1st Conference on Production Systems and Logistics

Sheet-Metal Production Scheduling Using AlphaGo Zero

Alexandru Rinciog[1], Carina Mieth[2], Paul Maria Scheikl[3], Anne Meyer[1]

[1]*Chair of Enterprise Logistics (LFO), TU Dortmund Univerity, Germany*
[2]*TRUMPF GmbH + Co. KG, Ditzingen, Germany*
[3]*Institute for Intelligent Process Automation and Robotics (IPR), Karlsruhe Institute of Technology, Germany*

Abstract

This work investigates the applicability of a reinforcement learning (RL) approach, specifically AlphaGo Zero (AZ), for optimizing sheet-metal (SM) production schedules with respect to tardiness and material waste. SM production scheduling is a complex job shop scheduling problem (JSSP) with dynamic operation times, routing flexibility and supplementary constraints. SM production systems are capable of processing a large number of highly heterogeneous jobs simultaneously. While very large relative to the JSSP literature, the SM-JSSP instances investigated in this work are small relative to the SM production reality. Given the high dimensionality of the SM-JSSP, computation of an optimal schedule is not tractable. Simple heuristic solutions often deliver bad results. We use AZ to selectively search the solution space. To this end, a single player AZ version is pretrained using supervised learning on schedules generated by a heuristic, fine-tuned using RL and evaluated through comparison with a heuristic baseline and Monte Carlo Tree Search. It will be shown that AZ outperforms the other approaches. The work's scientific contribution is twofold: On the one hand, a novel scheduling problem is formalized such that it can be tackled using RL approaches. On the other hand, it is proved that AZ can be successfully modified to provide a solution for the problem at hand, whereby a new line of research into real-world applications of AZ is opened.

Keywords

Production Scheduling; Sheet-Metal Production; Job Shop Scheduling Problem; Reinforcement Learning; Monte Carlo Tree Search; AlphaGo Zero

1. Introduction

1.1 Motivation

The reasons for focusing on sheet-metal (SM) production scheduling are twofold. On the one hand, SM products are ubiquitous. The spectrum of SM products is very large, ranging from industrial to household items. As such, successfully optimizing the production process would be of high impact. On the other hand, the problem is only summarily studied [1], [2] and of considerable difficulty.

The input to a sheet-metal production system is a stream of product specifications, where each product has a variable number of associated constituent parts, a monetary value and a deadline. To complete a product, parts are batched onto a metal sheet from which they are then separated, bent into three-dimensional shapes and assembled together. Parallel processing resources are available for each step. During batching, material waste occurs. If deadlines are missed, the product value is penalized. The goal of a production scheduler is to map operations to resources, such that both tardiness and material waste is minimized.

SM production scheduling is most closely related to the job shop scheduling problem (JSSP) with routing [3], [4], albeit considerably more complex and presenting a unique challenge: Since cutting operations occur in dynamic batches, whose processing times are only available after batch definition, scheduling is interwoven with a two-dimensional packing problem. Additionally, SM production systems are capable of processing a large number of highly heterogeneous parts simultaneously. Mapping the SM production reality to a JSSP would lead to far larger instances than any available in benchmark data sets [5].

Commonly, high dimensional JSSPs, are solved by selectively searching the space of all possible schedules using mainly local search and genetic algorithms (GA) [6]–[10]. More recently, reinforcement learning (RL) approaches for solving JSSPs started to gain momentum. This is because RL presents some advantages over GA/search and other approaches: As per Waschneck et al. [11], global transparency and global optimization are among the advantages offered by RL solutions. Global transparency pertains to scheduling systems' ability to monitor the production state as a whole, as opposed to merging several local optimization strategies. Global optimization describes the system's capability to jointly optimize different goals. Given the interdependence between batching and scheduling, both properties are very desirable for SM production scheduling. AZ has the added advantage of incorporating search with RL, thereby alleviating the sample inefficiency problem deep RL approaches can suffer from [12].

1.2 Related Work

Reinforcement Learning: RL has been applied to small JSSPs with varying optimization targets and degrees of success since the mid-90s [13]–[19]. More recently, Reyna et al. [20] uses a value based Q-learning approach on a simple JSSP with makespan as the optimization target. The RL solutions are tested against optimal solutions from the OR-Library benchmark data set (up to 20 jobs and 5 machines). Qu et al. investigate the adequacy of multi-agent Q-learning on top of an ontology in the context of a multi-objective dynamic flow shop scheduling problem [21]. Waschneck et al. embed the Atari deep Q-learning algorithm [22] in a multi-agent system aimed at optimizing the uptime utilization in the semiconductor industry [11]. The trained system fails to outperform the heuristic baseline. In [23], [24], policy learning, namely Trust Region Policy Optimization [25] is used to solve the complex JSSP of the semiconductor industry. Resource utilization and lead time provide the concomitant optimization goal. The RL algorithm surpasses a heuristic approach on both optimization targets.

Supervised Learning: Another AI solution to complex scheduling problems is offered by supervised learning (SL) as investigated in [26]. The authors use a data set created by human demonstrators and a synthetic data set created using heuristics for binary classifier training. The binary classifiers are used to discriminate high from low priority scheduling tasks given the two tasks and the scheduling problem's state. Using the trained models, tasks are ranked through pairwise comparison. The approach achieves a high task assignment accuracy of over 90%. The most recent supervised learning approach to solving the JSSP w.r.t. makespan was taken in [27]. Here, the authors generate optimal solutions for a 6 machines and 6 jobs JSSP. They then train a linear classifier to learn a dispatching rule given the production state. While the results obtained through SL are proved to be superior to heuristics, generating training data for large JSSPs is intractable.

AlphaGo Zero: The first version of AlphaGo, introduced in [28], combines SL, RL and a probabilistic search strategy, namely Monte Carlo Tree Search (MCTS) [29]. Through SL, human expertise is incorporated in a neural network (NN). Through self-play using RL and MCTS, the NN is further tuned until it outperforms human players at Go. The second version [30], named AlphaGo Zero (AZ), simplifies both the self-play algorithm and the training process considerably: Here, the algorithm relies solely on RL for training. In [31], the authors use a slightly modified version of AZ, to outperform the respective state of the art for Chess and Shogi, in addition to Go. This constitutes proof of AZ's game agnosticism.

1.3 This Work

The SM scheduling problem can be formalized as a JSSP with routing flexibility, cutting constraints, assembly constraints, (dynamic) batch processing capabilities and an additional optimization target. Jobs correspond to SM parts and consist of 3 operations: cutting, bending and assembly. The set of jobs is divided into partitions corresponding to SM products. In SM-JSSPs, operations are fixed to a machine type, whereby routing flexibility ensues. Standard JSSPs limit machines to processing one operation at a time in the absence of preemption. This applies solely to bending machines in the SM-JSSP context. Cutting operations can be processed simultaneously, as long as the associated parts fit on the same sheet of metal (*cutting constraints*). Assembly operations of jobs from the same partition must be processed together (*assembly constraints*). While the processing times for bending and assembly operations are known a priori, the duration of cutting operations is only known post the dynamic batch definition, i.e. during schedule computation. Figure 1 provides a representation of the main aspects of the SM-JSSP. Besides the common JSSP optimization targets (e.g. makespan, flow/lead time, tardiness [32]), the minimization of material waste incurred during cutting is additionally considered in SM-JSSPs (*additional target*).

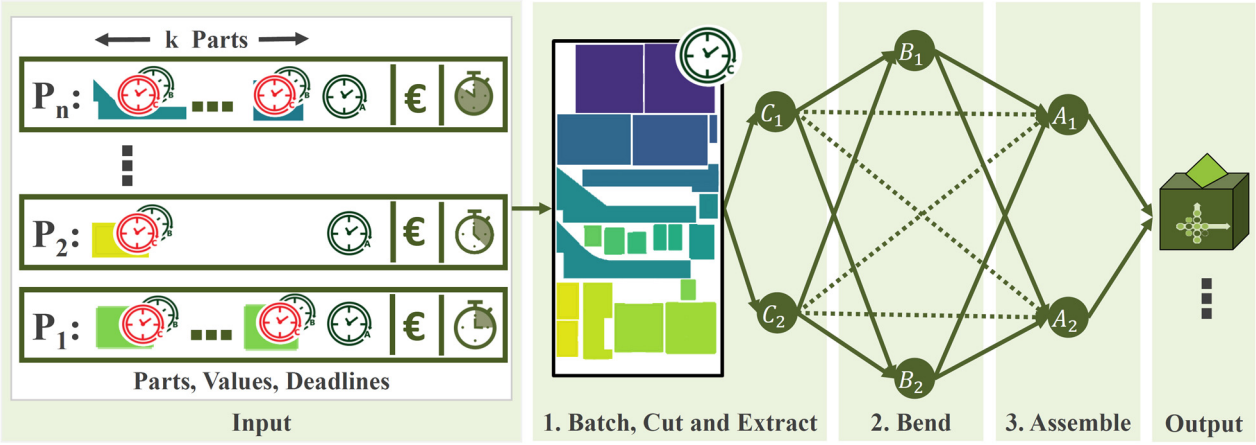


Figure 1: A representation of the SM-JSSP. A stream of products with variable number of parts, a monetary value and a deadline are the SM-JSSP input. Processing starts with batching parts onto a metal sheet. After cutting and extracting the parts, bending operations are executed. All parts from a product are assembled together in a final step. Bending and assembly times can be computed a priori per part and per product respectively. Cutting times vary depending on the batch content.

This work focuses on jointly minimizing tardiness and material waste for medium-sized, offline SM-JSSP instances using AZ. Since this is a preliminary study, we defer some supplementary sources of complexity to future work. For now, we do not consider setup times, machine/worker availability and transportation times. Furthermore, we flatten the two dimensional packing problem into one dimension, i.e. the cutting constraints now simply state that the summed areas of all parts batched on a sheet must be smaller than the sheets' area. We also limit ourselves to considering SM-JSSP instances, which, while large with respect to the JSSP literature, are not large enough to accommodate the larger SM production systems. The exact instance size is presented in Section 2.3.

In SM production, despite advanced planning systems (APS) being used for coarsely defining production plans, daily scheduling decisions on the shop floor are often done by human experts using simple heuristic solutions such as earliest due date (EDD) to prioritize operations. This is because APS plans can quickly become obsolete given unforeseeable events. The solution we investigate as an alternative involves modifying AZ to a single player version, training its integrated NN in a supervised fashion on heuristic solutions to offline SM-JSSP instances and fine-tuning the NN through self-play. Our scientific contribution is twofold. First, we design a new state formalism for a complex new scheduling problem and integrate it in an environment, which can be used with RL methods. Secondly, we show AZ to be able to incorporate and

outperform a heuristic approach dominated by the EDD. Priority ties are broken by supplementary criteria (see Section 2.3). MCTS was added as a supplementary comparison baseline.

This paper is structured as follows: Section 2 presents the inner workings of AZ in Subsection 2.1, followed by the RL design for SM-JSSP in Subsection 2.2, as well as our experiment setup in Subsection 2.3. After a brief discussion of our results in Section 3, we present our conclusion in Section 4.

2. Methods

2.1 AlphaGo Zero

The data structure central to AZ is a tree with nodes corresponding to game states s and edges corresponding to actions a. Each edge in the tree stores the probability $P(s,a)$ of it being the best action in the state s a visit count $N(s,a)$ and the average expected result $Q(s,a)$ of taking action a in state s. Nodes store the expected result over all actions possible from state s as VS Below we describe how moves are selected using NN and MCTS, whereupon the NN training scheme is detailed.

Neural MCTS: Given a state s AZ picks a move repeating four phases m times followed by a final move selection. The four phases are selection, expansion, evaluation and backup as shown in Figure 2. These correspond to the MCTS phases modified to be guided by a NN. During the selection phase (Figure 2a) nodes reached over edges maximizing $Q(s,a)+U(s,a)$ where

$$U(s,a)=P(s,a)c_{puct}\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}\propto\frac{P(s,a)}{1+N(s,a)} \tag{1}$$

are selected until a dangling edge is encountered. In the Equation (1), $\sum_b N(s,b)$ is the cumulative visit count of all outgoing edges from s and c_{puct} is a tunable exploration parameter. At first, actions with high probability and low visit count are preferred. Asymptotically, high valued actions are preferred. By increasing c_{puct} this asymptotic transition is slowed down. Assuming the current iteration is n and s_L^i is the leaf node reached in iteration i the current leaf node s_L^n reachable over the selected dangling edge, as well as its egress edges are added to the tree. This constitutes the expansion phase. During the evaluation phase, the NN f_θ is used to evaluate the new node $s_L^n : P(s_L^n,\cdot),V(s_L^n)) \coloneqq f_\theta(s_L^n)$. First $P(s_L^n,\cdot)$ is used to update the egress edges from s_L^n (Figure 2b). Then the backup phase ensues (Figure 2c). Herein, all edges up the selection path are updated. This update implies incrementing the edge visit counts $N(s,a)$ and setting their values $Q(s,a)$ to the average accumulated value up to the current iteration. This is described by Equation (2), where $1(s,a,i)$ is 1 if the s_L^i is reachable over the edge (s,a) and 0 otherwise:

$$Q(s,a)=\frac{1}{N(s,a)}\sum_{i=1}^{n}1(s,a,i)V(s_L^i). \tag{2}$$

After the predetermined number of MCTS iterations the *final move selection* is performed. The visit counts of the root edges are exponentiated with $1/\tau$, where τ is an exploration parameter, and normalized to create a probability distribution over the legal actions relative to the root node. The final move is then selected by sampling from this distribution. Note that infinitesimal values of τ induce a Dirac distribution, while high values of τ induce an asymptotically uniform distribution.

AZ Training Loop: The NN needs to be trained to estimate the move probabilities $P(s,\cdot)$ and the expected value of a node $V(s)$ accurately. AZ training has two steps, which are repeated until the model weights saturate. These are self-play and neural-network training. During self-play (Figure 3a), the MCTS scheme introduced above is used to play games from start to finish, i.e. iterations 1 to t by sampling from probability

distributions π_i returned by the MCTS algorithm for states s_i States and the corresponding MCTS action probabilities (s_i, π_i) are stored for every performed move. When a terminal state s_T is reached (i.e. end of the game), the reward $z \coloneqq r(s_T)$ is used to form triples (s_i, π_i, z). The values of z depend on how the reward function r is modelled, e.g. for chess, z is either -1, 0 or 1 for loss, draw or win respectively, for Go $z = \pm 1$. The AZ network can now be trained on them using stochastic gradient descent to minimize the loss function $l = (z - v)^2 - \pi^T \log p$ where $(p, v) = f_\theta(s)$ (Figure 3b).

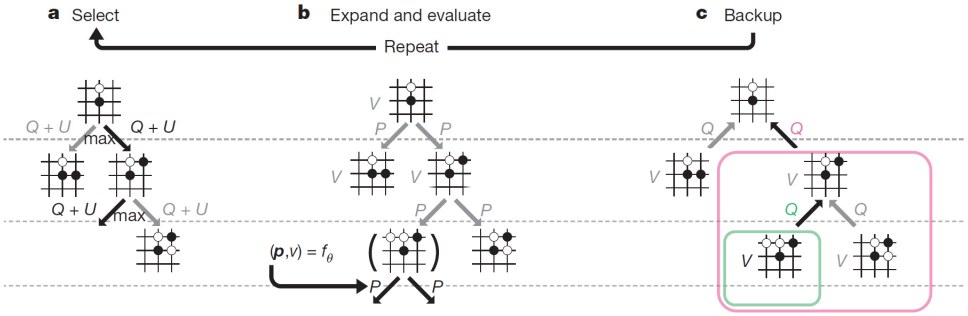


Figure 2: Neural MCTS: a Nodes are selected recursively by traversing edges corresponding to an action $a = argmax_a Q(s,a) + U(s,a)$ until the egress edge of a leaf node; b A new node s_L is added to the tree, $P(s,\cdot),V(s_L)) \coloneqq f_\theta(s_L)$ is evaluated and its egress edges are updated with probabilities $P(s,\cdot)$. c Action values Q are updated up the tree path using the mean of all state values V stored in the nodes.
Source: [30]

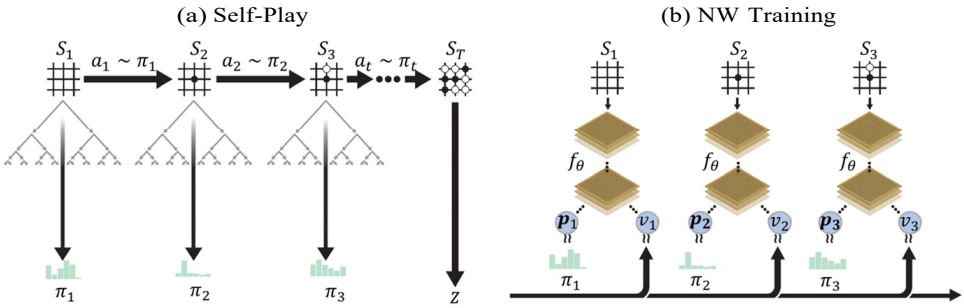


Figure 3: (a) Games are played from start to finish, using neural MCTS to select a move. The move probabilities π_i are stored together with the corresponding state s_i. When the end-state s_T is reached, the reward z is associated with every pair (s_i, π_i). (b) The network f_θ is trained using stochastic gradient descent to minimize a combination of mean squared error on the value head and cross entropy on the policy head. Source: [30].

2.2 RL Design for SM-JSSP

To apply AZ to the SM-JSSP, the agent environment interaction needs to be redesigned. This is done by creating the SM-JSSP state and action space together with a reward signal corresponding to the joint objective function of material waste and tardiness. Additionally, the NN architecture is adapted.

The following state space was used for SM-JSSP instances of m_1 cutting, m_2 bending and m_3 assembly stations and up to nk jobs. Processing stations are indexed using the set $M \coloneqq \{1 \dots m_1 + m_2 + m_3\}$, where 1 to m_1 correspond to cutting machines, $m_1 + 1$ to $m_1 + m_2$ correspond to bending machines and so on. Jobs are represented through the indices of a nxk matrix, where every row corresponds to a job partition. To each job we associate 2 descriptors: Area and outline. Both are needed to compute the dynamic processing times for the cutting (and extraction) operations. Areas are also needed to enforce area constraints. The material waste is stored per cutting machine in a vector of length m_1. Operation processing times are stored per job -

bending -, job partition - assembly - or per cutting machine - cutting -. To track remaining operations, for every job, the index $i \in M$ of the last machine a contained operation was assigned to, is noted down. Additionally, the state encodes whether the current machine i is actively processing the corresponding operation (1) or whether the operation has finished (2) for every job. The state also encodes the remaining slack time and value for every job partition. Finally yet importantly, the idle machine onto which an operation needs to be scheduled next is encoded. This yields a state space of $5nk + 3n + 2m_1 + 1$ entries.

The AZ agent schedules by interacting with a deterministic event discrete simulation: Whenever a machine is idle, the agent is asked to schedule an operation to it. The environment maps the operation, selects the next free machine for a decision and updates the state representation accordingly. If no decision is possible at the current time, the state is rolled forward in time by marking the operation with least remaining processing time as finished, updating all the time variables and freeing the corresponding machine. Cutting batches are defined iteratively by requesting operations for the same machine until the agent produces a finish flag. Assembly operations are triggered by the agent outputting the index of the first job in a job partition. Bending operations are triggered by a job index. As such the action space is given by $nk + 1$.

To compute the SM-JSSP reward, the environment keeps track of wasted material and the tardiness for every job partition. After all operations were scheduled, $r_{abs} \coloneqq -c(W) + \sum_{i=1}^{n} v_i - \lambda \max\{0, T_i\}$, where $c(W)$ is the cost of the total used material (including waste), T_i and v_i are the tardiness and value for the job partition i respectively and λ is parameter punishing tardiness. r_{abs} reflects the sum of product values, discounted proportional to deadlines, minus the total material cost. The reward is scaled to [0, 1] using the maximum possible score r_{max} i.e. no tardiness and no waste, $r_{rel} \coloneqq 1 - \frac{r_{max} - r_{abs}}{r_{max}}$.

Both move probabilities and expected value from a state are provided by a single NN as in [28]. As opposed to [28] however, a simple feed-forward NN is used for the SM-JSSP instead of ConvNets. This is because we do not stack the input states, and there is no geometric correlation in the SM-JSSP states. Note that since there is a strong temporal correlation between states, passing a stack of states to a ConvNet could be beneficial.

2.3 Experiment Setup

To validate our approach, we create 80 different offline scheduling instances and use AZ, the EDD heuristic, as well as an MCTS implementation as per [29] to find a scheduling sequence. Then we compare the respective results in three ways: First, we plot the relative scores achieved to get a rough assessment of the scheduling behavior. Secondly, we count the number of times a particular scheduling approach provided the best result among its peers. Lastly, we average the 80 obtained scores for each individual scheduling scheme.

We chose the EDD baseline as it is the most intuitive solution for a scenario where tardiness is to be minimized. Since due dates can be the same for different parts, ties are being broken by higher product value, larger area and higher number of bending steps. Whenever a machine is free, the operations that can be assigned to it are prioritized as per EDD and the one with the highest priority is scheduled to it.

The SM-JSSP instances are created as follows. The set of jobs J contains 10 job partitions. The number of jobs per partition is sampled from $U\{1,7\}$ Areas, outlines and number of bending steps are sampled from the uniform distributions $U(1dm^2, 164.26dm^2)$, $U(1dm, 122.83dm)$, and $U\{0,30\}$ respectively. The resulting one dimensional histograms are used to create a three dimensional histogram from which part descriptors are sampled uniformly at random for each job. These part descriptors are the independent variables needed to calculate the processing times for the different operations. The sheet area is fixed to $589.19dm^2$. Processing times for cutting are calculated on batch definition using a nonlinear estimator with the summed part areas to sheet area and summed part outlines to sheet outline ratios as input. Bending times are a linear function

of the area and the number of bends per part. Assembly times are a linear combination of the average part area, the average number of bends and the total number of parts in a job partition.

In terms of the AZ implementation, we use a fully connected feed forward NN with 2 layers à 1024 and 512 nodes respectively, a dropout of 0.3 to avoid over-fitting, batch-normalization to speed up training, and Adam for learning rate optimization. During self-play, τ is set to 1 for the first 15 decisions and then dropped to 0. During evaluation $\tau = 0$. c_{puct} is 1.5 throughout. Prior to any decision, the selection, evaluation, expansion and backup steps are repeated 6 times. The tree is kept until the end-state is reached.

The three scheduling techniques considered were implemented in python. The environment implementation is consistent with the openaiGym (https://gym.openai.com) API for RL. The AZ agent was implemented using Keras with a TensorFlow backend for the embedded NN and was trained using 12 CPU cores for parallel execution of self-play episodes and 2 GPUs for NN training.

Rather than training AZ from scratch, we pretrain it to mirror the EDD heuristic first. To that end, we use the random SM-JSSP instance generation scheme described above to generate 10^6 (state, decision, reward) triples on which we then train the NN until a validation accuracy of 90 % is achieved for move probabilities. To check the learned behavior, we do a small evaluation round on 30 SM-JSSP instances at this point as well. Thereafter we run the AZ RL training pipeline for 254 iterations, which takes about 5 days. For each iteration, data from 80 self-play episodes is added to a replay buffer of the 800 most recent SM-JSSP games, on which the NN is then trained.

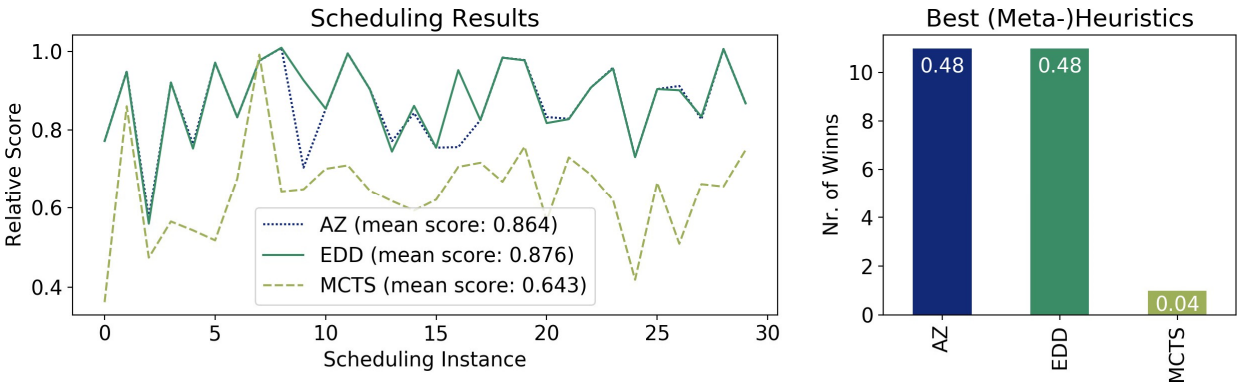
3. Evaluation

Scheduling a SM-JSSP instance to completion (circa 70 moves) takes about 150 seconds for both MCTS and AZ. Note that, while the self-play time is currently comparable, AZ should scale much better than MCTS with the size of the scheduling instance. For a game of depth n, MCTS runs n simulations to completion for every move selection step, while AZ simply makes n calls to the network's predict function. As n increases, so does the simulation time, while the call to the predict function stays constant.

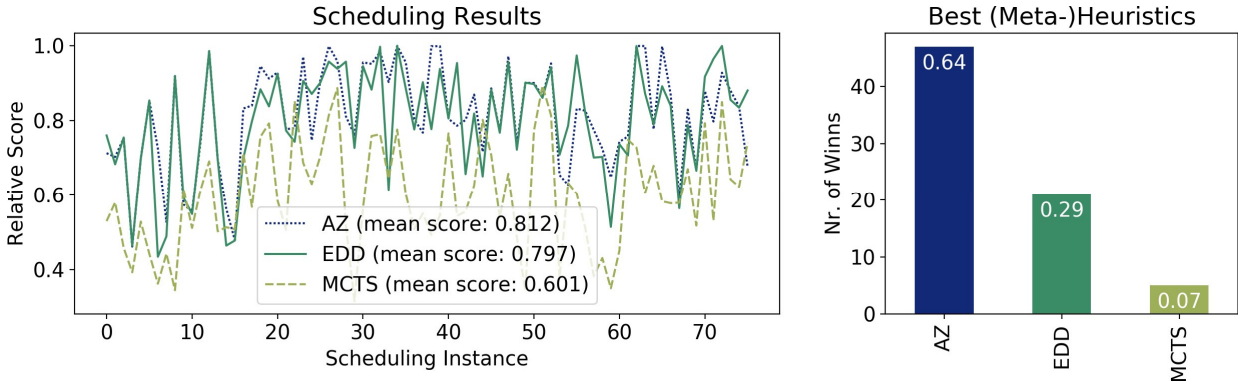
Figure 4 shows the performance of AZ, EDD and MCTS before and after AZ RL fine-tuning side by side. We elaborate on the aspects showcased by the figure. As a first observation we note that the performance of MCTS alone is quite lacking, compared to the heuristic approach or AZ. It registers both the lowest average score and the lowest number of wins in both evaluation rounds. This is to be expected given the small number of rollouts for a game tree as vast as the one corresponding to the SM-JSSP. Nevertheless, in 3% and 7% of post pretraining and post RL finetuning cases respectively, MCTS does find the best schedule.

Secondly, the scheduling behavior of AZ and EDD seems to be quite similar, as can be seen in the lineplots in Figure 4. This is not at all surprising given that EDD is a fairly good heuristic and AZ's network was pretrained on it. We notice that the overlap between the AZ and EDD curves is significantly higher immediately after pretraining than after RL fine-tuning. This suggests that AZ has developed strategies additional to EDD during self-play. The lineplots also reveal just how different the considered scheduling instances are, with best solutions achieving scores between 0.4 and 1 of the maximum producible value.

Thirdly, and most importantly, training AZ using RL leads to it outperforming its teacher, EDD. Immediately after pretraining however, the best scheduling approach, as ranked by both the number of wins and the average score, was EDD.



(a) AZ Performance Post Pretraining



(b) AZ Performance Post RL Fine Tuning (254 Iterations)

Figure 4: Comparison between AZ, MCTS and EDD using the relative score rrel defined in Section 2.2.

4. Conclusion

This work studied the applicability of AZ ran on modest hardware to the static SM-JSSP with a combined material waste and tardiness minimization target. We have shown, that AZ leads to better results than both EDD and MCTS, thereby providing the first successful application of AZ to the realm of production scheduling. On the way to our results, we formalized a novel production scheduling problem corresponding to sheet-metal production by extending the JSSP formalism and provided a RL design for it.

Our solution was tested with SM-JSSP instances of up to 70 jobs, 6 resources and 8 paths, leading to a combinatorial complexity higher than anything published in literature to date. For real world sheet-metal production however, the solution needs to scale to hundreds of jobs. To that end two aspects should be studied in the future. On the one hand, the investigation should be extended to larger SM-JSSP instances. On the other, it should be studied how AZ performs in an online setting. The two targets can be combined to provide the scalability required for real world applications.

Furthermore, it could pay off to offer some attention to the use of ConvNets within AZ for SM-JSSP to capture temporal correlations between states as well as relaxing the environment observability requirement. Currently, all the part descriptors involved in calculating the dynamic batch processing times are provided in the state model. If these times are to be computed accurately, the dimension of the agent's input space will explode, since both cutting and extraction times are dependent on a large number of factors such as machine configuration, material properties, further geometric features and so on. As such, it should be investigated whether AZ can still perform as strongly if we eliminate the part descriptors completely, save for areas.

Currently, the validation of AZ for the SM-JSSP is limited to the comparison with a heuristic baseline. It is planned to test AZ against GAs and exact solutions on smaller SM-JSSP instances. Additionally, AZ should be tested against the state of the art for JSSPs on benchmark data sets.

References

[1] F. Pfitzer, J. Provost, C. Mieth, W. Liertz, “Event-driven production rescheduling in job shop environments”, in 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), IEEE, pp. 939–944, 2018.

[2] M. Putz, A. Schlegel, “Simulationsbasierte Untersuchung von Priorit¨ats- und Kommissionierregeln zur Steuerung des Materialflusses in der Blechindustrie”, Simulation in Produktion und Logistik, pp. 370-379, 2019.

[3] L Li, C Li, L Li, Y Tang, Q Yang, “An integrated approach for remanufacturing job shop scheduling with routing alternatives.”, Mathematical biosciences and engineering: MBE, vol. 16, no. 4, pp. 2063–2085, 2019.

[4] M. Gondran, M.-J. Huguet, P. Lacomme, N. Tchernev, “Comparison between two ap- proaches to solve the job-shop scheduling problem with routing”, 2019.

[5] J. J. van Hoorn, “The current state of bounds on benchmark instances of the job-shop scheduling problem”, Journal of Scheduling, vol. 21, no. 1, pp. 127–128, 2018.

[6] S.-C. Lin, E. D. Goodman, W. F. Punch III, “A genetic algorithm approach to dynamic job shop scheduling problem”, in ICGA, 1997, pp. 481–488.

[7] T. Yamada, R. Nakano, “Scheduling by genetic local search with multi-step crossover”, in International Conference on Parallel Problem Solving from Nature, Springer, 1996, pp. 960– 969.

[8] B. M. Ombuki, M. Ventresca, “Local search genetic algorithms for the job shop scheduling problem”, Applied Intelligence, vol. 21, no. 1, pp. 99–109, 2004.

[9] E. S. Nicoara, F. G. Filip, N. Paraschiv, “Simulation-based optimization using genetic algorithms for multi-objective flexible jssp”, Studies in Informatics and Control, vol. 20, no. 4, pp. 333–344, 2011.

[10] L. Asadzadeh, “A local search genetic algorithm for the job shop scheduling problem with intelligent agents”, Computers & Industrial Engineering, vol. 85, pp. 376–383, 2015.

[11] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmu¨ller, T. Bauernhansl, A. Knapp, A. Kyek, “Optimization of global production scheduling with deep reinforcement learning”, Procedia CIRP, vol. 72, pp. 1264–1269, 2018.

[12] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, D. Hassabis, “Rein- forcement learning, fast and slow”, Trends in cognitive sciences, 2019.

[13] W. Zhang, T. G. Dietterich, “A reinforcement learning approach to job-shop scheduling”, in IJCAI, Citeseer, vol. 95, 1995, pp. 1114–1120.

[14] R. S. Sutton, A. G. Barto, et al., Introduction to reinforcement learning, 4. MIT press Cambridge, 1998, vol. 2.

[15] S. Mahadevan, G. Theocharous, “Optimizing production manufacturing using reinforce- ment learning”, in FLAIRS Conference, 1998, pp. 372–377.

[16] S. J. Bradtke, M. O. Duff, “Reinforcement learning methods for continuous-time markov decision problems”, in Advances in neural information processing systems, 1995, pp. 393–400.

[17] S. Riedmiller, M. Riedmiller, “A neural reinforcement learning approach to learn local dispatching policies in production scheduling”, in IJCAI, vol. 2, 1999, pp. 764–771.

[18] C. D. Paternina-Arboleda, T. K. Das, “A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem”, Simulation Modelling Practice and Theory, vol. 13, no. 5, pp. 389–406, 2005.

[19] T. Gabel, M. Riedmiller, “Scaling adaptive agent-based reactive job-shop scheduling to large-scale problems”, in 2007 IEEE Symposium on Computational Intelligence in Scheduling, IEEE, 2007, pp. 259–266.

[20] Y. C. F. Reyna, Y. M. Jim´enez, J. M. B. Cabrera, B. M. M. Hern´andez, “A reinforce- ment learning approach for scheduling problems”, Investigaci´on Operacional, vol. 36, no. 3, pp. 225–231, 2015.

[21] S. Qu, J. Wang, S. Govil, J. O. Leckie, “Optimized adaptive scheduling of a manufactur- ing process system with multi-skill workforce and multiple machine types: An ontology-based, multi-agent reinforcement learning approach”, Procedia CIRP, vol. 57, pp. 55–60, 2016.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Ried- miller, “Playing atari with deep reinforcement learning”, arXiv preprint arXiv:1312.5602, 2013.

[23] A. Kuhnle, L. Sch¨afer, N. Stricker, G. Lanza, “Design, implementation and evaluation of reinforcement learning for an adaptive order dispatching in job shop manufacturing systems”, Procedia CIRP, vol. 81, pp. 234–239, 2019.

[24] N. Stricker, A. Kuhnle, R. Sturm, S. Friess, “Reinforcement learning for adaptive order dispatching in the semiconductor industry”, CIRP Annals, vol. 67, no. 1, pp. 511–514, 2018.

[25] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, “Trust region policy optimization”, International conference on machine learning, 2015, pp. 1889–1897.

[26] M. Gombolay, R. Jensen, J. Stigile, S.-H. Son, J. Shah, “Apprenticeship scheduling: Learning to schedule from human experts”, AAAI Press/International Joint Conferences on Artificial Intelligence, 2016.

[27] H. Ingimundardottir, T. P. Runarsson, “Supervised learning linear priority dispatch rules for job-shop scheduling”, International conference on learning and intelligent optimization, 2011, pp. 263–277.

[28] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., “Mastering the game of go with deep neural networks and tree search”, nature, vol. 529, no. 7587, p. 484, 2016.

[29] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, “Monte-carlo tree search: A new framework for game ai”, in AIIDE, 2008.

[30] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al., “Mastering the game of go without human knowledge”, Nature, vol. 550, no. 7676, p. 354, 2017.

[31] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al., “Mastering chess and shogi by self-play with a general reinforcement learning algorithm”, arXiv preprint arXiv:1712.01815, 2017.

[32] R. D. Reid, N. R. Sanders, Operations Management. John Wiley & Sons, 2012, pp. 561– 564.

Biography

Alexandru Rinciog obtained his M.Sc. in Computer Science from Karlsruhe Institute of Technology in 2019. He became a Ph.D. candidate at the Chair of Enterprise Logistics of the TU Dortmund University the same year. As part of the DFG project “Smart Production Control”, his focus lies with production optimization using machine learning.

Carina Mieth is a PhD candidate in mechanical engineering at the TU Dortmund University. She received her M.Sc. in electrical engineering from the Karlsruhe Institute of Technology in 2017. Currently, she works in the predevelopment networked systems at TRUMPF GmbH + Co. KG. She is an associated member in the DFG-funded research training group adaption intelligence of factories in a dynamic and complex environment. Her research focuses on cyber-physical production systems and simulation input modeling for manufacturing simulation.

Paul Maria Scheikl received his B.Sc. and M.Sc. in mechanical engineering from Karlsruhe Institute of Technology, Germany in 2015 and 2017, respectively. He is now pursuing his Ph.D. in robotics to continue

his research with learning systems and collaborative robots. His research interests include dynamic systems, computer vision, and machine learning.

Anne Meyer is a junior professor for enterprise digitalization and supply-chain management at the Chair of Enterprise Logistics of the TU Dortmund University. She obtained her Ph.D. from Karlsruhe Institute of Technology in 2015. The 10-year colaboration with the FZI Research Center for Information Technology, served to grow her hands-on expertise in the fields of logistics, machine learning and data analytics.