5th Conference on Production Systems and Logistics

On The Effectiveness Of Bottleneck Information For Solving Job Shop Scheduling Problems Using Deep Reinforcement Learning

Constantin Waubert de Puiseau[1], Lennart Zey[2], Merve Demir[1], Hasan Tercan[1], Tobias Meisen[1]

[1]*Institute for Technologies and Management of Digital Transformation der Bergischen Universität Wuppertal , Wuppertal, Germany*
[2]*Lehrstuhl für Produktion und Logistik der Bergischen Universität Wuppertal, Wuppertal, Germany*

Abstract

Job shop scheduling problems (JSSPs) have been the subject of intense studies for decades because they are often at the core of significant industrial planning challenges and have a high optimization potential. As a result, the scientific community has developed clever heuristics to approximate optimal solutions. A prominent example is the shifting bottleneck heuristic, which iteratively identifies bottlenecks in the current schedule and uses this information to apply targeted optimization steps. In recent years, deep reinforcement learning (DRL) has gained increasing attention for solving scheduling problems in job shops and beyond. One design decision when applying DRL to JSSPs is the observation, i.e., the descriptive representation of the current problem and solution state. Interestingly, DRL solutions do not make use of explicit notions of bottlenecks that have been developed in the past when designing the observation. In this paper, we investigate ways to leverage a definition of bottlenecks inspired by the shifting bottleneck heuristic for JSSPs with DRL to increase the effectiveness and efficiency of model training. To this end, we train two different DRL base models with and without bottleneck features. However, our results indicate that previously developed bottleneck definitions neither increase training efficiency nor final model performance.

Keywords

Production Scheduling; Reinforcement Learning; Bottleneck Identification; Operations Research; Artificial Intelligence

1. Introduction

Scheduling problems occur in every sector of the manufacturing industry and often have substantial optimization potential. Industrial scheduling problems are classified as combinatorial optimization problems [1], which are prohibitively computationally expensive to solve optimally at the scale of real-world scenarios. Consequently, they have been studied for many years, especially in the fields of operations research (OR) and computer science. The result of these research efforts are numerous priority dispatching rules (PDRs) [2] and (meta-)heuristics that are still under very active development [3, 4]. All solution methods cover different spectrums in the trade-off between computation time and optimality of the found solution. Generally, PDRs are fast and interpretable, but the solutions are often more than 20% worse than the optimum [5, 6], whereas (meta-)heuristics obtain more optimal solutions at the cost of more computation time. In the last five years the application of deep reinforcement learning (DRL) has gained popularity in the scheduling domain [7, 8], driven by the increasing availability of computation and rapidly improving learning algorithms. DRL has already shown competitive results for standardized scheduling problems, such as the Job Shop Scheduling Problem (JSSP) [6, 9]. Since learning-based solutions like DRL have the inherent

DOI: https://doi.org/10.15488/15306
ISSN: 2701-6277

ability to adapt to different environments during training, the driving long-term hypothesis of researchers in the field is that DRL will lead to better results than manually tailored heuristics, require less manual effort for the mathematical description of the production environment and will generalize well to stochasticity in production lines [10, 11].

Existing DRL-based solutions can be roughly divided into two research directions. One computes information-rich, simple features for the agents' observation space, such as remaining processing times of jobs, and relies on small and shallow neural network topologies. The second trains much more elaborate neural network topologies on raw and high-dimensional problem state representations. However, neither research direction makes explicit use of the intuition and work that has been build and done over the course of many decades by the operations research (OR) community when designing observation spaces. For example, the notion of bottlenecks is very common in the OR literature and builds the foundation of one of the most cited heuristics for the JSSP, the shifting bottleneck heuristic (SBH) [12]. Yet, bottleneck features are not commonly found within DRL-based solution approaches. Therefore, in this paper, we investigate whether there is a benefit to the incorporation of bottleneck features into DRL. Building upon the definition of a bottleneck in the SBH for the design of an input feature for DRL agents, we combine established knowledge and new, powerful learning algorithms. We hypothesize that the use of the bottleneck feature leads to faster training and/or more optimal plan generation by agents, as it provides useful information about the problem instance and the current scheduling situation. The main contributions of this paper are:

- The design of a bottleneck feature inspired by the SBH that can be integrated into DRL-based solution methods.
- The analysis of the effectiveness of this feature for DRL-based scheduling methods by integrating the feature into two different existing approaches: a self-designed, comparatively small-scale DRL-based approach that relies on engineered features, and a state-of-the-art approach with an end-to-end architecture.

The remainder of this paper is structured as follows: We begin with a brief introduction to the background of this study with respect to DRL for the JSSP and definitions of bottlenecks. Next, we discuss related work with respect to bottleneck definitions and their integration into DRL-based solution approaches. In section 3, the methods and experiments are detailed, followed by the results in section 4. Section 5 concludes the study and offers perspectives for future work.

2. Background and Related Work

2.1 Deep Reinforcement Learning and Job Shop Scheduling

DRL is a machine learning paradigm, in which deep learning models are trained through interaction with an environment by sampling experience data and autonomously deriving action sequences which maximize a cumulative reward signal across a task [13]. In theory, the DRL paradigm can be applied to any sequential decision problem that can be formulated as a Markov decision process (MDP), i.e., in which a given problem state is independent of the way in which the problem state was reached. In practice, the success of the application of DRL to industrial use-cases depends on a combination of the fit of the used DRL algorithm, the neural network topology and other design decisions, such as the actions an agent can take and the reward signal, to the underlying problem. In some use cases, such as robotic control [14, 15], warehouse management [16] or load carrier control [17], DRL has already achieved impressive results. Since scheduling problems can also be formulated as MDPs, much interest has emerged in recent years to address scheduling problems with DRL as well.

In a JSSP, J jobs must be processed on M machines, where every job consists of M operations and has to be processed once on every machine in a fixed order and with different processing times. Only one job may be

processed on any machine at once and, once started, an operation may not be interrupted [1]. JSSP problem sizes are often abbreviated as JxM JSSP, meaning that a JSSP with six jobs and six machines is abbreviated as 6x6 JSSP. The objective of the combinatorial optimization often is the minimization of the makespan, i.e., the timespan between the start of the first operation and the end of the last operation in the schedule. The JSSP cannot be solved in polynomial time, making it infeasible to find exact solutions for large problem settings in acceptable time.

Considerable research effort has already been put into addressing the JSSP with DRL. Although first breakthroughs date back to 2016 [18], solutions that outperform rule-based dispatching strategies by larger margins have only recently been discovered. Therein, DRL is used in one of two ways: firstly, as stand-alone solution for constructive scheduling [5, 6, 9, 19], meaning that schedules are constructed such that Gantt charts are created from left to right. Secondly, and less frequently, DRL is used to guide improvements on already existing solutions [20, 21]. As mentioned before, a crucial part of the algorithm design is the definition of a suitable problem representation as input, called observation, for the DRL-agent. Many researchers make use of features like the remaining processing times of jobs and machines, or the number of remaining operations, which are also used by common priority dispatching rules [5, 19, 22, 23]. The most competitive observation designs, however, are raw problem descriptions of the scheduling instance and current solution state that leverage neural network topologies that capture the structure of the underlying problem through graph networks [6] or recurrent architectures [9]. Surprisingly, we found no work using more elaborate features like bottleneck information, which has proven useful in OR problem descriptions and advanced heuristics.

2.2 The Concept of Bottlenecks

In general terms, bottlenecks in production scenarios are resources or jobs which have a large impact on the final scheduling performance. Logically, it is common to identify these resources or jobs and prioritize them in one way or another. Therefore, when relying on the notion of bottlenecks, the tasks are to identify bottlenecks and to prioritize them effectively [1, 24]. For example, Zhang and Wu identified constraining bottleneck machines and jobs through statistical analysis of multiple simulations and applied this information to various genetic algorithms [25–27]. Definitions regarding resource utilization, queue lengths, and average waiting times are also common [28, 29]. To the best of our knowledge, only one DRL-based solution approach actively incorporates a bottleneck identification step. In this work, Thomas et al. propose two agents, one to classify resources as bottlenecks and the other to schedule operations [30]. However, the goal of their study was to adjust machine capacities, not to schedule on a fixed machine park. One of the most widely known effective use of the bottleneck concept is the shifting bottleneck heuristic (SBH) [12]. On an abstract level, it iterates over all machines in the problem following three steps:

1. *Identification of the bottleneck machine* of this iteration step based on the current problem state.
2. Determining the order of operations on this bottleneck machine.
3. Incorporating this order of operations on the bottleneck machine into the current problem state.

Evidently, the identification of the bottleneck machine is a central step of the SBH. For a detailed description of each step and the overall algorithm, we refer to the original publication by Adams at al. [12] and the textbook version by Pinedo [1]. Our adaption of the bottleneck identification step as feature for a DRL-agent is described in the following section 3.1.

3. Methods and Experiments

This section firstly provides details of the suggested bottleneck information. Then, the description of the self-designed and adapted base models is given, including how these models are augmented with the bottleneck information as feature, and how the models are trained.

3.1 Bottleneck Information

The bottleneck information calculation procedure we propose is closely related to the bottleneck identification step in the SBH. The abbreviated pseudo-code of the algorithm we discuss in this section is presented in Algorithm 1. The bottleneck information is calculated in every step of the DRL-based schedule generation and should provide information on the current situation. For simplicity, we refer to all non-complete solutions (including solutions without any operation scheduled) as partially solved from here on.

The algorithm starts from a partially solved schedule. In the first step, as in the SBH, we determine the lower-bound of the makespan, $makespan_{LB}$, of the current schedule. We do so under the artificial assumption that all machines can process all unscheduled operations at the same time but while still adhering to already fixed orders of operations in the schedule and to precedence constraints within jobs (see [12]). Next, from $makespan_{LB}$, we can infer pseudo release dates, being the time when operations start in the partial solution, as well as pseudo due dates that tell us when each operation would have to be finished to fulfill the lower-bound makespan.

After this preliminary calculation, we calculate the bottleneck information for each machine separately. To do so, we assume that the sequence of operations on the considered machine is not (partially) fixed. We then determine a sequence of operations minimizing the maximum tardiness on that machine, given the previously calculated release dates and due dates. In practice, we solve this one-machine tardiness minimization problem with the algorithm developed by Carlier [31]. The solution to the one-machine tardiness minimization problem returns the minimal tardiness, $tardiness_{MIN}$. It can be shown that the sum of the $makespan_{LB}$ and the $tardiness_{MIN}$ is a valid lower-bound makespan for the regarded machine [12]. As in the SBH, we interpret this makespan as an indication for the bottleneck machine, since a machine with a large lower-bound makespan is likely to have a larger influence on the final makespan. Having calculated this bottleneck information for every machine, we finally min-max scale across all bottleneck information values to the range [0, 1]. How this bottleneck information is inserted into the observation of each DRL base model is described in the respective sections 3.2.1 and 3.2.2.

Algorithm 1: Calculation of the bottleneck information list

Algorithm 1: Bottleneck Information Calculation

```
Input: current schedule;
bottleneck_info_list ← []
makespan_LB ← find lower-bound makespan
calculate pseudo due dates
for every machine do
    tardiness_MIN ← minimize tardiness of the one-machine-problem
    bottleneck_info ← makespan_LB + tardiness_MIN
    bottleneck_list.append(bottleneck_info)
end
min-max(bottleneck_info_list)
Output: bottleneck_info_list
```

3.2 Base Models and Bottleneck Feature Integration

In this paper, we experiment with two different DLR models: a self-designed model architecture, that relies on engineered features and a small neural network, as well as a large model architecture proposed by Iklassov et al. [9], that uses both raw data and features to achieve state-of-the-art results. In both cases, we modify the existing base model such that the additional bottleneck feature can be added as input. To ensure that differences in the model performances with and without the bottleneck feature can be attributed to the effect of the feature, we overwrite the bottleneck feature vector with zeros whenever the bottleneck features are not used. This way, the number of model parameters remains the same with and without the additional

bottleneck feature. All experiments were conducted on 6x6 and 10x10 JSSPs and repeated with three random seeds.

3.2.1 Self-designed Base Model

In the self-designed base model, the DRL-agent interacts with its environment by choosing from the next operations that can be scheduled per job, thereby iteratively constructing a schedule starting with the first operations of jobs. Once scheduled, the resulting starting and ending times of operations remain unchanged. The design choices, features and hyperparameters were found in extensive preliminary experiments and resulted in a stable learning behavior and competitive makespans compared to the common priority dispatching rules *most-tasks-remaining*, *shortest-processing-time-first* and randomly generated schedules. As DRL algorithm we use Proximal Policy Optimization (PPO) [32] with action masking [33] to avoid impossible action suggestions such as scheduling an operation of a finished job. The policy and value networks in PPO are feedforward neural networks with two hidden layers of size 256 and Tanh activation function. Both networks are separate networks, meaning that they do not share parameters. The used feature vector contains concatenated information on the next operations to be scheduled per job. Each one of these next operations is described by:

- its processing time
- the remaining processing time of the corresponding job
- the number of remaining operations of the corresponding job
- the remaining processing time on the required machine
- the number of remaining operations on the required machine
- the earliest starting time for the task given the current planning status

In addition, an MTR-feature is calculated, named after the common *most-tasks-remaining* dispatching rule. It is a vector of the element-wise comparison between the number of remaining operations of all jobs, where a 1, -1, and 0 indicate if there are more, less, or equal remaining operations, respectively. The bottleneck feature is also used as descriptive feature per next operation by inputting the calculated bottleneck feature of the machine processing the next operation. The data flow from problem state to the action and value predictions is depicted in Figure 1.

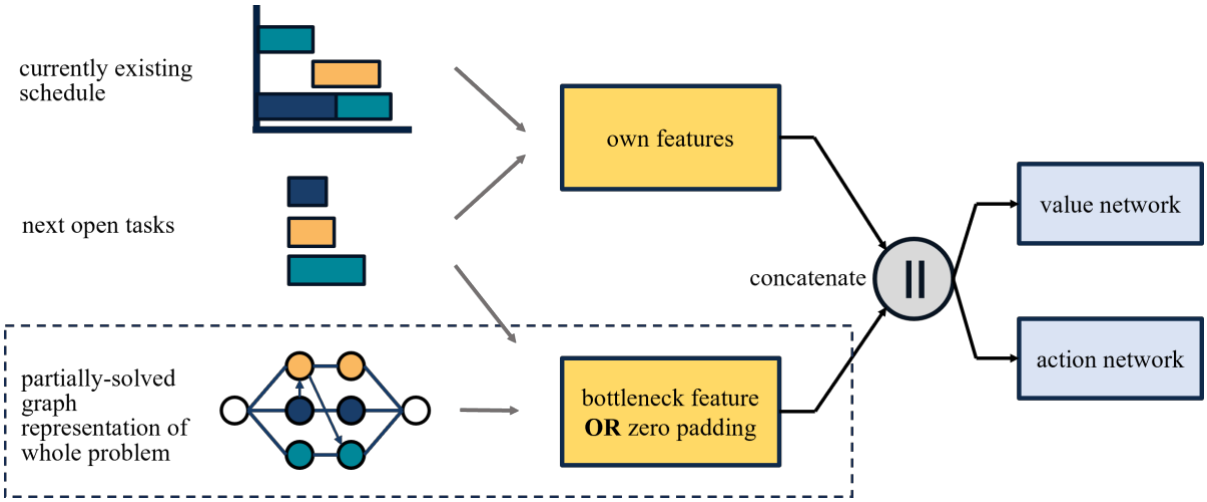

Figure 1: Data flow in the self-developed base model. Content in dashed box indicates the additions that were made in this study.

The implementation of masked PPO as well as the training and testing datasets and procedures are adapted from the *schlably* framework for DRL base production scheduling experiments [34]. The training instances were generated by randomly assigning machines to operations within a job and with operation times

determined from integers uniformly distributed in the interval [1, 11]. Training was performed on a fixed set of 1,200 training instances. The model was validated on ten separate instances every 30,000 training steps and the model that performed best on the validation data during training was finally tested on 40 test instances. Training was ended after 4,000,000 time steps for the 6x6 JSSP and 700,000 time steps for 10x10 JSSPs, as no further improvement was observed through further training in preliminary experiments. All used hyperparameters are given in Appendix Table 1.

3.2.2 Learning-To-Generalize (l2g) Base Model

Our learning-to-generalize base model that we use in this study is adapted from a recent publication by Iklassov et al. [9], which achieves state-of-the-art results for all DRL-based JSSP solutions, where the model iteratively constructs a schedule operation by operation, similarly to our approach described above (see section 3.2.1). We deploy the best reported performing configuration and hyperparameters but modify the algorithm in two ways to fit it to the objective of our experiments: First, we solely train the 6x6 JSSP model on 6x6 JSSP training instances and the 10x10 model 10x10 training instances instead of training instances of multiple sizes per model. This minimizes the required training time per model, because training on much larger instances increases the calculation time significantly, due to the calculation time of the bottleneck feature that scales faster than quadratically with increasing problem sizes. We accept here that this may lead to a worse overall performance of the models, since our primary interest lies in the comparison between models trained with and without the bottleneck feature. The second modification is with respect to the input features. The bottleneck features are added as a fourth dynamic feature category next to the information about the last processed operations per job, the machine status and remaining processing times, as illustrated in Figure 2. To this end, the bottleneck features, like the other dynamic features, are expanded to the embedding size 128 through a linear layer, concatenated with the static input embedding and passed through a recurrent set2set model [35], before entering the actor and critic networks.

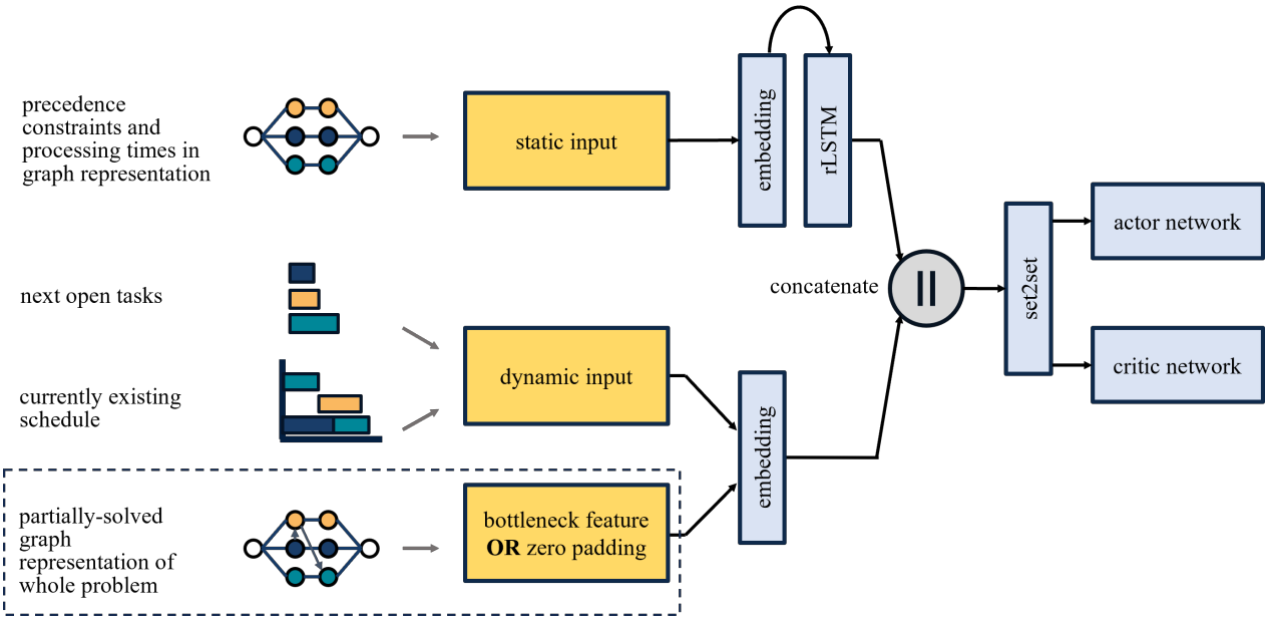

Figure 2: Data flow in the l2g base model. Content in dashed box is the addition that was made for this study.

Adhering closely to the original training procedure, all instances consist of operations with processing times sampled uniformly from the interval [0, 100]. This has the benefit that the bottleneck features is not only tested on multiple base models but varying JSSP settings. For every episode, new training instances are randomly created. Evaluations during training are carried out every 100 episodes on ten fixed evaluation instances. Finally, the best performing model on the evaluation instances is tested on separate test instances.

Training was stopped after 2,800 episodes and 1,100 episodes for the 6x6 and 10x10 JSSPs, respectively, because no further improvement was observed after this point in preliminary experiments.

4. Results

We first analyze the training behavior of the DRL agents to verify the working hypothesis that the bottleneck feature is a useful addition to the observation space. To this end, the achieved makespans on the evaluation instances are plotted over the number of instances seen during training. Every plot in Figure 3 shows the evaluation makespans averaged across three seeds for models trained with the bottleneck feature (blue) and without the bottleneck feature (orange) as solid lines. Shaded areas represent the standard deviation across the three seeded training runs per model. Generally, the evaluation results drop fast in the beginning and then plateau after a few hundred training instances. Critically, we cannot identify significantly faster learning, nor significantly lower plateaus, nor other striking differences between any of the two lines per plot. Therefore, the bottleneck feature does not seem to qualitatively change the learning behavior, which falsifies our working hypothesis.

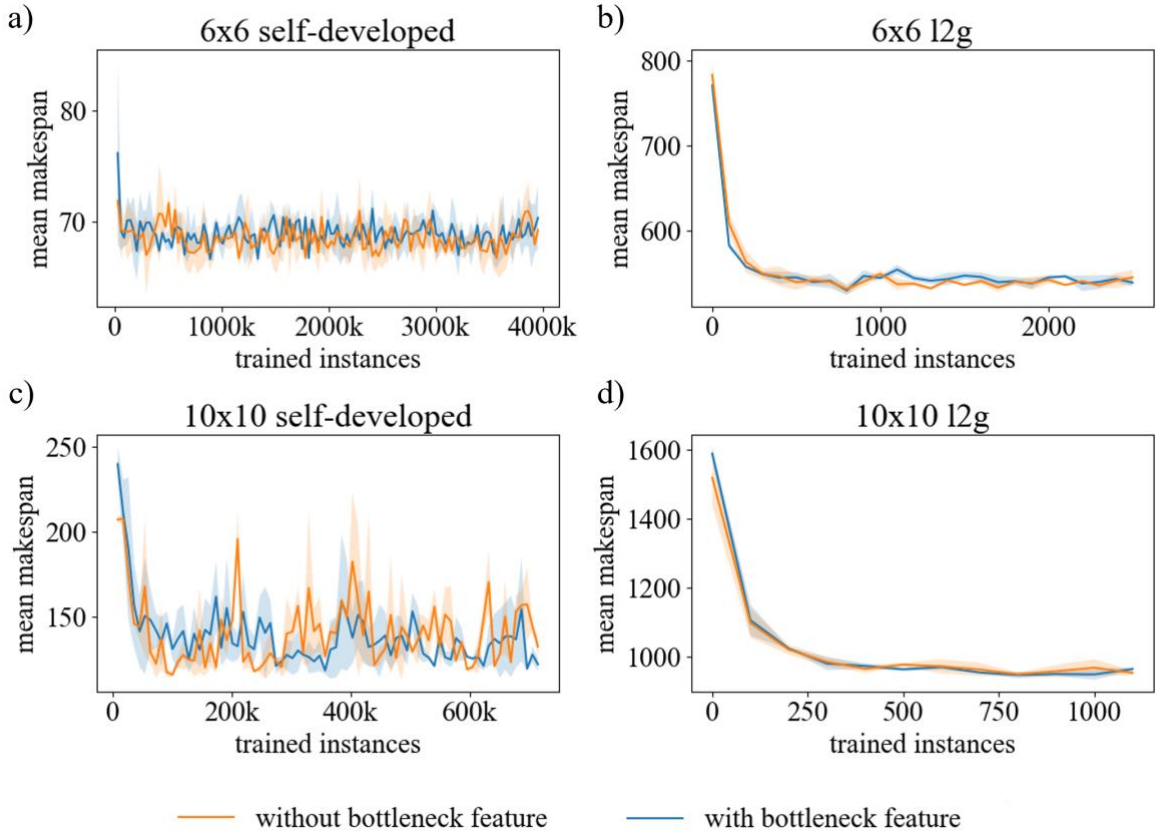

Figure 3: Learning curves of evaluation runs, averaged across seeds. a) Self-developed base model on 6x6 JSSP; b) L2g base model on 6x6 JSSP; c) Self-developed base model on 10x10 JSSP; d) L2g base model on 10x10 JSSP

The second part of the analysis aims at the evaluation of the performance of the final trained models. The averaged results across the three training seeds on the test instances are reported in Table 1. The column *optimal* represents the average makespans of optimal solutions that were calculated using the CP-SAT solver of the OR Tools library [36]. The percentual gap to the optimum is reported in the column *opt.-gap*. The other columns show the average makespans achieved through common PDRs: *MTR*, "most-tasks-remaining", always selects the next unscheduled operation that belongs to the job with the most unscheduled operations, with ties resolved by the smallest assigned job id. *SPT*, "shortest-processing-time", selects the next unscheduled operation with the shortest processing time, with ties resolved as before. *Random* chooses the next job of which the next operation will be scheduled by sampling the job id from a uniform random

distribution. Comparing the PDR results with the results of the agent, all agents except the ones using the self-designed base on the 10x10 JSSP clearly outperform the PDRs. The critical results with respect to our working hypothesis are the differences between the model performances with and without the bottleneck feature. Here, we find that the resulting makespans are very similar and that those models trained without the bottleneck feature even sometimes outperform those trained with the bottleneck feature. The results of individual models can be found in Appendix B.

Table 1: Summary of average test results of the trained models

base model	JSSP size	bottlenck-feature	agent	optimal	opt.-gap	MTR	SPT	Random
self-designed	6x6	yes	69.5 ± 2.1	59	17.8 %	72.1	140.7	98.9
		no	68.7 ± 2		16.4 %			
	10x10	yes	123.7 ± 5.1	96.9	27.5 %	117.2	320.1	165.3
		no	123.9 ± 2.5		27.8 %			
l2g based	6x6	yes	543.7 ± 1.4	494.8	9.9 %	602.2	1209.6	828.625
		no	542.0 ± 3.5		9.5 %			
	10x10	yes	962.2 ± 3.9	827.65	16.3 %	1094.6	2910.0	1599.3
		no	963 ± 2.5		16.4 %			

The experiments were designed to indicate whether the bottleneck feature carries more useful information than zeros. Considering the above-presented results, the additional non-zero features seem to introduce no noise to the experience data rather than to provide useful information for the learning task. It should also be noted that both, the inclusion of the bottleneck feature and that of zeros, had a small detrimental effect on the final agents' performance when compared to models without such extensions. This shows that more seemingly meaningful features and parameters do not necessarily improve model performance. On the contrary, features with little or no informative value lead to worse results. This has been previously mentioned in other studies regarding DRL based scheduling [5, 19], but has never explicitly been shown for the JSSP.

5. Conclusion and Future Work

This paper presented a study on the integration of a novel shifting bottleneck heuristic-inspired feature into the observation space of DRL agents solving JSSPs. The training behavior and final performance of two different solution approaches, both with and without the new feature, were analyzed and compared. From the results, we conclude that the inclusion of the bottleneck feature does not improve the training procedure, but rather leads to slightly worse final performances. In the light of these results and the considerably prolonged training times resulting from the calculation of the feature in every step of the solution generation, it does not seem like a valuable addition to existing approaches. Yet, we believe that there is unexploited potential in established algorithms and intuitions developed by the OR community over the course of decades, which have received enough attention by DRL based solutions. Some work in this area has already shown great success on combinatorial problems [20, 37, 38]. In future work we aim to identify such potentials in local search algorithms for scheduling problems and build upon the underlying insights to optimize given schedules with DRL.

Appendix

Appendix A: Hyperparameters in Accordance with the Implementation in the *schlably* Framework [34]

Appendix Table 1: Hyperparameters in the self-implemented approach

Parameter	Value
Algorithm	PPO-masked
Clip-Range	0.2
Batch size	256
Ent_coef	0
Gamma	1
Learning rate	0.02
N_epochs	5
Policy_activation	Tanh
Policy_layer	[256, 256]
Value_activation	Tanh
Value_layer	[256, 256]
Rollout_scale	2048

Appendix B: Detailed Test Results

Appendix Table 2: Test results of the self-developed base model with and without the bottleneck feature on 6x6 JSSP instances

	seeds	agent	optimal	opt. - gap	MTR	SPT	Random
with bottleneck feature	1	72.13	61.03	18 %	75.50	138.20	101.30
	2	69.28	58.93	18 %	71.50	144.05	100.98
	3	67.05	57.05	18 %	69.15	139.98	94.50
	average	69.48	59.00	**18 %**	72.05	140.74	98.93
without bottleneck feature	1	71.43	61.03	17 %	75.50	138.20	101.30
	2	68.15	58.93	16 %	71.50	144.05	100.98
	3	66.53	57.05	17 %	69.15	139.98	94.50
	average	68.70	59.00	**16 %**	72.05	140.74	98.93

Appendix Table 3: Test results of the self-developed base model with and without the bottleneck feature on 10x10 JSSP instances

	seeds	agent	optimal	opt. - gap	MTR	SPT	Random
with bottleneck feature	1	116.6	95.5	22.1 %	115.5	318.4	159.8
	2	126.1	98.0	28.7 %	118.9	320.2	168.6
	3	128.3	97.3	31.9 %	117.2	321.7	167.6
	average	123.7	96.9	**27.5 %**	117.2	320.1	165.3
without bottleneck feature	1	120.4	95.5	26.1 %	115.5	318.4	159.8
	2	125	98	27.6 %	118.9	320.2	168.6
	3	126.2	97.3	29.7 %	117.2	321.7	167.6
	average	123.9		**27.8 %**	117.2	320.1	165.3

Appendix Table 4: Test results of the l2g base model with and without the bottleneck feature on 6x6 JSSP instances

	seeds	agent	optimal	opt. - gap	MTR	SPT	Random
with bottleneck feature	1	541.7	494.8	9.5 %	602.2	1209.6	828.6
	2	544.3	494.8	10.0 %	602.2	1209.6	828.6
	3	545.1	494.8	10.2 %	602.2	1209.6	828.6
	average	543.7	494.8	**9.9 %**	602.2	1209.6	828.6
without bottleneck feature	1	540.0	494.8	9.1 %	602.2	1209.6	828.6
	2	547.0	494.8	10.5 %	602.2	1209.6	828.6
	3	539.2	494.8	9.0 %	602.2	1209.6	828.6
	average	542.0	494.8	**9.6 %**	602.2	1209.6	828.6

Appendix Table 5: Test results of the l2g base model with and without the bottleneck feature on 10x10 JSSP instances

	seeds	agent	optimal	opt. - gap	MTR	SPT	Random
with bottleneck feature	1	965.5	827.7	16.7 %	1094.6	2910.0	1599.3
	2	956.7	827.7	15.6 %	1094.6	2910.0	1599.3
	3	964.5	827.7	16.5 %	1094.6	2910.0	1599.3
	average	962.2	827.7	**16.3 %**	1094.6	2910.0	1599.3
without bottleneck feature	1	962.9	827.7	16.3 %	1094.6	2910.0	1599.3
	2	960.0	827.7	16.0 %	1094.6	2910.0	1599.3
	3	966.1	827.7	16.7 %	1094.6	2910.0	1599.3
	average	963.0	827.7	**16.4 %**	1094.6	2910.0	1599.3

References

[1] Pinedo, M., 2012. Scheduling: Theory algorithms and systems, 4th ed. Springer, New York.

[2] Holthaus, O., Rajendran, C., 1997. Efficient dispatching rules for scheduling in a job shop. International Journal of Production Economics 48 (1), 87–105.

[3] Luo, X., Qian, Q., Fu, Y.F., 2020. Improved Genetic Algorithm for Solving Flexible Job Shop Scheduling Problem. Procedia Computer Science 166, 480–485.

[4] Leonor Hernández-Ramírez, Juan Frausto Solis, Guadalupe Castilla-Valdez, Juan Javier González-Barbosa, David Terán-Villanueva, María Lucila Morales-Rodríguez, 2019. A Hybrid Simulated Annealing for Job Shop Scheduling Problem. International Journal of Combinatorial Optimization Problems and Informatics 10 (1), 6–15.

[5] van Ekeris, T., Meyes, R., Meisen, T., 2021. Discovering Heuristics And Metaheuristics For Job Shop Scheduling From Scratch Via Deep Reinforcement Learning. Proceedings of the Conference on Production Systems and Logistics: CPSL2021, 709–718.

[6] Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P.S., Xu, C., 2020. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. 34th Conference on Neural Information Processing Systems.

[7] Panzer, M., Bender, B., Gronau, N., 2021. Deep Reinforcement Learning In Production Planning And Control: A Systematic Literature Review. Proceedings of the Conference on Production Systems and Logistics: CPSL2021, 535–545.

[8] Khadivi, M., Charter, T., Yaghoubi, M., Jalayer, M., Ahang, M., Shojaeinasab, A., Najjaran, H., 2023. Deep Reinforcement Learning for Machine Scheduling: Methodology, the State-of-The-Art, and Future Directions. SSRN Journal.

[9] Iklassov, Z., Medvedev, D., Solozabal, R., Takac, M., 2022. Learning to generalize Dispatching rules on the Job Shop Scheduling. https://arxiv.org/pdf/2206.04423.

[10] Waubert de Puiseau, C., Meyes, R., Meisen, T., 2022. On reliability of reinforcement learning based production scheduling systems: a comparative survey. Journal of Intelligent Manufacturing 33 (4), 911–927.

[11] Rinciog, A., Meyer, A., 2022. Towards Standardising Reinforcement Learning Approaches for Production Scheduling Problems. Procedia CIRP 107, 1112–1119.

[12] Adams, J., Balas, E., Zawack, D., 1988. The Shifting Bottleneck Procedure for Job Shop Scheduling. Management Science 34 (3), 391–401.

[13] Sutton, R.S., Barto, A., 2018. Reinforcement learning: An introduction, 2nd ed. The MIT Press, Cambridge, Massachusetts, London, England.

[14] Gomes, N.M., Martins, F.N., Lima, J., Wörtche, H., 2022. Reinforcement Learning for Collaborative Robots Pick-and-Place Applications: A Case Study. Automation 3 (1), 223–241.

[15] Han, D., Mulyana, B., Stankovic, V., Cheng, S., 2023. A Survey on Deep Reinforcement Learning Algorithms for Robotic Manipulation. Sensors (Basel, Switzerland) 23 (7).

[16] Waubert de Puiseau, C., Nanfack, D.T., Tercan, H., Löbbert-Plattfaut, J., Meisen, T., 2022. Dynamic Storage Location Assignment in Warehouses Using Deep Reinforcement Learning. Technologies 10 (6), 129.

[17] Hadwiger, S., Lavrik, V., Liao, L.X., Meisen, T., 2023. Simulation-to-Reality Transfer of a Two-Stage Deep Reinforcement Learning Controller for Autonomous Load Carrier Approaching. Proceedings of the 2023 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), 232–238.

[18] Bello, I., Pham, H., Le V, Q., Norouzi, M., Bengio, S., 2016. Neural Combinatorial Optimization with Reinforcement Learning. https://arxiv.org/pdf/1611.09940.

[19] Kuhnle, A., 2020. Adaptive Order Dispatching based on Reinforcement Learning: Application in a Complex Job Shop in the Semiconductor Industry. Shaker Verlag.

[20] Falkner, J.K., Thyssens, D., Bdeir, A., Schmidt-Thieme, L., 2023. Learning to Control Local Search for Combinatorial Optimization, in: . Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, Cham, pp. 361–376.

[21] Chen, X., Tian, Y., 2018. Learning to Perform Local Rewriting for Combinatorial Optimization. https://arxiv.org/pdf/1810.00337.

[22] Luo, S., 2020. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. Applied Soft Computing 91, 106208.

[23] Altenmüller, T., Stüker, T., Waschneck, B., Kuhnle, A., Lanza, G., 2020. Reinforcement learning for an intelligent and autonomous production control of complex job-shops under time constraints. Production Engineering 14 (3), 319–328.

[24] Wang, Y., Zhao, Q., Zheng, D., 2005. Bottlenecks in production networks: An overview. Journal of Systems Science and Systems Engineering 14 (3), 347–363.

[25] R. Zhang, C. Wu. Bottleneck Machine Identification for Shop Scheduling Problems. 2008 3rd International Conference on Innovative Computing Information and Control 2008, 149.

[26] R. Zhang, C. Wu, 2008. A simulation-based approach to job shop scheduling with bottlenecks, in: 2008 6th IEEE International Conference on Industrial Informatics. 2008 6th IEEE International Conference on Industrial Informatics, pp. 1007–1012.

[27] Rui Zhang, Cheng Wu, 2008. An effective immune particle swarm optimization algorithm for scheduling job shops, in: 2008 3rd IEEE Conference on Industrial Electronics and Applications. 2008 3rd IEEE Conference on Industrial Electronics and Applications, pp. 758–763.

[28] Pollett, P., 2000. Modelling congestion in closed queueing networks. International Transactions in Operational Research 7 (4-5), 319–330.

[29] Law, A.M., Kelton, W.D., 1991. Simulation modmeling and analysis, 2nd ed. Mc Graw-Hill, New York.

[30] Thomas, T.E., Koo, J., Chaterji, S., Bagchi, S., 2018. Minerva: A reinforcement learning-based technique for optimal scheduling and bottleneck detection in distributed factory operations. Proceedings of the 10th International Conference on Communication Systems & Networks, 129–136.

[31] Carlier, J., 1982. The one-machine sequencing problem. European Journal of Operational Research 11 (1), 42–47.

[32] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal Policy Optimization Algorithms. https://arxiv.org/pdf/1707.06347.

[33] Huang, S., Ontañón, S., 2022. A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. Proceedings of the Thirty-Fifth International Florida Artificial Intelligence Research Society Conference, 35.

[34] Waubert de Puiseau, C., Peters, J., Dörpelkus, C., Tercan, H., Meisen, T., 2023. schlably: A Python framework for deep reinforcement learning based scheduling experiments. SoftwareX 22, 101383.

[35] Vinyals, O., Bengio, S., Kudlur, M., 2016. Order Matters: Sequence to sequence for sets. https://arxiv.org/abs/1511.06391

[36] Perron, L., Furnon, V., 2023. Or-Tools. Google.

[37] Hottung, A., Tierney, K., 2020. Neural Large Neighborhood Search for the Capacitated Vehicle Routing Problem. Proceedings of the 24th European Conference on Artificial Intelligence (ECAI2020).

[38] Wu, Y., Song, W., Cao, Z., Zhang, J., Lim, A., 2022. Learning Improvement Heuristics for Solving Routing Problems. IEEE Transactions on Neural Networks and Learning Systems 33 (9), 5057–5069.

Biography

Constantin Waubert de Puiseau (*1994) holds a master's degree from RWTH Aachen University in Mechanical Engineering. Since 2019, he works at the Institute for Digital Transformation Technologies and Management at the University of Wuppertal. His research focuses on solving real-world planning and scheduling problems with deep reinforcement learning.

Lennart Zey (*1987) holds a doctorate and is a scientific researcher at the chair of production and logistics at the University of Wuppertal since 2014. His research focuses on scheduling algorithms, meta heuristics, and deep reinforcement learning.

Merve Demir (*1994) holds a master's degree from Erzurum Ataturk University in Industrial Engineering. She received a scholarship from the Türkiye Ministry of National Education for her doctoral study. Since 2022, she works at the Institute for Digital Transformation Technologies and Management at the University of Wuppertal. Her research focuses on solving planning and scheduling problems with deep reinforcement learning.

Hasan Tercan (*1988) holds a master's degree from TU Darmstadt in Computer Science. Since 2018, he is a scientific researcher at the Institute for Technologies and Management of Digital Transformation and leader of the research group Industrial Deep Learning. In his research, he investigates the application of machine learning methods in industrial processes.

Tobias Meisen (*1981) is a professor for Technologies and Management of Digital Transformation at the University of Wuppertal since 2018. He is also the institute director of the In-Institute for Systems Research in Information, Communication and Media Technology, the chair of the Interdisciplinary Centre for Data Analytics and Machine Learning.