Manufacturing Control in Job Shop Environments with Reinforcement Learning

Vladimir Samsonov[1][a], Marco Kemmerling[1][b], Maren Paegert[1][c], Daniel Lütticke[1][d], Frederick Sauermann[2][e], Andreas Gützlaff[2][f], Günther Schuh[2][g] and Tobias Meisen[3][h]

[1]*Institute of Information Management in Mechanical Engineering, RWTH Aachen University, Aachen, Germany*

[2]*Laboratory for Machine Tools and Production Engineering WZL, RWTH Aachen University, Aachen, Germany*

[3]*Chair of Technologies and Management of Digital Transformation, University of Wuppertal, Wuppertal, Germany*

Keywords: Manufacturing Control, Production Scheduling, Job Shop Scheduling, Deep Reinforcement Learning, Combinatorial Optimization.

Abstract: Computing solutions to job shop problems is a particularly challenging task due to the computational hardness of the underlying optimization problem as well as the often dynamic nature of given environments. To address such scheduling problems in a more flexible way, such that changing circumstances can be accommodated, we propose a reinforcement learning approach to solve job shop problems. As part of our approach, we propose a new reward shaping and devise a novel action space, from which a reinforcement learning agent can sample actions, which is independent of the job shop problem size. A number of experiments demonstrate that our approach outperforms commonly used scheduling heuristics with regard to the quality of the generated solutions. We further show that, once trained, the time required to compute solutions using our methodology increases less sharply as the problem size grows than exact solution methods making it especially suitable for online manufacturing control tasks.

1 INTRODUCTION

Optimization problems in production planning and control (PPC) are often solved using simple priority rules and heuristics, which tend to achieve reasonable results with short computation time. While exact approaches such as integer linear programming (ILP) can be used to find optimal solutions, the time required to find such solutions is often impractical due to the NP-hardness of problems in this space (Garey and Johnson, 1979; Kan and Rinnooy, 2012). Furthermore, such methods create complete solutions for a given set of inputs, e.g. all jobs on a given day. Should any of the underlying assumptions of the optimization problem change after the solution is created, these algorithms will have to create a complete solution once more, which again will result in a high cost in terms of computation time and render the application of such methods impractical in a dynamic production environment.

Reinforcement learning (RL) offers an alternative to the aforementioned approaches. Here, a so-called agent creates a solution step by step, while the current situation is reassessed at every step, such that new information about changing circumstances is reflected in the future decision-making process of the agent. Before such an agent can be employed in practice, it needs to be trained with regard to a specific problem formulation. While this training is usually time-intensive, it only needs to be performed once and a trained agent can create scheduling solutions in reasonable time. Since a reinforcement learning agent essentially learns a heuristic suitable to a specific problem, there are no mathematical guarantees for solutions created in this manner to be optimal, but there is a possibility that such a learned heuristic can outperform currently used heuristics and priority rules

[a] https://orcid.org/0000-0003-4147-6470
[b] https://orcid.org/0000-0003-0141-2050
[c] https://orcid.org/0000-0003-4952-6639
[d] https://orcid.org/0000-0002-7070-8018
[e] https://orcid.org/0000-0001-7343-9338
[f] https://orcid.org/0000-0003-4370-8829
[g] https://orcid.org/0000-0002-6076-0701
[h] https://orcid.org/0000-0002-1969-559X

Samsonov, V., Kemmerling, M., Paegert, M., Lütticke, D., Sauermann, F., Gützlaff, A., Schuh, G. and Meisen, T.
Manufacturing Control in Job Shop Environments with Reinforcement Learning.
DOI: 10.5220/0010202405890597
In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Volume 2*, pages 589-597
ISBN: 978-989-758-484-8

in terms of solution quality while keeping the time required for scheduling low.

While reinforcement learning approaches have been investigated in production planning and scheduling before (Zhang and Dietterich, 1995; Aydin and Öztemel, 2000; Schneckenreither and Haeussler, 2018; Gabel and Riedmiller, 2008; Waschneck et al., 2018; Qu et al., 2016), some shortcomings still exist. One crucial issue is the capability of an agent to generalize to previously unseen job shop problems (JSPs) of various sizes. If state representation and action space are directly linked to the size of the problem, i.e. the number of machines and orders in the production, generalization is only possible to a very limited degree. This paper aims to make one step towards a reinforcement learning approach capable of generalizing to different problem sizes by presenting a novel action space design independent of the number of machines and orders involved in the planning. How this design affects generalization performance will be a focus in the work presented here. Additionally, we introduce a new reward shaping approach aimed to incentivize the learning of near-to-optimal scheduling strategies through high reward gradients towards the optimum.

The remainder of the paper is structured as follows. The next section gives a brief description of the necessary background on production planning and scheduling, JSP, relevant metrics, common scheduling approaches, as well as a brief introduction to reinforcement learning. Section 3 discusses existing work on the use of RL for production planning and control, while section 4 elaborates on our own approach to the issue. Section 5 presents experimental results generated by our approach and some final considerations are given in section 6.

2 BACKGROUND

2.1 Production Planning and Control

Today, manufacturing companies are, among other trends, facing the problem of increasing demand for customized products (Zijm and Regattieri, 2019; Jacobs, 2011; Gyulai et al., 2018). As a consequence, companies need to increase the flexibility of their production. Therefore, they often decide for the principle of job shop production, allowing to manufacture more customer-specific product offerings (Duffie et al., 2017; Schuh et al., 2019). Here, components can pass through the workstations required for processing in a flexible sequence in a mostly undirected material flow (Zijm and Regattieri, 2019). As a result, complexity in PPC increases and demands for well-designed strategies that support mastering this complexity (ElMaraghy et al., 2013).

For mastering the complexity, targets of PPC need to be understood. In general, the so-called logistical targets can be classified in logistical performance and logistical costs. The target of improving logistical performance can be further specified by shortening lead times and increasing the adherence to production schedules, both internally and externally. On the other hand, the target of reducing logistical costs is expressed by increasing utilization of production resources and reducing work-in-process (WIP) and tied capital, subsequently. However, those targets depend on each other and improving one can lead to a deterioration of others (e.g. reducing WIP can lead to a decrease in utilization). Hence, the optimization of all targets can not be pursued at the same time, and companies need to prioritize, with high adherence to schedules and especially delivery dates usually being the most important targets (Lödding, 2013; Gyulai et al., 2018).

The planning of an optimal sequence and assignment of orders to machines to achieve the logistic goals prioritized by companies is laborious and complex. To approach these problems, manufacturing companies usually use software systems such as an enterprise resource planning (ERP) system. These systems are based on the logic of manufacturing resource planning (MRP II), which is an extension of material requirements planning (MRP) (Zijm and Regattieri, 2019). In a MRP II process, end products are initially broken down into their components. Then, the steps of operation of these components are assigned to work stations. Finally, the different components are put into order on each machine, trying to meet the end completion date of the whole order (Zijm and Regattieri, 2019; Kurbel, 2016). Due to the immense combinatorial complexity of PPC tasks such as order release and sequencing on machines, practical approaches often make use of heuristics to enable nearly optimal solutions with significantly reduced effort (Kurbel, 2016).

The task of order release is to determine an optimal point in time to start with the production of an order in a job shop production environment. This task has a strong impact on the logistical targets. Releasing orders close to their delivery dates can reduce WIP and lead times, but bears the risk of a low adherence to delivery dates. On the other hand, releasing orders early does not necessarily lead to a higher probability of better adherence to delivery dates, as an increase of WIP results in longer and more scattered lead times that are difficult to plan (Buker, 2001; Mather and

Plossl, 1978). As a result, the goal is to release orders as late as possible while still trying to meet delivery dates. Typical heuristics for releasing orders are e.g. constant WIP (conWIP), bottleneck control, or workload control (Lödding, 2013).

With order lead times up to several weeks or months, a second important task in PPC is order sequencing at work stations (Schuh et al., 2019). The goal of order sequencing is to support the achievement of logistical targets. The aim is to sort and process work steps on the work stations in such a way that all orders are completed on schedule. Common heuristics for sequencing are first-in-first-out (FIFO), setup time optimized or minimum slack. Further heuristics to address the problem are the shortest-processing-time (SPT) and longest-processing-time (LPT) priority rules, in which operations are sequenced in increasing, respectively decreasing order of their processing times. Finally, the longest-remaining-processing-time (LRM) rule selects the operation of the job with the longest remaining processing time. Here, the operation under consideration is excluded from the computation of the remaining processing time (Lödding, 2013).

2.2 Reinforcement Learning

Reinforcement learning is an area of machine learning which investigates how an agent can learn to perform a specific task in a given environment. In order to apply reinforcement learning, the problem at hand needs to be modeled as a Markov decision process (MDP). Such a MDP features a state s which can change over the course of discrete time steps. In each time step, an action a available in the current state can be chosen, which may influence the state s' in the next time step and lead to a reward $R_a(s,s')$. A reinforcement learning agent learns a policy, i.e. a mapping from states to actions, by receiving the corresponding rewards for performing actions in a problem modeled as an MDP. A policy is learned with the goal of maximizing not the reward at any single time step, but rather the long-term cumulative reward (Sutton and Barto, 2018). If a sequence of states has a clear beginning and endpoint, it is called an episode and the corresponding problem is considered an episodic task.

Reinforcement learning algorithms can be divided into value-based, policy-based, and hybrid approaches. In value-based approaches, the value of each available action in a specific state is estimated and a policy is derived subsequently from these estimated values. In policy-based based approaches, a policy as described above is learned directly as a probability distribution of all actions in a given state. Such policy approaches typically update the learned policy at the end of an episode based on the overall reward received in that episode. Since actions are not evaluated individually, the current state of the policy might be evaluated as good overall while some of the actions performed where actually disadvantageous or vice versa. So-called actor-critic approaches are a hybrid of value-based and policy-based approaches that allow for an evaluation of individual actions by updating the parameters of an actor, which is the policy-based component, based on the value estimates computed by a critic, which is the value-based component. In contrast to value-based approaches, actor-critic and policy-based approaches lend themselves well to high-dimensional and continuous action spaces (Konda and Tsitsiklis, 2000). In our work, we apply Deep Q-Learning (DQN) as an example of a value-based approach operating with discrete action spaces and Soft Actor-Critic (SAC) as a representative of the actor-critic group of algorithms relying on continuous action space design.

DQN is a variant of the well-known Q-Learning algorithm (Watkins, Christopher John Cornish Hellaby, 1989), in which the action-value function $Q(s,a)$, i.e. the function describing the long-term expected reward when performing action a in state s, is approximated. These estimates are updated based on the Bellmann equation given below:

$$Q(s,a) = Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right) \tag{1}$$

where the learning rate α determines how much new information affects the current estimate, r is the current reward and γ is a discounting factor describing how much projected rewards from future actions influence the current value.

While these estimates were originally stored in simple tables where each cell corresponded to the Q-Value of one specific action in one specific state, current approaches such as DQN use deep neural networks to approximate action-values. These networks improve by updating their parameters in the direction of the gradient of a so-called loss function, which compares the value $Q(s,a)$ predicted by the network with the correct value. Since the correct value is not known in this case, it is substituted by $r + \gamma \max_{a'} Q(s',a')$, similarly as in eq. (1). In contrast to DQN, the SAC algorithm maximizes not just the expected reward but also the entropy of its policy, which promotes exploration and prevents premature convergence to local optima by encouraging the agent to act with as much randomness as possible while still succeeding at the given task. (Haarnoja et al., 2018).

3 RELATED WORK

Early applications of reinforcement learning in the area of production planning and scheduling comprised the modification of an initial, but infeasible solution by iteratively reducing the number of violated constraints in order to arrive at a feasible solution (Zhang and Dietterich, 1995; Zhang and Dietterich, 1996). Further work has been carried out in the application of reinforcement learning for order release (Schneckenreither and Haeussler, 2018). Another approach focuses on the selection of appropriate priority rules to schedule jobs based on the current state (Aydin and Öztemel, 2000).

While the number of single-agent approaches to the job-shop problem in the literature is scarce, multi-agent approaches have been investigated particularly often in the area, possibly because modeling such a problem becomes easier when various decisions can be decoupled from each other and addressed separately. One such example is the assignment of a separate agent to each machine in a job-shop scheduling problem as described in (Gabel and Riedmiller, 2008; Gabel, 2009). More recent examples of this type of approach have been applied in the context of a semi-conductor production facility (Waschneck et al., 2018), and in the combination of an agent performing scheduling with an agent performing human resource management to solve a flow-shop problem (Qu et al., 2016).

While multi-agent approaches offer an easy way to scale to bigger problem sizes by increasing the number of agents, training such a group of agents tends to be more difficult due to the inherent non-stationary of an environment inhabited by other agents, as well as the need to ensure that the agents cooperate to solve overarching problem. Scaling to bigger problem instances using a single-agent approach is not trivial, as state and action spaces are typically dependent on the number of machines and jobs. We partially address this issue by introducing an action space independent of the number of jobs.

4 EXPERIMENTAL SETUP AND SOLUTION DESIGN

The given experimental setting aims to evaluate the proposed RL approach designed for simultaneous management of two manufacturing control tasks in complex production environments: order release and operation sequencing. We pick job-shop production as one of the most well known and challenging manufacturing control environments. Even small problem instances with 3 jobs and 3 machines are demonstrated to have NP-hard combinatorial complexity of underlying scheduling tasks (Sotskov and Shakhlevich, 1995).

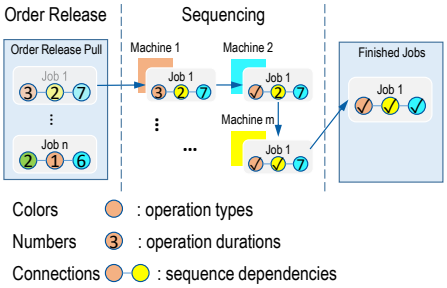


Figure 1: Job-Shop Order Release and Scheduling Setup.

4.1 Job-shop Problem Formalization

A set of job-shop problems used in this study is formalized as follows: each problem instance has a set of jobs $J = \{J_1, ..., J_n\}$ to be manufactured. Each job J_i $(i = 1, ..., n)$ includes a set of operations $O_i = \{O_{i1}, ..., O_{in}\}$ that can be processed in a predefined sequence only. Each operation O_{iq} $(q = 1, ..., n)$ has a non-negative integer processing time t_{iq} and is assigned for processing to one machine from a machine set $M = \{M_1, ..., M_m\}$. Each machine M_j $(j = 1, ..., m)$ can process one operation O_{iq} $(q = 1, ..., n)$ at a time (Sotskov and Shakhlevich, 1995). We define the optimization objective as finding a schedule J on M that minimizes the latest end time of any job J_i, also referred to as the makespan C_{max}.

To use RL for the task of order release and operation sequencing, we define the solution process as a Markov Decision Problem (MDP) (Howard, 1960). In the initial state, all machines are idle and orders included in the production program are placed in an order release pull. Through a set of actions, the RL agent can decide to stay inactive for one time unit, move an order from order release pull into production (order release), or move an order from one machine to another machine (order sequencing). The final state of an episode is achieved when all planned orders are finished or the number of actions executed by the RL agent exceeds the predefined limit. A sparse reward is used, meaning that a reward is given only at the last step of an episode and the reward for all intermediate steps is zero. This sparse reward is inversely related to the achieved makespan (see Figure 1).

4.2 RL Implementation

The considered RL implementation is divided into three major design aspects: state space representation,

action space, and reward shaping.

4.2.1 State Space Representation

The state observed by the agents is comprised of six separate components: (1) the machine states, i.e. the remaining time for the operations currently being processed on each individual machine, (2) the sum of all operations' processing times currently in the queue of each individual machine, (3) the sum of all operations' processing times for each individual job, (4) the duration of the next operation for each individual job, (5) the index of the next required machine for each individual job, and finally (6) the time already passed in any given moment. If no job is being processed on a machine at a particular point in time, the corresponding entry in (1) will be -1. The relevant elements of (4) and (5) will likewise be set to -1 if no further operation needs to be carried out for a particular job. Each of these components is normalized before being passed onto the agent.

4.2.2 Action Space Design

Based on the observed state, the agent needs to decide which operation O_{ij} to process next, if any. To ensure a consistent action space regardless of the number of jobs and operations per job, the agent selects a relative duration a which will be mapped to a specific operation, rather than selecting the operation directly. Such a relative duration selected by the agent is then mapped back to an absolute duration using the overall minimum and maximum processing times t_{min} and t_{max}, respectively. This absolute duration is then compared against every available operation's processing time t_{iq} and the operation O_{sel} with the closest processing time is selected, as formalized in eq. (2).

$$O_{sel} \leftarrow \arg\min_{i,q} \left(|t_{iq} - (t_{min} + \frac{a}{10} * (t_{max} - t_{min})| \right) \quad (2)$$

Should multiple operations with identical processing times exist, ties will be broken by applying the SPT rule as a first step and, if necessary, applying the LRM rule in a second step. The proposed action space design allows for the use of RL agents with both discrete or continuous action space. In the discrete case, possible actions span the integers in a range $[0, 10]$, which represent relative durations. A continuous action space modification comprises the real numbers instead of integers in a range $[0, 10]$, thus potentially enabling better precision when selecting operations. Aside from the already mentioned action space above, there is one special action with value -1, upon which no operation will be selected at all, extending the overall action space to a range of integers or real numbers $[-1, 10]$.

4.2.3 Reward Shaping

Next to state and action spaces, a reward function is necessary in order to apply RL. In this work, we define a sparse reward function which only evaluates the performance of the agent at the end of an episode. While learning usually benefits from more frequent rewards, in this case intermediate rewards are difficult to define in such a way that they correspond to the actual optimization objective, which can only be accurately evaluated once an episode has terminated.

The sparse reward function $r(T)$ employed here is defined as in eq. (3), giving high rewards when the makespan T of the solution is close to the optimal one T_{opt} and exponentially lower rewards as the quality of the solution moves away from the optimal one. Coefficient γ defines the steepness of the reward gradient and is set to 1.025 in this work.

$$r(T) = 1000 \frac{\gamma^{T_{opt}}}{\gamma^{T}} \quad (3)$$

4.3 Evaluation Approach

The main evaluation criteria in this study are solution quality, solution speed, and scalability to bigger problem instances. We evaluate our proposed RL approach against three alternative approaches: the well-known OR-tools implementation of the CP-SAT solver (Perron and Furnon, 2020), as well as two common priority rule heuristics: shortest processing time (SPT) and longest processing time (LPT). This choice is by no means an exhaustive representation of common JSP solution approaches and is meant to give an estimation of the relative performance against a set of alternative JSP solvers designed specifically for solution speed or optimality.

In this work three jsp instance sizes are used in experiments:

- jsp instances with 6 machines, 6 jobs, 6 operations per machine refereed to as 6x6x6 jsp instances.
- jsp instances with 10 machines, 10 jobs, 10 operations per machine refereed to as 10x10x10 jsp instances.
- jsp instances with 15 machines, 15 jobs, 15 operations per machine refereed to as 15x15x15 jsp instances.

The generation of JSP instances for training and evaluation is completely random with no selection involved. Durations of operations are uniformly sam-

pled within a range $[1,11]$ time units. Each operation is randomly assigned to one of the machines. No selection of JSP instances is performed. Each JSP instance is solved with the CP-SAT, SPT, and LPT solver. The makespan for each JSP instance as found by the CP-SAT solver is considered to be a reference value for reward calculation (see eq. (3)) and is used as a performance benchmark in our study. Solution quality is formalized as an optimality gap ($OptGap$) and can bee seen as a function of difference between achieved makespan (T) and the reference makespan found by CP-SAT solver (T_{opt}) (see eq. (4)). Lower values of the optimality gap correspond to better solutions.

$$OptGap = \frac{T - T_{opt}}{T_{opt}} \cdot 100 \tag{4}$$

5 EXPERIMENTS AND RESULTS

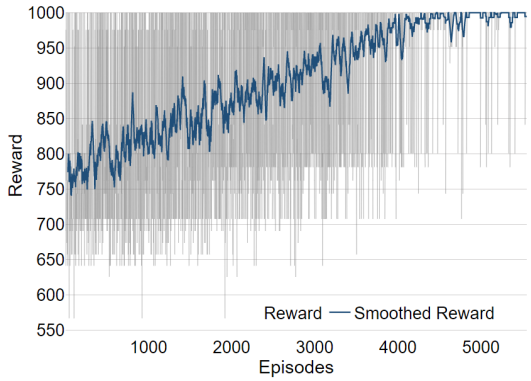
During the first iteration of experiments we conduct several training and evaluation runs on single JSP instances of various sizes. This is meant to test the capability of an RL agent to iteratively find and improve a JSP solution through exploration as well as to test the applicability of RL agents with discrete and continuous action spaces. Figure 2 demonstrates that the new action space design is equally suitable for the use with RL agents relying on discrete action spaces (e.g. DQN) as well as RL agents operating on continuous action spaces (e.g. SAC). Due to space limitations, the rest of the work presents results achieved with DQN RL agents only. However, as demonstrated here, comparable results can be achieved with an SAC RL agent.

As a next step we look into using the same RL setup on single JSP instances of 10x10x10 and 15x15x15 size (see Figure 3). Both runs demonstrate a gradual improvement of reward per episode during the training phase. For JSP sizes 10x10x10 and 15x15x15, the RL agent is given 1,500,000, and 2,000,000 time steps respectively to gradually improve the dispatching and scheduling strategy. A growing optimality gap can be seen with a growing JSP size. Nevertheless, in both cases, the chosen RL approach surpasses the common LPT and SPT heuristics demonstrating the possibility of the chosen approach to scale to bigger JSP instances with sufficient training.

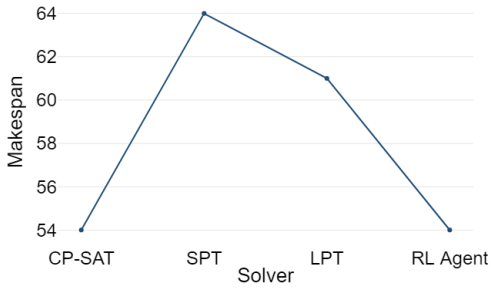
Depending on operations strategies, SPT and LPT are well-known sequencing rules in terms of throughput or utilization maximization. One of the success factors of those priority rule heuristics is the possibility to generate a new production schedule in a short period of time. A lower runtime required to find a feasible and near-to-optimal schedule increases the planning flexibility and allows to minimize the negative impact of unexpected changes and disturbances in the production environment. Figure 4 demonstrates the average time needed to solve 50 JSP instances of sizes 6x6x6, 10x10x10 and 15x15x15 with the SPT and LPT heuristics, the proposed RL agent, as well as with the CP-SAT solver. While the SPT heuristic, LPT heuristic and RL approach demonstrate a moderate increase in required runtime to generate a JSP solution with growing problem size, the CP-SAT runtime quickly becomes infeasible for online scheduling. A 15x15x15 JSP instance can require up to 2.2 hours of runtime for the CP-SAT solver. The longest observed solution time for the investigated RL approach for a JSP instance of the same size is 40 seconds, while the SPT and LPT heuristics consume up to 14 seconds each. All runtime benchmarks are conducted on the same hardware featuring two Intel Xeon E5-2687W CPUs and 256 GB RAM.

Further analysis of RL agent runtimes demonstrates that the time required for inference of the next step by the RL agent based on the given production state is bellow 1% of the calculation times spend on state updates within the production simulation environment. In future work the execution time of the developed production simulation can be greatly increased by using faster Python implementations such as *PyPy* or by adopting process-based discrete-event simulation frameworks such as *SimPy*.

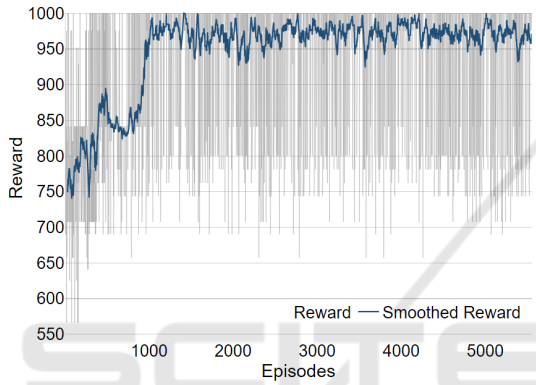
The main interest of our work lies in the investigation of the generalization capabilities of the proposed RL approach. To ensure the generalization capabilities of the RL agent, we conduct training on 900 JSP instances of size 6x6x6. It takes 2,000,000 training steps to learn a scheduling strategy surpassing the SPT and LPT heuristics for unseen JSP instances. The final evaluation is conducted on 50 JSP instances not seen during training. Each RL training is conducted ten times with different fixed random seeds. Evaluation results from all ten training runs are used in the final comparison against the alternative solution approaches. Figure 5 depicts optimality gap distributions for schedules found by the trained RL agent, SPT, and LPT heuristic compared to the reference JSP solutions found by the CP-SAT solver. On average, schedules generated by the trained RL agent have 4.5% smaller optimality gap compared to the next best SPT heuristic. 6% of generated RL solutions are optimal, while no optimal schedules were found by SPT or LPT heuristics. The biggest observed optimality gaps on the evaluation JSP set for RL agent,



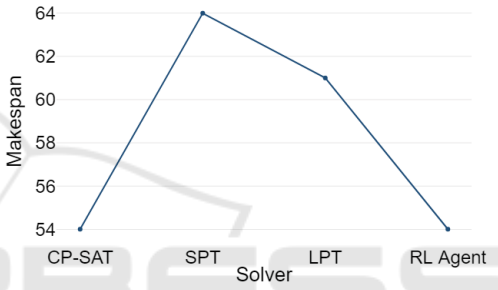
(a) DQN training for 200,000 time steps: RL agent incrementally learns better JSP solution



(b) DQN evaluation: RL agent has found a solution with the shortest makespan

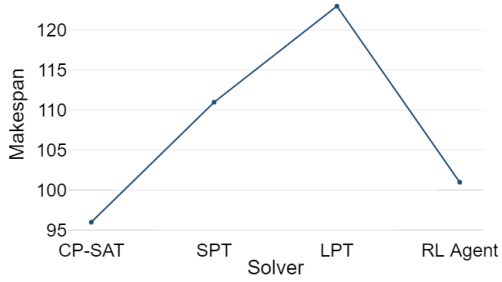


(c) SAC training for 200,000 time steps: RL agent incrementally learns better JSP solution

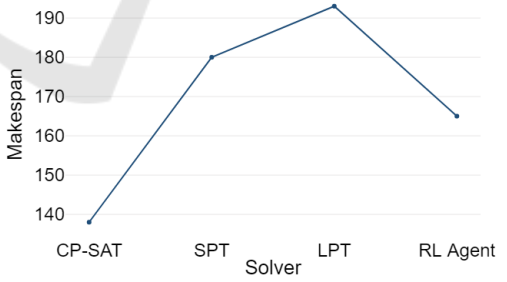


(d) SAC evaluation: RL agent has found a solution with the shortest makespan

Figure 2: Training and evaluation of RL agents on a single 6x6x6 JSP instance: both RL agents with continuous and discrete action space can be successfully used with the proposed action space design.



(a) Single 10x10x10 jsp instance: DQN agent found better JSP solution comparing to SPT and LPT heuristics



(b) Single 15x15x15 JSP instance: DQN agent found better JSP solution comparing to SPT and LPT heuristics

Figure 3: DQN agent benchmarking on bigger single JSP instances.

SPT and LPT heuristics are 39%, 39%, and 69% respectively. The best RL run reduces the biggest optimality gap down to 26%.

To statistically investigate observed differences in optimality gaps between different scheduling methods, the Wilcoxon signed-rank test (Rey and Neuhäuser, 2011) is used. It is a non-parametric statistical test that makes no data normality assumption and is used to compare two related samples. In this case, we do the pairwise comparison of the SPT and

Table 1: Statistical investigation of observed optimality gap differences: Wilcoxon signed-rank test p-values.

Priority Rules	RL Run 1	RL Run 2	RL Run 3	RL Run 4	RL Run 5	RL Run 6	RL Run 7	RL Run 8	RL Run 9	RL Run 10
SPT	0.066	0.025	0.010	0.040	0.008	0.034	0.004	0.012	0.07	0.011
LPT	$1.8E^{-4}$	$2.14E^{-5}$	$6.58E^{-6}$	$7.17E^{-5}$	$1.03E^{-5}$	$4.76E^{-5}$	$1.99E^{-6}$	$1.03E^{-5}$	$2.69E^{-4}$	$7.23E^{-6}$

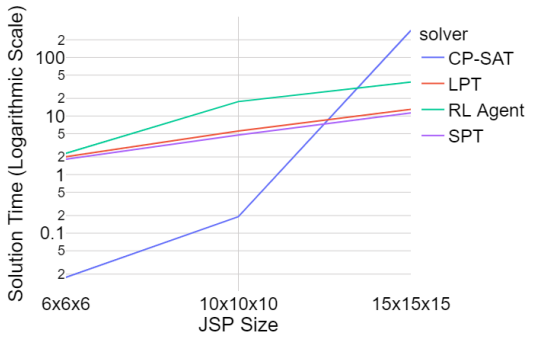


Figure 4: Comparison of average runtime per JSP instance of various sizes.

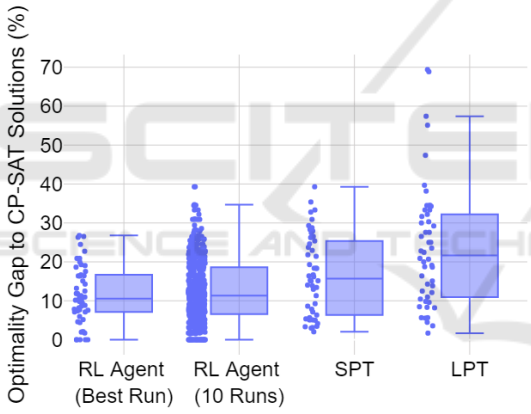


Figure 5: Generalization test: optimality gaps for different JSP solvers on a set of 50 unseen 6x6x6 jsp instances.

LPT heuristics with RL agents trained in ten independent runs. All comparisons are conducted on the same 50 test JSP instances. The null hypothesis is stated as follows: there is no difference between the medians of the two observed populations, e.g. there is no difference in performance between the given priority rule and RL agent. Table 1 provides resulting p-values for the formulated null-hypothesis. Assuming a significance level of 0.05, no conclusion about the observed difference between the LPT heuristic and RL runs 1 and 9 can be derived. For the remaining 8 comparisons, there is sufficient evidence to support the claim that the RL approach yields scheduling solutions with lower optimality gaps compared to the SPT and LPT heuristics.

6 CONCLUSION AND OUTLOOK

This work adapts methods of RL for job shop scheduling. The main contribution of this work is the introduction of a novel action space, which is independent of the number of machines and orders allowing for use of RL agents with continuous and discrete action spaces. To the best of our knowledge, this is the first action space design independent of the problem dimensionality. Additionally, we introduce a new reward shape encouraging the learning of optimal schedules through higher reward gradients for near-optimal solutions. In several evaluations, we demonstrate that the trained RL agent can find consistently better schedules for unseen JSPs compared to common priority rule approaches, and is orders of magnitudes faster compared to state of the art constraint-programming solvers such as the CP-SAT implementation from OR-tools. Our RL approach offers a good balance between speed and solution quality which is a crucial factor for online scheduling applications in dynamic production environments.

One important direction of future work is transferring the achieved results to bigger, and hence more realistic JSP instances. This work will concentrate on three main directions: increasing RL training speed and efficiency, enhancing reward design, as well as developing new solution designs for the state-space representation. To make extensive training of RL agents possible, a runtime optimization of the developed production simulation should be conducted. Additionally, distributed RL training can be adopted by the deployment of such RL methods as Proximal Policy Optimization (Schulman et al., 2017). Our proposed reward shape requires a precalculated optimum for every JSP instance used for training. This can be eliminated by adopting the ”ranked reward” idea presented by Laterre et al. (Laterre et al., 2018). Finally, in addition to the action space, it is important to make the state space representation agnostic to the problem dimensionality as well, so that one RL agent can be used for JSP problems of various sizes.

ACKNOWLEDGEMENTS

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2023 Internet of Production – 390621612.

REFERENCES

Aydin, M. and Öztemel, E. (2000). Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 33(2-3):169–178.

Buker, D. W. (2001). Inventory management and control. In Maynard, H. B. and Zandin, K. B., editors, *Maynard's industrial engineering handbook*, McGraw-Hill standard handbooks, pages 1591–1614. McGraw-Hill, New York.

Duffie, N., Bendul, J., and Knollmann, M. (2017). An analytical approach to improving due-date and lead-time dynamics in production systems. *Journal of Manufacturing Systems*, 45:273–285.

ElMaraghy, H., Schuh, G., ElMaraghy, W., Piller, F., Schönsleben, P., Tseng, M., and Bernard, A. (2013). Product variety management. *CIRP Annals*, 62(2):629–652.

Gabel, T. (2009). Multi-agent reinforcement learning approaches for distributed job-shop scheduling problems.

Gabel, T. and Riedmiller, M. (2008). Adaptive reactive job-shop scheduling with reinforcement learning agents. *International Journal of Information Technology and Intelligent Computing*, 24(4):14–18.

Garey, M. R. and Johnson, D. S. (1979). *Computers and intractability*, volume 174. freeman San Francisco.

Gyulai, D., Pfeiffer, A., Nick, G., Gallina, V., Sihn, W., and Monostori, L. (2018). Lead time prediction in a flow-shop environment with analytical and machine learning approaches. *IFAC-PapersOnLine*, 51(11):1029–1034.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*.

Howard, R. A. (1960). Dynamic programming and markov processes.

Jacobs, F. R. (2011). *Manufacturing planning and control for supply chain management*. McGraw-Hill, New York, apics/cpim certification ed. edition.

Kan, A. and Rinnooy, H. G. (2012). *Machine scheduling problems: classification, complexity and computations*. Springer Science & Business Media.

Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.

Kurbel, K. (2016). *Enterprise Resource Planning und Supply Chain Management in der Industrie: Von MRP bis Industrie 4.0*. De Gruyter Studium. De Gruyter, Berlin/Boston, 8., vollst. überarb. und erw. auflage edition.

Laterre, A., Fu, Y., Jabri, M. K., Cohen, A.-S., Kas, D., Hajjar, K., Dahl, T. S., Kerkeni, A., and Beguir, K. (2018). Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization.

Lödding, H. (2013). *Handbook of Manufacturing Control*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Mather, H. and Plossl, G. (1978). Priority fixation versus throughput planning. *Journal of Production and Inventory Management*, (19):27–51.

Perron, L. and Furnon, V. (2020). Or-tools.

Qu, S., Wang, J., Govil, S., and Leckie, J. O. (2016). Optimized adaptive scheduling of a manufacturing process system with multi-skill workforce and multiple machine types: An ontology-based, multi-agent reinforcement learning approach. *Procedia CIRP*, 57:55–60.

Rey, D. and Neuhäuser, M. (2011). *Wilcoxon-Signed-Rank Test*. Springer Berlin Heidelberg.

Schneckenreither, M. and Haeussler, S. (2018). Reinforcement learning methods for operations research applications: The order release problem. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 545–559.

Schuh, G., Prote, J.-P., Sauermann, F., and Franzkoch, B. (2019). Databased prediction of order-specific transition times. *CIRP Annals*, 68(1):467–470.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sotskov, Y. and Shakhlevich, N. V. (1995). Np-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59(3):237–266.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A., and Kyek, A. (2018). Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP*, 72(1):1264–1269.

Watkins, Christopher John Cornish Hellaby (1989). *Learning from delayed rewards*. King's College, Cambridge.

Zhang, W. and Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. In *Proceedings of the 14th international joint conference on Artificial intelligence-Volume 2*, pages 1114–1120.

Zhang, W. and Dietterich, T. G. (1996). High-performance job-shop scheduling with a time-delay td (łambda) network. In *Advances in neural information processing systems*, pages 1024–1030.

Zijm, H. and Regattieri, A. (2019). Manufacturing planning and control systems. In Zijm, H., Klumpp, M., Regattieri, A., and Heragu, S., editors, *Operations, Logistics and Supply Chain Management*, pages 251–271. Springer International Publishing, Cham.