find satisfactory solutions or take long computation time on other instances of the same problem (Chopard and Tomassini 2018).

RL Approaches. Recently, RL methods attracted researchers as an approach to overcome the issues of scheduling methods discussed above. Several works study variants of the classic job-shop scheduling problem (Zhang et al. 2020; Liu, Chang, and Tseng 2020; Han and Yang 2020; Tassel, Gebser, and Schekotihin 2021). While these approaches provide better results than handcrafted dispatching rules, they have only been tested on small-scale benchmark instances (Taillard 1993; Demirkol, Mehta, and Uzsoy 1998) with less than 100 jobs and 20 machines, so that they do not take the challenges faced in industrial settings into account. Another issue of such RL-based scheduling approaches is that they rely on reward shaping to deal with the credit assignment problem (Minsky 1961), faced due to the optimization of episodic job completion objectives.

For the RCPS problem, which is more general than job-shop scheduling, Sung, Choi, and Nielsen (2020) proposed an RL approach that associates actions with jobs. Consequently, the method is not size-invariant and requires re-training for instances of different sizes. Empirical improvements compared to greedy heuristics are thus limited to the tested benchmark instances of small size, comprising at most 20 projects and 90 operations at maximum.

Moreover, researchers applied RL methods to improve the performance of complete optimization approaches like MIP and CP. Modern MIP solvers incorporate heuristic reasoning techniques, such as cutting planes (Marchand et al. 2002), branch-and-bound (Gomory 2010), branch-and-cut (Padberg and Rinaldi 1991), or metaheuristics (Glover 1990), which can be configured and tuned by means of RL methods (Tang, Agrawal, and Faenza 2020; Nair et al. 2020; Prouvost et al. 2020; Ding et al. 2020). Similar domain-independent tuning efforts have been undertaken for CP solvers as well (Cappart et al. 2021; Chalumeau et al. 2021). However, automatic algorithm configuration cannot circumvent limitations of exact optimization approaches due to inherent problem complexity.

DispatchR Method

This section formulates dispatching decisions in terms of an RL problem, presents a respective deep RL-based architecture, specifies training and application methods.

Dispatching as an RL Problem

An RL problem is specified in the form of an MDP: at any time point $t \geq 0$, an agent is in a *state* $s_t \in \mathcal{S}$, takes an *action* $a_t \in \mathcal{A}$, receives an instantaneous *reward* $r_t \in \mathbb{R}$, and transitions to the next state $s_{t+1} \sim p(\,\cdot \mid s_t, a_t)$. A *policy* $\pi : \mathcal{S} \mapsto \mathcal{P}(\mathcal{A})$ maps any state s_t to an action $\pi(a_t \mid s_t)$ with the intent of maximizing the expected cumulative reward over a time horizon. We consider parameterized policies π_θ and seek to solve the optimization problem of finding $\theta^* = \arg\max_\theta J(\pi_\theta)$, the *optimal parameters* for our policy. In the following, we formulate dispatching decisions on jobs' operations to be scheduled in terms of an MDP. We specify the state space $\mathcal{S}$, action space $\mathcal{A}$, reward r_t, and the transition function $s_{t+1} \sim p(\,\cdot \mid s_t, a_t)$.

State Space $\mathcal{S}$. A state $s_t \in \mathcal{S}$ is given by the set $\mathcal{L}^t$ of *legal jobs* at time point t, i.e., jobs whose current tasks can be scheduled to free resources, along with their features. For each legal job j, we consider the following features: *(1)* remaining time until the deadline d_j; *(2)* number of remaining tasks until completing the job j; *(3)* importance weight $w_j \in \mathbb{N}$; *(4)* sum of buffer times $b_\tau \in \mathbb{N}$ over remaining tasks $\tau \in T_j$; *(5)* number of remaining coupled tasks in T_j; and *(6)* processing time $p_\tau \in \mathbb{N}^+$ of the current task $\tau \in T_j$.

Action Space $\mathcal{A}$. The set $\mathcal{A}$ of *available actions* at time point t consists of the legal jobs in $\mathcal{L}^t$, i.e., $\mathcal{A} = \mathcal{L}^t$. Hence, the action space is discrete, and the cardinality of $\mathcal{A}$ may vary from one time point to another. The latter condition is different from conventional action spaces, which have a fixed number of actions for all possible states.

Reward r_t. Due to the specificities of our training approach, which employs a derivative-free optimization method, we do not require a continuous and differentiable reward function. As a consequence, our method can directly use the original objective function in Equation (1), even though it is sparse and yields feedback only when jobs are completed.

State Transition. Given a state $s_t \in \mathcal{S}$ along with an ordered set $\mathcal{O}^t$ over the actions in $\mathcal{A} = \mathcal{L}^t$, the new state s_{t+1} is determined as follows. For each job j in the order given by $\mathcal{O}^t$, allocate j if its resources are (still) free, and move on to the next job in $\mathcal{O}^t$ otherwise. If the agent allocates a job j, the required resources are occupied, and other jobs needing them have to wait for the processing time p_τ until completion of the scheduled task $\tau \in T_j$. After all legal jobs have been traversed, the environment increments the time point by 1 and determines the legal jobs whose current tasks are executable at s_{t+1} along with their features. Note that, even though all jobs in $\mathcal{O}^t$ are initially legal at s_t, their successive allocation may occupy the resources also required for later jobs in $\mathcal{O}^t$.

Network Architecture

Before feeding state representations to a neural network, we rescale observed job features by applying a min-max normalization scaler and then proceed as follows.

Position-wise Feed-Forward Network. Our policy learner consists of a simple, *fully connected feed-forward network* composed of three linear transformations with a *tanh* activation in between:

$$\mathrm{FFN}(x) = \tanh(\tanh(xW_1 + b_1)W_2 + b_2)W_3 + b_3$$

Similar to the transformer architecture (Vaswani et al. 2017), we apply the neural network to each legal job separately. The inner layers have dimensionality $d_{ff} = 32$, while the network outputs a scalar representing the estimated job priority. We transform the obtained priority vector into a probability vector by stacking the estimated priorities of jobs and applying the *Softmax* function to the resulting vector. Having a probability vector rather than a priority vector facilitates

Algorithm 1: Policy Rollout

```
 1: Input: policy network parameters $\theta$ and set $J$ of jobs.
 2: Initialize time point $t = 0$
 3: while unfinished job $j \in J$ exist do
 4:     Filter all legal jobs $\mathcal{L}^t$
 5:     Compute features of the jobs in $\mathcal{L}^t$
 6:     Sample without replacement an ordered set $\mathcal{O}^t$ of actions using the distribution given by policy $\pi_\theta(\,\cdot \mid s_t)$
 7:     for all $j \in \mathcal{O}^t$ do
 8:         if $j$ can be allocated then
 9:             Allocate $j$ at time point $t$
10:         end if
11:     end for
12:     $t \leftarrow t + 1$
13: end while
```

exploration during the learning phase. In particular, we determine the allocation order by sampling without replacement from this probability vector, thus avoiding tie-breaking in situations when two jobs have the same estimated priority.

Policy Rollout. In Algorithm 1, we put everything together and describe one policy rollout. The resulting environment comes close to classic static dispatching heuristics, except that the policy network dynamically computes the job dispatching order. Moreover, the enforcement of coupling and buffer constraints on the ordered set $\mathcal{O}^t$ is accomplished by a custom propagator built-in to the environment.

Training Method

Objective functions of scheduling problems are defined over discrete sets of possible job allocations and result in learning environments with non-smooth objectives. The latter are not differentiable and, therefore, unsuitable for training RL agents by traditional methods backpropagating gradients. In our approach, we overcome this issue by using a combination of two methods: estimation of a natural gradient using the Parallelized Evolution Strategies (PES) algorithm by Salimans et al. (2017) and a modern gradient-based optimization algorithm to update the mean vector of the policy network weight distribution.

Evolution Strategies (ES) is a family of black-box optimization methods inspired by natural evolution that, in every iteration, generate a set of parameter vectors, evaluate their fitness using the objective function, and retain the "fittest" ones. The generation procedure often uses d-dimensional multivariate Gaussian, which mean vector and covariance matrix represent candidate solution center and mutation matrix of parameter vectors, respectively. ES methods optimize these distribution parameters using plain gradients, which often causes issues with slow convergence of ES algorithms (Wierstra et al. 2014). To solve this problem, Natural Evolution Strategies (NES) (Wierstra et al. 2014) algorithms use *natural gradient* (Amari 1998), which uses a more "natural" Kullback-Leibler divergence (Kullback and Leibler 1951) to measure the distance between distributions instead of the Euclidean distance used by plain gradients.

When applied to the training of a neural network $F(\theta)$, NES methods operate on the distribution of parameters p_θ and not on the network parameters θ themselves. In this case, the goal is to minimize the expected value $\mathbb{E}_{\theta' \sim p} F(\theta')$ by searching over the parameter space of p_θ with stochastic gradient optimization. Thus, the variant of PES suggested by Salimans et al. (2017) assumes that p_θ is a multivariate Gaussian with mean vector θ and fixed covariance $\sigma^2 I$. In this case, the gradient estimator can be rewritten using the log-likelihood trick in terms of the parameter vector θ, leading to an estimator of the form:

$$\nabla_\theta F(\pi_\theta) \approx \tfrac{1}{\sigma \cdot N} \textstyle\sum_{i=1}^{N} F(\pi_{\theta'_i}) \epsilon_i \qquad (2)$$

where $\epsilon_i \sim \mathcal{N}(0, \mathbb{I})$ is sampled from a multivariate Gaussian, $\theta'_i = \theta + \sigma\epsilon_i$, $F(\pi_{\theta'_i})$ is estimated using a single trajectory, and $\sigma > 0$ is a fixed constant. This gradient estimator is used to evaluate the performance of each individual neural network of the population on the current state of a scheduling instance. Roughly, PES algorithm used in DispatchR can be summarized as follows:

0. Initialize the current policy parameter vector θ.
1. Sample N neighboring solutions θ'_i from a normal distribution centered around θ with a standard deviation σ.
2. Evaluate each neighbor θ'_i, using the tardiness function in Equation (1) and estimate a natural gradient using Equation (2).
3. Provide the estimated gradient to an optimization algorithm that updates the policy parameter vector θ.
4. Repeat from step 1 until convergence.

Because PES works with the function evaluations, there is no need to rely on a continuous reward function. Therefore, we can use the unbiased objective function of the original problem definition. Moreover, Metz et al. (2021) recently pointed out that ES may be a better alternative for training RL agents than gradient descent, providing better gradient estimation when the policy function is low-dimensional.

Schedule Computation

Computation of schedules using a trained RL agent only, i.e., as a heuristic for a greedy best-first strategy, might be too narrow since an agent selects actions one by one or sequence by sequence, as in our case. Greedy search methods, which iteratively select the action with the highest probability according to the probability distribution of the policy network, might ensure a short computation time. However, its application often leads to solutions of a lower quality (Stahlberg and Byrne 2019). Beam search provides a good balance between greedy best-first and complete strategies and, therefore, it is the preferred method for decoding auto-regressive machine translation models (Meister, Cotterell, and Vieira 2020). In particular, beam search maintains k beams of possible solutions, which correspond to a form of pruned breadth-first search where the breadth is limited to $k \in \mathbb{Z}^+$ hypotheses.

Our beam search algorithm, illustrated in Figure 1, has two phases: the greedy and the exploration phase. In the first phase, the algorithm starts with the initial state s_0 and greedily constructs a solution by sampling action vectors a_t from the probability distribution $\pi(a_t \mid s_t)$, where $0 \leq$

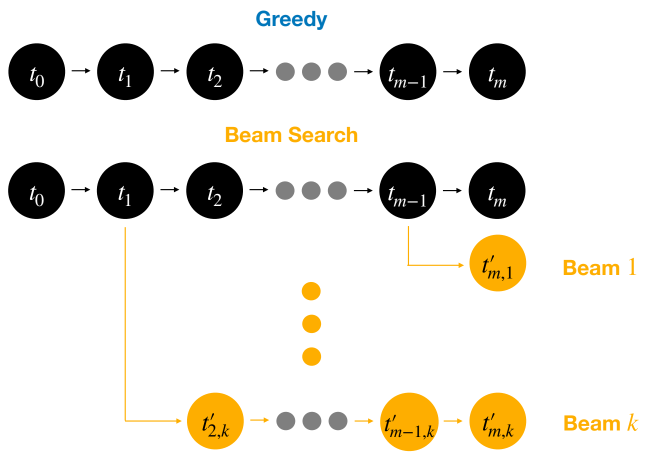

Figure 1: Two-phase beam search to compute schedules using a previously trained RL agent, where the initial greedy phase selects all actions best-first, while k beams explore alternative schedules in the second phase.

$t \leq m$ and m is the maximum starting time of any task in the schedule. In each step of the greedy construction, the algorithm takes the most probable action from action vector a_t that comprises all legal actions for the current state s_t ranked by their probabilities. The greedy phase stops when a complete schedule is computed.

In the second phase, our method computes an entropy for each state in the trace of the greedy algorithm. The obtained entropy values indicate the uncertainty of the policy network while generating the action vector. Roughly, a state has high entropy if all its actions have similar probabilities. Next, we select k states with the highest entropy and run the greedy search from the first stage starting from each of the k selected nodes. The corresponding branches in the search tree are created using the second-best action. At the end of the second phase, the algorithm has obtained $k + 1$ schedules, compares them, and returns the one with the best objective function value as a result. While being k times more expensive than greedy search, beam search usually generates better solutions in an acceptable time. Moreover, one can easily speed up the search for k additional solutions by parallel computations.

Experiments

This section presents the results of two experiments. First, we evaluate DispatchR on a large-scale industrial RCPS problem and compare the solution quality to existing MIP- and CP-based methods (Kovács et al. 2021).[1] In the second experiment, we investigate whether DispatchR can improve the results of these MIP and CP methods by supplying them with high-quality initial solutions.[2]

In both experiments, we used the following setup. We applied Adam (Kingma and Ba 2015) as a gradient-based optimizer with learning rate $\alpha = 0.04$. The gradient estimation was done with PES, using the standard deviation $\sigma = 0.05$ and a population size of $N = 300$. Moreover, the beam search algorithm was limited to $k = 80$ beams, providing a good balance between computational effort and exploration of the search space. The RL agents were trained until convergence, reached after 37800 episodes (126 iterations), which amount to about one hour of computation time on the used hardware—a machine equipped with two 20 cores/40 threads Intel Xeon 6138 CPUs.

[1]Unfortunately, the RL approaches from the literature surveyed in Section come without publicly available implementations or instance sets, and we could thus not include them in our experiments.

[2]Our implementation, a pre-trained model, and benchmarks are available at: https://github.com/ingambe/LearningHeuristics

Benchmark Instances. We perform both experiments on a set of large industrial RCPS problem instances provided by Kovács et al. (2021). Compared to previously mentioned RCPS problems from the literature, this benchmark set is very challenging due to its size. On average, the 15 industrial instances include 6274 jobs, 215 machines, and 17529 tasks to schedule, modeling the projected work for six weeks on production lines at different points in time. That is, the instances of the benchmark set are samples of a common stochastic process. The latter is often not the case for random instances generated independently of one another and, therefore, not exhibiting patterns we could capture with ML models. Unlike that, an RL agent trained on some of the instances by Kovács et al. (2021) can potentially learn scheduling policies applicable to unseen instances as well. Given the limited number of instances, we take the first two (arbitrarily chosen) of 15 instances for training and the 13 remaining ones as test instances. Note that we could not observe any significant improvement in the generalization ability of RL agents when using more instances for training.

Experiment 1: DispatchR as Standalone Solver

In the original work, Kovács et al. (2021) presented MIP and CP models for the investigated industrial RCPS problem, implemented using the state-of-the-art solvers Gurobi (Gurobi Optimization 2021) and OR-Tools CP-SAT (Perron and Furnon 2019), respectively. Due to the size and complexity of instances for scheduling six weeks in advance, both methods require an initial solution to start the optimization. Otherwise, the solvers fail to find a feasible schedule within 3600 seconds (one hour) of computation time, which we set as a limit in our experiments. With an initial solution at hand, both the MIP and CP solver can find solutions of significantly higher quality in the given computation time. Hence, Kovács et al. (2021) proposed a greedy search method with a handcrafted dispatching heuristic based on the features of jobs for determining some initial solution to optimize further. We take these existing methods as baselines to compare the performance of scheduling policies learned by DispatchR without relying on any handcrafted heuristic or initial solution.

Results. The box plot in Figure 2 visualizes the solution quality obtained with the greedy approach as well as the CP and MIP methods by Kovács et al. (2021) on the one hand, and with our DispatchR approach on the other. While the initial solutions determined by greedy search have significant room for improvement, the large size of instances representing six planning weeks only admits modest optimization progress measured by the objective function in Equation (1) with the MIP model, as also observed in (Kovács et al. 2021). The CP approach can improve the initial solutions more successfully yet still leaves a noticeable gap to DispatchR whose schedules usually yield the lowest total tardiness.

Unlike the CP and MIP methods, which perform anytime optimization up to the time limit of 3600 seconds (as the industrial RCPS instances are too complex for earlier termination with an optimal solution), the beam search of DispatchR investigates the fixed number of $k = 80$ beams for schedule computation. Hence, as detailed in Table 1, DispatchR terminates with a high-quality schedule in less than a minute for every instance. The first two instances are distinguished in boldface as we used them for training the policy network of DispatchR, while the other 13 instances form the test set. We crosschecked with an alternative splitting into two training and 13 test instances that the average solution quality varies by less than 0.5%, indicating that the choice of particular training instances does not strongly affect DispatchR results on the investigated industrial RCPS problem. In fact, the schedules found by DispatchR improve on the MIP method by a substantial margin for every instance, amounting to 29% lower total tardiness on average. The average advantage is still about 17% relative to the better scaling CP method, and the improvements are also consistently distributed over the instances, with the only exception of the lower tardiness obtained with the CP solver on Instance 5. However, recall that the latter is run for one hour per instance, while DispatchR finds almost as good or better schedules in less than a minute.

Without providing further detailed results, let us note that the instances for scheduling six weeks in advance are the largest provided by Kovács et al. (2021). On smaller instances covering two planning weeks, the results reported in (Kovács et al. 2021) (for a time limit of 1800 seconds per instance) show an average improvement of 47% relative to the initial solutions determined by greedy search with the CP approach. The MIP method, which scales worse on the large instances, has an advantage here and cuts the total tardiness down by remarkable 65% on average. We also applied DispatchR to the instances representing two planning weeks and obtained an average improvement of 24% over the greedy search with handcrafted dispatching heuristic, thus being less effective than CP and MIP. This means that complete optimization approaches remain the way of choice when they scale within a given computation time, and that learned policies rather than handcrafted heuristics should be taken for greedy search.

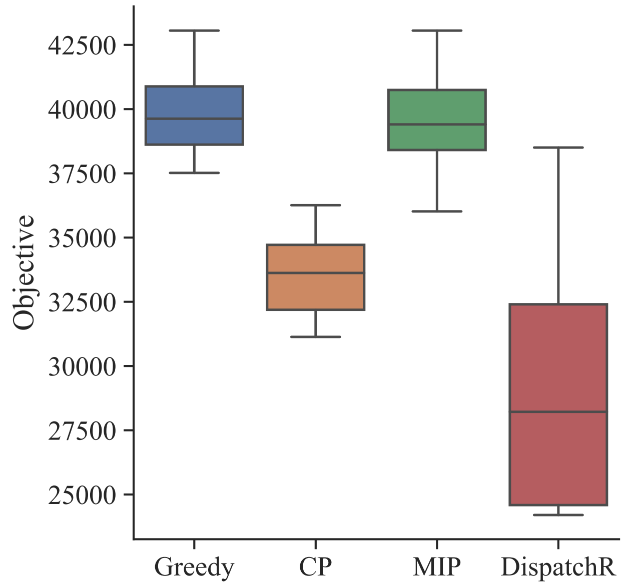

Figure 2: Box plot comparing the quality of solutions (up to cost 44000) by DispatchR to a handcrafted heuristic as well as CP and MIP methods presented by Kovács et al. (2021).

Instances		CP	MIP	DispatchR
Instance 1	Obj.	32061	37517	**24656**
	Time	3600	3600	**38**
Instance 2	Obj.	33752	39405	**24203**
	Time	3600	3600	**38**
Instance 3	Obj.	32374	38806	**28450**
	Time	3600	3600	**41**
Instance 4	Obj.	33125	36019	**32191**
	Time	3600	3600	**42**
Instance 5	Obj.	**31681**	38584	32191
	Time	3600	3600	**42**
Instance 6	Obj.	35366	41023	**32613**
	Time	3600	3600	**45**
Instance 7	Obj.	36120	41994	**32670**
	Time	3600	3600	**45**
Instance 8	Obj.	36260	43056	**32674**
	Time	3600	3600	**44**
Instance 9	Obj.	32217	38720	**28218**
	Time	3600	3600	**40**
Instance 10	Obj.	32162	38233	**24579**
	Time	3600	3600	**40**
Instance 11	Obj.	33623	39627	**24598**
	Time	3600	3600	**38**
Instance 12	Obj.	31134	37726	**28210**
	Time	3600	3600	**42**
Instance 13	Obj.	33744	40255	**24206**
	Time	3600	3600	**38**
Instance 14	Obj.	34067	40467	**24202**
	Time	3600	3600	**38**
Instance 15	Obj.	54058	60815	**38504**
	Time	3600	3600	**41**
Average		34783	40816	**28811**
		3600	3600	**41**

Table 1: Solution quality and computation time (lower numbers are better for both) per instance obtained with the existing CP and MIP methods or DispatchR's scheduling policy. The time is measured in seconds.

Experiment 2: DispatchR plus CP and MIP Solvers

The experiments by Kovács et al. (2021) showed that starting from an initial solution can significantly improve the performance of exact methods. Especially for industrial scheduling problems, it is important to find such a solution quickly to leave time for complete solvers to further improve the quality. In our second experiment, we substitute the original handcrafted heuristic with DispatchR to determine initial schedules to supply to the CP and MIP solvers. We again run the CP and MIP methods for one hour and evaluate the outcomes.

Results. Figure 3 displays the solution quality of the CP and MIP methods starting from initial solutions by DispatchR as well as DispatchR as a standalone solver (like in Figure 2) for comparison. We observe that both the CP and MIP solver succeed to improve the solution quality relative to the standalone DispatchR by 12% or 5%, respectively, on average. Although the MIP approach scales worse than CP on the large instances for six planning weeks, we obtain 33% improvement when using DispatchR instead of the handcrafted dispatching heuristic by Kovács et al. (2021) to determine initial solutions. With the CP method, which leads to the best overall results starting from the schedules found by DispatchR, the average tardiness is 27% lower, thus exhibiting a significant boost of solution quality by combining complete methods with our approach.

Interpretability

Interpreting outputs from neural networks is a notoriously difficult challenge (Zhang et al. 2021). However, understanding the allocation decisions made by a neural network can provide meaningful insights into the heuristic learned by an RL agent and assist human operators in their work. In order to assess the impact of features on the scheduling policy learned by DispatchR, we thus applied Integrated Gradients (Sundararajan, Taly, and Yan 2017), which belongs to the family of attribution methods (Binder et al. 2016; Shrikumar, Greenside, and Kundaje 2017).

The aggregated feature attribution information over training instances in Figure 4 indicates the correlations between job features and the estimated job priorities. Despite the policy network we use is non-linear and may thus involve complex dependencies between job features and priorities, the absolute value of a feature weight quantifies the degree of correlation and its sign expresses whether the feature has a positive or negative influence, respectively. Hence, the feature weights obtained for DispatchR's policy network point out that jobs with few remaining tasks, a short processing time for the current task, buffer constraints between remaining tasks, close deadlines, and few coupled tasks are prioritized. Interestingly, the importance weights w_j for jobs j do not impact the estimated priorities, and a plausible explanation is that the investigated RCPS instances associate the uniform importance weight 1 with almost all of the considered jobs.

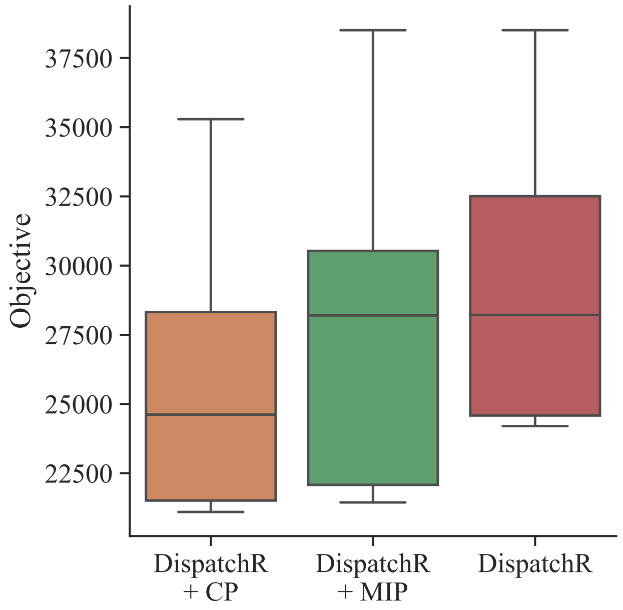

Figure 3: Box plot of solution quality improvements by using DispatchR (instead of the original handcrafted heuristic) to supply an initial solution to the existing CP and MIP methods.

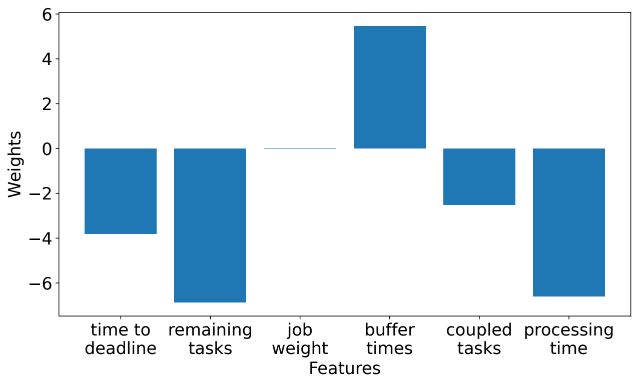

Figure 4: Average feature importance over the training instances, where the signs of weights indicate a positive/negative correlation and absolute values quantify the significance.

Conclusions and Future Work

In this paper, we present DispatchR, a deep RL approach to automatically learn high-quality dispatching strategies, and apply it to a large-scale industrial scheduling problem. We introduce an efficient MDP formulation of dispatching decision-making along with a policy network architecture that does not rely on any specific order and size of jobs. Unlike previous approaches, we do not perform reward shaping but use the unbiased original objective function for training RL agents. Our experiments on a set of large industrial RCPS problem instances show that DispatchR succeeds to produce high-quality schedules in a fraction of the runtime required by exact CP and MIP methods. Since the latter also take advantage of supplied initial solutions, DispatchR can be used both as a standalone solver or an approximate scheduling method for boosting domain-independent optimization techniques.

In future work, we plan to extend DispatchR to generalizations of the considered RCPS problem and to study further scheduling application domains as well. The goal is to obtain general insights about neural architectures required to learn effective dispatching heuristics from the features of large-scale instances of real-world scheduling problems. In addition, we aim to improve the performance of our approach further by exploring more advanced techniques such as Tabu and Monte Carlo tree search. Finally, we intend to broaden the scope of DispatchR to dynamic scheduling scenarios by taking live updates and rescheduling demands into account.

Acknowledgements

This work was partially funded by KWF project 28472, cms electronics GmbH, FunderMax GmbH, Hirsch Armbänder GmbH, incubed IT GmbH, Infineon Technologies Austria AG, Isovolta AG, Kostwein Holding GmbH, and Privatstiftung Kärntner Sparkasse. We are grateful to the anonymous reviewers for their constructive and helpful comments.

References

Ahire, S.; Greenwood, G.; Gupta, A.; and Terwilliger, M. 2000. Workforce-constrained Preventive Maintenance Scheduling Using Evolution Strategies. *Decision Sciences*, 31(4): 833–859.

Amari, S. 1998. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2): 251–276.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2021. Machine Learning for Combinatorial Optimization: A Methodological Tour d'Horizon. *European Journal of Operational Research*, 290(2): 405–421.

Bénichou, M.; Gauthier, J.; Girodet, P.; Hentges, G.; Ribière, G.; and Vincent, O. 1971. Experiments in Mixed-Integer Linear Programming. *Mathematical Programming*, 1(1): 76–94.

Bianchi, L.; Dorigo, M.; Gambardella, L.; and Gutjahr, W. 2009. A Survey on Metaheuristics for Stochastic Combinatorial Optimization. *Natural Computing*, 8(2): 239–287.

Binder, A.; Montavon, G.; Lapuschkin, S.; Müller, K.; and Samek, W. 2016. Layer-Wise Relevance Propagation for Neural Networks with Local Renormalization Layers. In *Proceedings of ICANN*, 63–71. Springer.

Blackstone, J.; Phillips, D.; and Hogg, G. 1982. A State-of-the-Art Survey of Dispatching Rules for Manufacturing Job Shop Operations. *International Journal of Production Research*, 20(1): 27–45.

Blazewicz, J.; Lenstra, J.; and Rinnooy Kan, A. 1983. Scheduling Subject to Resource Constraints: Classification and Complexity. *Discrete Applied Mathematics*, 5(1): 11–24.

Blum, C.; and Roli, A. 2003. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3): 268–308.

Çalis, B.; and Bulkan, S. 2015. A Research Survey: Review of AI Solution Strategies of Job Shop Scheduling Problem. *Journal of Intelligent Manufacturing*, 26(5): 961–973.

Cappart, Q.; Moisan, T.; Rousseau, L.; Prémont-Schwarz, I.; and Ciré, A. 2021. Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization. In *Proceedings of AAAI*, 3677–3687. AAAI Press.

Cavalcante, V.; Cardonha, C.; and Herrmann, R. 2013. A Resource Constrained Project Scheduling Problem with Bounded Multitasking. *IFAC Proceedings Volumes*, 46(24): 433–437.

Chalumeau, F.; Coulon, I.; Cappart, Q.; and Rousseau, L. 2021. SeaPearl: A Constraint Programming Solver Guided by Reinforcement Learning. In *Proceedings of CPAIOR*, 392–409. Springer.

Chopard, B.; and Tomassini, M. 2018. *An Introduction to Metaheuristics for Optimization*. Springer.

Demirkol, E.; Mehta, S.; and Uzsoy, R. 1998. Benchmarks for Shop Scheduling Problems. *European Journal of Operational Research*, 109(1): 137–141.

Ding, J.; Zhang, C.; Shen, L.; Li, S.; Wang, B.; Xu, Y.; and Song, L. 2020. Accelerating Primal Solution Findings for Mixed Integer Programs Based on Solution Prediction. In *Proceedings of AAAI*, 1452–1459. AAAI Press.

Gao, K.; Cao, Z.; Zhang, L.; Chen, Z.; Han, Y.; and Pan, Q. 2019. A Review on Swarm Intelligence and Evolutionary Algorithms for Solving Flexible Job Shop Scheduling Problems. *IEEE/CAA Journal of Automatica Sinica*, 6(4): 904–916.

Garey, M.; Johnson, D.; and Sethi, R. 1976. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 1(2): 117–129.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Morgan & Claypool Publishers.

Glover, F. 1990. Tabu Search - Part II. *INFORMS Journal on Computing*, 2(1): 4–32.

Gomory, R. 2010. Outline of an Algorithm for Integer Solutions to Linear Programs *and* An Algorithm for the Mixed Integer Problem. In *50 Years of Integer Programming*, 77–103. Springer.

Grabot, B.; and Geneste, L. 1994. Dispatching Rules in Scheduling: A Fuzzy Approach. *International Journal of Production Research*, 32(4): 903–915.

Gurobi Optimization, L. 2021. Gurobi Optimizer Reference Manual. https://www.gurobi.com. Accessed: 2022-03-27.

Han, B.; and Yang, J. 2020. Research on Adaptive Job Shop Scheduling Problems Based on Dueling Double DQN. *IEEE Access*, 8: 186474–186495.

Hartmann, S.; and Briskorn, D. 2010. A Survey of Variants and Extensions of the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, 207(1): 1–14.

Herroelen, W.; De Reyck, B.; and Demeulemeester, E. 1998. Resource-Constrained Project Scheduling: A Survey of Recent Developments. *Computers & Operations Research*, 25(4): 279–302.

Ishibuchi, H.; and Murata, T. 1998. A Multi-Objective Genetic Local Search Algorithm and its Application to Flowshop Scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 28(3): 392–403.

Kingma, D.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of ICLR*. https://arxiv.org/abs/1412.6980.

Kovács, B.; Tassel, P.; Kohlenbrein, W.; Schrott-Kostwein, P.; and Gebser, M. 2021. Utilizing Constraint Optimization for Industrial Machine Workload Balancing. In *Proceedings of CP*, 36:1–36:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Kullback, S.; and Leibler, R. A. 1951. On information and sufficiency. *The annals of mathematical statistics*, 22(1): 79–86.

Laud, A. 2004. *Theory and Application of Reward Shaping in Reinforcement Learning*. Ph.D. thesis, University of Illinois.

Liu, C.; Chang, C.; and Tseng, C. 2020. Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems. *IEEE Access*, 8: 71752–71762.

Marchand, H.; Martin, A.; Weismantel, R.; and Wolsey, L. 2002. Cutting Planes in Integer and Mixed Integer Programming. *Discrete Applied Mathematics*, 123(1-3): 397–446.

Meister, C.; Cotterell, R.; and Vieira, T. 2020. If Beam Search is the Answer, What was the Question? In *Proceedings of EMNLP*, 2173–2185. Association for Computational Linguistics.

Metz, L.; Freeman, D.; Schoenholz, S.; and Kachman, T. 2021. Gradients are Not All You Need. https://arxiv.org/abs/2111.05803.

Minsky, M. 1961. Steps toward Artificial Intelligence. *Proceedings of the IRE*, 49(1): 8–30.

Muth, J.; and Thompson, G., eds. 1963. *Industrial Scheduling*. Prentice-Hall.

Nair, V.; Bartunov, S.; Gimeno, F.; von Glehn, I.; Lichocki, P.; Lobov, I.; O'Donoghue, B.; Sonnerat, N.; Tjandraatmadja, C.; Wang, P.; Addanki, R.; Hapuarachchi, T.; Keck, T.; Keeling, J.; Kohli, P.; Ktena, I.; Li, Y.; Vinyals, O.; and Zwols, Y. 2020. Solving Mixed Integer Programs Using Neural Networks. https://arxiv.org/abs/2012.13349.

Padberg, M.; and Rinaldi, G. 1991. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. *SIAM Review*, 33(1): 60–100.

Pellerin, R.; Perrier, N.; and Berthaut, F. 2020. A Survey of Hybrid Metaheuristics for the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, 280(2): 395–416.

Perron, L.; and Furnon, V. 2019. OR-Tools. https://developers.google.com/optimization. Accessed: 2022-03-27.

Pinedo, M. 2016. *Scheduling: Theory, Algorithms, and Systems*. Springer.

Potts, C.; and Strusevich, V. 2009. Fifty Years of Scheduling: A Survey of Milestones. *Journal of the Operational Research Society*, 60(S1).

Prouvost, A.; Dumouchelle, J.; Scavuzzo, L.; Gasse, M.; Chételat, D.; and Lodi, A. 2020. Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers. https://arxiv.org/abs/2011.06069.

Rinnooy Kan, A. 2012. *Machine Scheduling Problems: Classification, Complexity and Computations*. Springer.

Rossi, F.; van Beek, P.; and Walsh, T., eds. 2006. *Handbook of Constraint Programming*. Elsevier.

Salimans, T.; Ho, J.; Chen, X.; and Sutskever, I. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. https://arxiv.org/abs/1703.03864.

Shrikumar, A.; Greenside, P.; and Kundaje, A. 2017. Learning Important Features Through Propagating Activation Differences. In *Proceedings of ICML*, 3145–3153. PMLR.

Sotskov, Y.; and Shakhlevich, N. 1995. NP-hardness of Shop-scheduling Problems with Three Jobs. *Discrete Applied Mathematics*, 59(3): 237–266.

Stahlberg, F.; and Byrne, B. 2019. On NMT Search Errors and Model Errors: Cat Got Your Tongue? In *Proceedings of EMNLP-IJCNLP*, 3354–3360. Association for Computational Linguistics.

Sundararajan, M.; Taly, A.; and Yan, Q. 2017. Axiomatic Attribution for Deep Networks. In *Proceedings of ICML*, 3319–3328. PMLR.

Sung, I.; Choi, B.; and Nielsen, P. 2020. Reinforcement Learning for Resource Constrained Project Scheduling Problem with Activity Iterations and Crashing. *IFAC-PapersOnLine*, 53(2): 10493–10497.

Taillard, E. 1993. Benchmarks for Basic Scheduling Problems. *European Journal of Operational Research*, 64(2): 278–285.

Tang, Y.; Agrawal, S.; and Faenza, Y. 2020. Reinforcement Learning for Integer Programming: Learning to Cut. In *Proceedings of ICML*, 9367–9376. PMLR.

Tassel, P.; Gebser, M.; and Schekotihin, K. 2021. A Reinforcement Learning Environment For Job-Shop Scheduling. https://arxiv.org/abs/2104.03760.

Vaessens, R.; Aarts, E.; and Lenstra, J. 1996. Job Shop Scheduling by Local Search. *INFORMS Journal on Computing*, 8(3): 302–317.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *Proceedings of NIPS*, 5998–6008. Curran Associates, Inc.

Wierstra, D.; Schaul, T.; Glasmachers, T.; Sun, Y.; Peters, J.; and Schmidhuber, J. 2014. Natural Evolution Strategies. *Journal of Machine Learning Research*, 15(1): 949–980.

Zhang, C.; Song, W.; Cao, Z.; Zhang, J.; Tan, P.; and Xu, C. 2020. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. In *Proceedings of NIPS*, 1621–1632. Curran Associates, Inc.

Zhang, Y.; Tiňo, P.; Leonardis, A.; and Tang, K. 2021. A Survey on Neural Network Interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5): 726–742.