A Dynamic Multi-objective Scheduling Approach for Gradient-Based Reinforcement Learning [⋆]

Katharina Hengel [*] **Achim Wagner** [*] **Martin Ruskowski** [*,**]

[*] *German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany (e-mail: {katharina.hengel, achim.wagner, martin.ruskowski}@dfki.de)*
[**] *Technologie-Initiative SmartFactory KL e.V., Kaiserslautern, Germany*

Abstract: In manufacturing dynamic scheduling is a complex task which is influenced by various factors, including several possibly contrary optimization objectives. While multi-objective optimization approaches are not novel anymore, prevailing deep learning solutions lack in the ability to dynamically adjust the preferences of the different objectives. For this reason, we developed a collaborate human-AI scheduling algorithm where Reinforcement Learning (RL) is utilized for the optimization of single objectives, while the human is left in control to flexibly mix and weight the influence of each objective into the final schedule. The approach performs on par with prevailing RL multi-objective approaches. However, it surpasses the later in its ability to dynamically adjust the preferences of the objectives without retraining.

Keywords: Scheduling Reinforcement Learning Multi-objective Optimization

1. INTRODUCTION

In manufacturing dynamic scheduling is a complex task which is influenced by various factors, including several possibly contrary optimization objectives. With increased attention on environmental sustainability objectives such as CO2 emission and energy consumption are getting more important. Nevertheless, manufacturing aiming at the adherence to delivery deadlines as well as for a high throughput is just as important for the economic efficiency of a company. While multi-objective optimization approaches are not novel anymore, prevailing deep learning solutions lack in the ability to dynamically adjust the preferences of the different objectives. In RL the optimization objectives are included in the reward function. The most common way to handle multi-objective optimization problems is to use the weighted sum of all objectives as reward function (Popper et al., 2021a) (Yuan et al., 2024). Preferences between the objectives can then be addressed by adjusting the weight for each objective. However, this common solution lacks in situation where the schedule must be optimized for multiple objectives while the preferences of the objectives dynamically change. Due to the necessary training time of RL, the model must be retrained whenever the reward function changes. Beyond this Hayes et al. (2022) state even more disadvantages of using a single scalar additive reward function for multi-objective optimization. The process of finding the correct weights for each objective is tedious. The relationship between the reward weights and the actual objective outcome might be non-linear. Hence, the AI-engineers have to make an educated guess at the scalarisation a priori. This leaves the decision of the objectives preferences with the AI-engineers and not the domain experts. Different reward functions are tested until the behavior of the policy is satisfactory which results in a semi-blind manual process. The necessity to iteratively re-engineer the reward function can eventually have a large hidden cost in terms of sample-complexity and computation time. Furthermore, the approach lacks a posteriori explainability of the results.

Hence, a collaborate human-AI dynamic scheduling algorithm is necessary utilizing RL for the optimization of single objectives, while the human is left in control to flexibly mix and weight the influence of each objective into the final schedule. We extended an auction-based online scheduling algorithm using reinforcement learning (ABOS-RL) to the multi-objective setting. In the algorithm the machines bid for the tasks. The machine with the highest bid wins and the task is assigned to this machine. RL is utilized to compute the bidding of a machine. For each single objective a separate policy is trained. The different policies are combined using a utility function of the bids for each objective.

The remainder of this paper is organized as follows: Section 2 gives an introduction into the scheduling problem and the state-of-the-art of multi-objective optimization with RL. The scheduling algorithm is introduces on which our proposed multi-objective scheduling approach

[⋆] This work has been funded by the European Commission through the HORIZON-CL4-2022-HUMAN-02 project HumAIne - Hybrid Human-AI Decision Support for Enhanced Human Empowerment in Dynamic Situations (No.101120218).

described in section 3 is build upon. Experimental results are presented in section 4 and discussed in section 5.

2. RL-BASED SCHEDULING

2.1 Flexible Job Shop Problem

In this paper we focus on the scheduling for a modular production. This scheduling problem is known as flexible job shop problem (FJSP) (Pinedo, 2012). In this problem n jobs $\mathcal{J} = \{j_1, j_2, ..., j_n\}$ must be processed on m machines $\mathcal{M} = \{m_1, m_2, ..., m_m\}$. Each job j_i is being composed of a set of tasks $\mathcal{T}_i = \{t_1^i, t_2^i, ..., t_k^i\}$ which must be processed in a given sequential order. Let d_i be the deadline for job j_i and $\mathcal{D} = \{d_1, d_2, ..., d_n\}$ be the set of deadlines. A task can be process by a specific subset $\mathcal{M}_{ij} \subset \mathcal{M}$ of all machines. A machine can only process one operation at a time. The processing time and energy consumption of task t_j^i on machine l are denoted by p_l^{ij} and e_l^{ij}.

2.2 Multi-objective Optimization in RL

Roijers et al. (2013) divide the multi-objective optimization approaches into two classes: single policy and multiple policy. In the single policy case only one policy is trained. The objectives are scalarized using weights which must be known a priori. In contrast multiple solutions can be inferred in the multi-policy case such that the Pareto front of the optimization problem can be explored. The later approach is preferred in cases were the weights are unknown and change over time.

Popper et al. (2021a) and Yuan et al. (2024) both use the single policy approach for scheduling of FJSPs. In an auction based multi-agent reinforcement learning (MARL) scheduling algorithm Popper et al. (2021a) used the linear weighted sum of the objectives as reward function. The proposed approach outperforms the shortest-job-next and first-come-first-serve heuristic in minimizing the contrary designed optimization goals of energy costs and the number of delayed jobs. Using also the weighted sum of the objectives inside the reward function, Yuan et al. (2024) optimizes for three different objectives: makespan, mean lateness and mean flow time.

Hayes et al. (2022) developed a guide for multi-objective optimization using the multi policy approach. It proposes to use a utility function which maps the state values of the objectives to a scalar value.

$$u : \mathbf{R}^d \to \mathbf{R} \tag{1}$$

$$V_u^\pi = u(V^\pi) \tag{2}$$

This approach is also adopted by Méndez-Hernández et al. (2019) and Huo and Wu (2023). While Méndez-Hernández et al. (2019) investigate the optimization of the job shop problem for makespan and tardiness, Huo and Wu (2023) do this for the FJSP and the objectives maximum completion time and total machine load. Both approaches use a MARL algorithm and use the borda rule to select among the given policies.

Our approach differs from the above in that it uses a multi-policy approach combining the different policies not using the value or action value function of each objective. Instead using a gradient-based RL algorithm the utility function of the multi-policy approach is directly applied on the RL action, which is the bid of an auction process. Furthermore, we stress that the weights of the objectives can be dynamically controlled by the human, resulting in a joint and collaborative scheduling process of AI and human.

2.3 ABOS-RL

We use an auction-based online scheduling algorithm using reinforcement learning (ABOS-RL) which is based on the RL-algorithm by Popper et al. (2021b). Let S be the set of currently open tasks, i.e. a task is in this set, if the precedence task has already finished and the task has not been assigned to a machine yet. For each task $t \in S$ an auction process is started. Each machine is requested to issue a bid for t. The highest bid for a given machine m and task t wins. If the bid is above a given threshold, the task is assigned to the machine and added into its waiting queue. Afterwards, set $S = S \setminus \{o\}$. As soon as t is finished being processed, the successive operation of t is included into S. This procedure is depicted in algorithms 1 and 2.

Algorithm 1: ABOS-RL auction

```
input : set of jobs $\mathcal{J}$, set of machines $\mathcal{M}$,
        threshold $B$
$\mathcal{S} := \{t_1^1, t_1^2, ..., t_1^n\}$
while $\mathcal{S} \neq \{\}$ do
    forall $t \in \mathcal{S}, m \in \mathcal{M}$ do
        $obs^{t,m} := m.get\_observations(t)$
        $bid^{t,m} := get\_bid(obs^{t,m})$
    $t, m := \arg\max_{t \in \mathcal{S}, m \in \mathcal{M}} bid^{t,m}$
    if $bid^{t,m} > B$ then
        $m.process(t)$
        $\mathcal{S} := \mathcal{S} \setminus \{t\}$
```

Algorithm 2: ABOS-RL machine

```
input : task queue $\mathcal{Q}$
while True do
    $t_j^i := \mathcal{Q}.pop()$
    $process(t_j^i)$
    $\mathcal{S} := \mathcal{S} \cup t_{j+1}^i$
```

The bid is computed by using MARL, where each machine is represented by one RL agent. The reward is only given at the end of each problem instance. Proximal Policy Optimization (PPO) (Schulman et al., 2017) is used for optimization.

3. MULTI-OBJECTIVE RL FOR ABOS-RL

In our improved multi-objective approach we use the bid instead of the reward to combine all objectives into one. Even through we use PPO which is a policy gradient method, the bid is similar to a state value in RL. Hence, we can use a utility function as in Hayes et al. (2022) only applied to the bids not the state values of each policy. The procedure of one RL step for scheduling with ABOS-RL in a multi-objective setting is depicted in algorithm 3. Algorithm 3 differs from algorithm 1 in that it extends the original algorithm by a second for-loop over all objectives. For each task t, machine m and objective o a bid is created. Afterwards the bid for task t and machine m over all objectives $\mathcal{O}$ is calculated using a utility function. We exemplary use the weighted sum of the bids for each objective.

Algorithm 3: multi-objective ABOS-RL auction

```
input : set of jobs $\mathcal{J}$, set of machines $\mathcal{M}$,
        threshold $B$, set of objectives $\mathcal{O}$,
        utility function $u : \mathbb{R}^{|\mathcal{O}|} \to \mathbb{R}$
$\mathcal{S} := \{t_1^1, t_1^2, ..., t_1^n\}$
while $\mathcal{S} \neq \{\}$ do
    forall $t \in \mathcal{T}, m \in \mathcal{M}$ do
        $obs^{t,m} := m.get\_observations(t)$
        forall $o \in \mathcal{O}$ do
            $bid_o^{t,m} := get\_bid(obs^{t,m})$
        $bid^{t,m} := u(bid_{o_1}^{t,m}, ..., bid_{o_{|\mathcal{O}|}}^{t,m})$
    $t, m := \arg\max_{t \in \mathcal{S}, m \in \mathcal{M}} bid^{t,m}$
    if $bid^{t,m} > B$ then
        $m.process(t)$
        $\mathcal{S} := \mathcal{S} \setminus \{t\}$
```

4. RESULTS

A set of 10000 FJSP was randomly generated with $m = |\mathcal{M}| = 4$, $n = |\mathcal{J}| = 10$, $k = |\mathcal{T}_i| = 5$ and and $p_l^{ij} \in [10 - 100] \forall i \in \mathcal{J}, j \in \mathcal{T}_i, l \in \mathcal{M}$. The energy e_l^{ij} was constructed contrary to the processing time p_l^{ij} using equation 3, i.e. tasks with a long processing time require less energy while tasks with a small processing time require more energy.

$$e_l^{ij} = \mathcal{N}(101 - p_l^{ij}, 0.5^2) \tag{3}$$

Hence, a task can be processed either fast with high energy consumption or slow with low energy consumption. The deadlines of the jobs are constructed in a way, that they almost always can be reached if the task is processed on the machine with shorter processing time, but higher energy consumption. On the other side the deadlines cannot be reached, if the task is processed on the machine with larger processing time, but lower energy consumption.

In the implementation of ABOS-RL the observations vector of machine l and task t_j^i includes the following information:

- processing time: p_l^{ij}
- remaining time till deadline: $d_i - \text{now}$
- duration how long m is approximately still occupied
- number of task which are queued already for the machine
- total number of jobs currently being manufactured in the plant
- energy consumption: e_l^{ij}

In the remainder of the paper ABOS-RL combined with our improved multi-objective approach using the linear weighted bid is denoted by RL-SB while ABOS-RL in combination with the linear weighted sum of the objectives as reward function is denoted by RL-SR. Depending on the objective the reward in the algorithm RL-SB is computed as in equation 4 and 5:

$$r^{energy} = 1 - \frac{E - e^l}{e^u - e^l + 1} \tag{4}$$

$$r^{lateness} = \frac{1}{L + 1} \tag{5}$$

E denotes the total energy consumption, e^u an upper bound, e^l a lower bound of E and L the total lateness of all jobs $\mathcal{J}$. The objectives are combined using the utility function in equation 6

$$\begin{aligned} u(bid_{energy}^{t,m}, bid_{lateness}^{t,m}) =& w_{energy} \cdot bid_{energy}^{t,m} \\ &+ w_{lateness} \cdot bid_{lateness}^{t,m} \end{aligned} \tag{6}$$

In the first experiment we use a weight of 0.1 for the energy objective and 0.9 for the lateness objective. In this way we can minimize the environmental impact as much as possible while still meeting delivery deadlines. For comparison reasons we use the same weights in the reward of RL-SR which is depicted in equation 7.

$$r = 0.1 \cdot r^{energy} + 0.9 \cdot r^{lateness} \tag{7}$$

All results of RL-SB and RL-SR are averaged over 10 different training runs.

Furthermore, we apply the dispatching rules service in random order (SIRO), minimum energy (ENERGY) and minimum slack (MS). Comparing our results to SIRO gives us an impression of how much we actually improve. We use ENERGY and MS to easily get a good estimated of the optimal value of one objective. ENERGY reaches the actual optimum while MS gives only an estimation. For a better estimate we calculated MS by $d_i - \text{now} + p_l^{ij}$. Instead of subtracting the processing time as usually, we add it. Since in this case a task, which can be processed on two different machines, is preferably processed on the machine with the smaller processing time instead of the higher processing time. Hence, the dispatching rule gives a good comparison in relation to the lateness objective. Figures 1-3 show the results of the different algorithms evaluated on 100 FJSP instances. The mean is depicted by a green triangle and the median by an orange line. In both objectives RL-SB performs on par with RL-SR. RL-SB achives a mean lateness of 10.5 and energy consumption of 2233.9 while RL-SR a mean lateness of 8 and energy consumption of 2256.3. The dispatching rules ENERGY and MS perform good for one objective while poor for the other objective. ENERGY achieves a mean energy consumption of 1818 and mean lateness of 61.1. MS achieves a mean energy consumption of 2685 and mean lateness of 11.6. SIRO is in between the above two with a mean energy consumption of 2248 and mean lateness of 12.8. Figure 3 shows the results of the reward. The reward is calculated for each algorithm using the reward of RL-SR. In this way we can compare the overall performance of the different algorithms. RL-SB performs the best with a mean reward of 0.917 followed by RL-SR with 0.925, SIRO with 0.854, MS with 0.805 and ENERGY with 0.717.

In the second experiment we use the two models trained beforehand for RL-SB and vary the weights of the objectives. Without more training being necessary, we compute approximate values for the Pareto front. We use 0.2 increments for $w_{energy}, w_{lateness} \in [0, 1]$ while $w_{energy} + w_{lateness} = 1$ holds. Figures 4 and 5 how the results averaged over 100 FJSP instances. Gradually increasing $w_{energy} \in [0, 1]$ leads to a total energy consumption de-

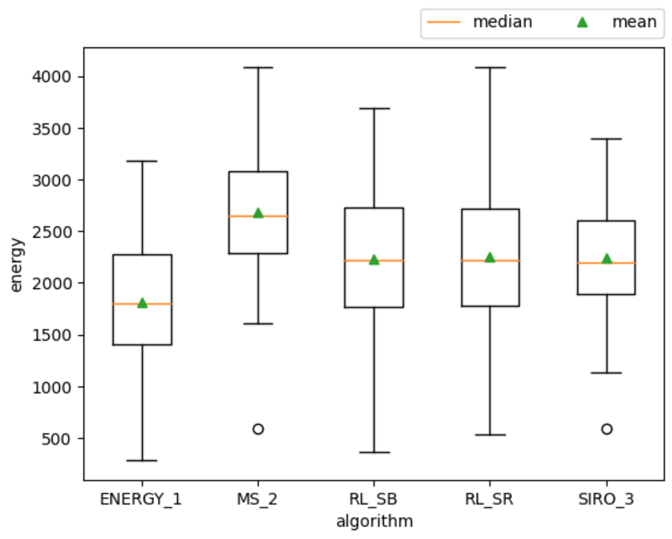

Fig. 1. Total energy consumption per algorithm

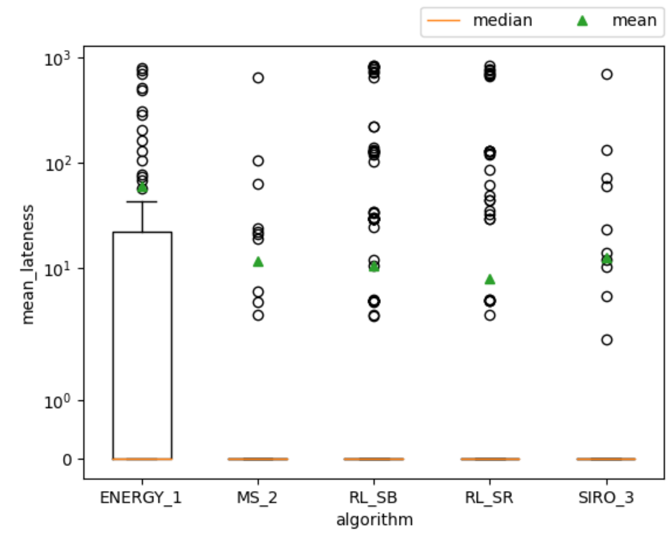

Fig. 2. Mean lateness of a job per algorithm

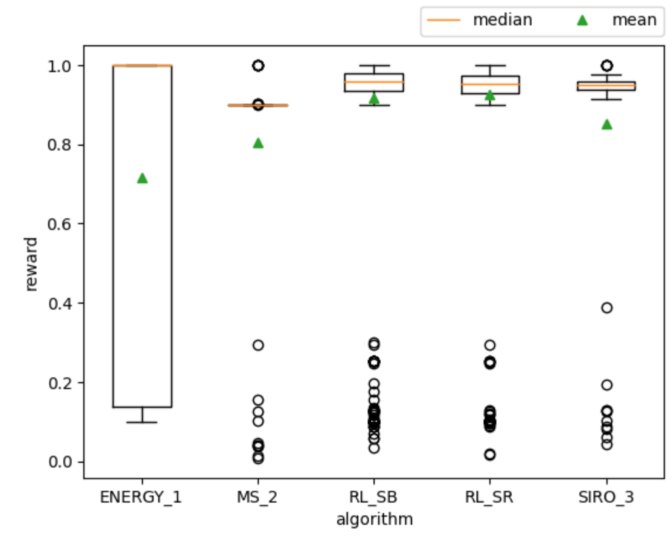

Fig. 3. Reward achieved by each algorithm

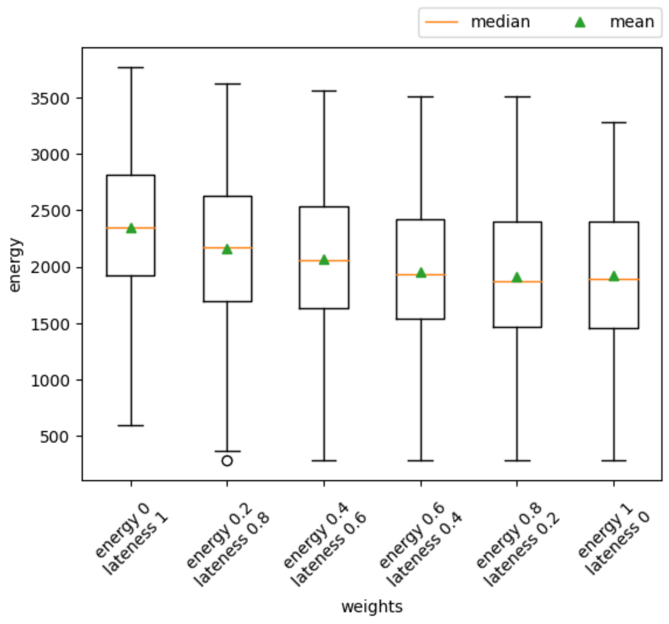

Fig. 4. Total energy consumption of RL-SB using different weights

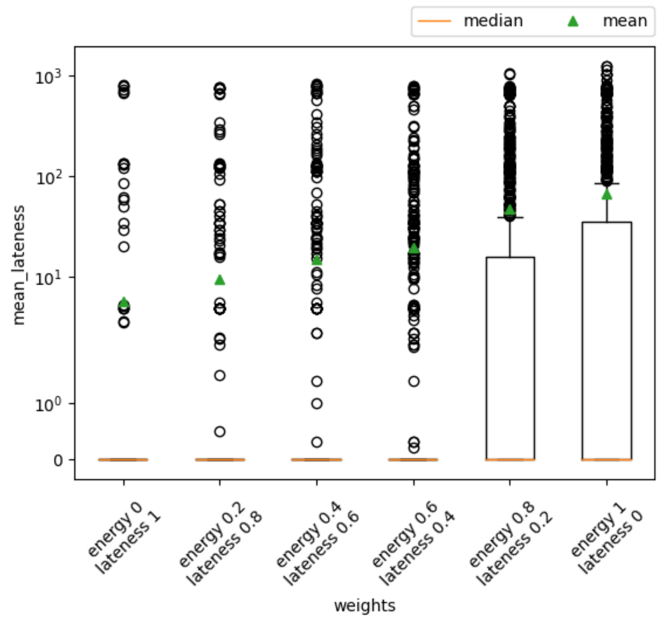

Fig. 5. Mean lateness of RL-SB using different weights

creases from 2350 to 1918.9. In contrast the mean lateness increases from 5.8 to 66.4 when $w_{lateness} \in [0, 1]$ is gradually decreased.

5. DISCUSSION

The RL-SB and the RL-SR approach achieve equal performance with only negligible difference. However, the superiority of RL-SB over RL-SR is in its flexible and dynamic adaptation to changing objectives. In RL-SB the objectives are mixed outside of the actual RL algorithm while in the RL-SR approach the objectives are mixed inside of the RL algorithm using the reward function. The former approach might have higher costs in the beginning, since a separate policy must be trained for each individual objective. The required sample complexity, training time and disk space scale linearly with the number of objectives. However, training can be done prior to deployment. Once the set of policies is trained the whole Pareto front of optimal solutions between the objectives can be explored. No additional training time and disk space is required after the deployment. In contrast, RL-SR only explores one solution of the Pareto front with one training run. Disk space consumption is at all times lower compared to RL-SB, since only one model must be stored at all times. Training time and sample complexity scale linearly with the number of times that the preferences are adjusted. If there are more objective changes than objectives in total RL-SB outperforms RL-SR in this point. Contrary to RL-SB the training phase cannot be prepend to initial deployment. Instead it is scattered over the usage. With each objective change a new training phase and fresh deployment is necessary. During this time the production either must be stopped or runs on with the old objectives and reduced performance. In summary, if the objective preferences are not known in advance or change often, RL-SB achieves a lower sample complexity and training time in the long run. The training phase can be completed beforehand such that the preference changes can be executed immediately and without interrupting the production. This comes at the cost of a higher disk space usage.

The decision making process is split between the human operator and the artificial intelligence (AI). By selecting and adjusting the weights for each objective the production manager is in control over the influence of each objective on the final schedule. During production he can dynamically adjust them to his preferences. This can be done by domain experts and not the AI algorithm engineers. The AI on the other hand is left in control over the computation of the actual schedule with the given objectives. In this way the AI aids the human. Both human and AI have their own fields of responsibility. The output of the human tasks, i.e. the selection of the preferences, is the input to the AI task, i.e. the computation of a schedule. This gives a defined interface between the human and the AI for a joint decision making process within the presented dynamic multi-objective scheduling algorithm.

The dynamic multi-objective scheduling approach has the unique subsequent phases training, deployment and production. In the training phase one model for each objective is trained. In the deployment phase these models are deployed to the machines. In the production phase the production manager is monitoring the production. As soon as one metrics shift to far from its desirable value, he can intervene in the production by setting new weights for the objectives.

6. FUTURE WORK

In the future, the dynamic multi-objective scheduling algorithm will be deployed to one testbed of the *SmartFactory*$^{\mathrm{KL}}$ (see Figure 6). This allows investigation of its applicability to modular production systems which focus on flexibility, resilience and robustness. By facilitating all resources of the factory like machines and transportation units with software agents, the factory can control itself to a high degree autonomously. The dynamic multi-objective scheduling algorithm hereby allows a threefold flexibility. On the one hand there is the flexibility in the factory setup. Due to the auction component in the scheduling algorithm the factory setup can change. Machines which are currently out of order due to maintenance or failures, do not take part in the auction while new machines will participate after their corresponding software agent is spawned. Secondly, there is flexibility in the products. The production is not planned in advance. Instead every task is scheduled after its predecessor has successfully finished. Hence, quality issues or defects of the product can be taken care of as soon as they are identified. New product types can be easily incorporated into the factory by defining the sequence of tasks necessary to produce them. And at last there is the flexibility in the objectives which can be changed and adjusted dynamically during production. New objectives can be added after training a new model and deploying it to the machines.

Fig. 6. Modular testbed of the *SmartFactory*$^{\mathrm{KL}}$

This flexibility fosters not only robustness and resilience, but also sustainability concerns can be addressed. In the example above we already minimized the energy consumption. Further research can be conducted to examine the algorithm performance together with a demand side management system, where the energy consumption combined with dynamic energy prices can be monitored.

7. CONCLUSION

Our improved multi-objective scheduling algorithm utilizes the gradient-based RL-algorithm PPO and an auction procedure. The objectives are combined using a utility function on the bids of the auction. We shown that our approach performs as good as the common multi-objective RL approach where the objectives are combined using the reward function. Nevertheless, it outperforms it in its applicability and flexibility. The control is split between the human and the RL algorithm. RL is utilized for single objective optimization and controls the next task-machine assignment in the scheduling process. The human on the other hand retains the control over the influence of the objectives into the schedule and is able to dynamically mix them.

REFERENCES

Hayes, C.F., Rădulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L.M., Dazeley, R., Heintz, F., et al. (2022). A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1), 26.

Huo, S. and Wu, W. (2023). Multi-objective fjsp based on multi-agent reinforcement learning algorithm. doi: 10.1109/iccnea60107.2023.00079.

Méndez-Hernández, B.M., Bazan, E.D.R., Jiménez, Y.M., Libin, P., and Nowé, A. (2019). A multi-objective reinforcement learning algorithm for JSSP. In I.V. Tetko, V. Kurková, P. Karpov, and F.J. Theis (eds.), *Artificial Neural Networks and Machine Learning - ICANN 2019: Theoretical Neural Computation - 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17-19, 2019, Proceedings, Part I*, volume 11727 of *Lecture Notes in Computer Science*, 567–584. Springer. doi:10.1007/978-3-030-30487-4_44.

Pinedo, M.L. (2012). *Scheduling*, volume 29. Springer.

Popper, J., Motsch, W., David, A., Petzsche, T., and Ruskowski, M. (2021a). Utilizing multi-agent deep reinforcement learning for flexible job shop scheduling under sustainable viewpoints. In *2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, 1–6. doi: 10.1109/ICECCME52200.2021.9590925.

Popper, J., Ruskowski, M., and Rheinheimer, I. (2021b). Using multi-agent deep reinforcement learning for flexible job shop scheduling problems. In R. Teti and D.M. D'Addona (eds.), *CIRP Proceedings. CIRP Conference on Intelligent Computation in Manufacturing Engineering (CIRP ICME), Italy*, volume 15th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 14-16 July 2019, Gulf of Naples, Italy. Elsevier B.V.

Roijers, D.M., Vamplew, P., Whiteson, S., and Dazeley, R. (2013). A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48, 67–113.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347. URL `http://arxiv.org/abs/1707.06347`.

Yuan, E., Wang, L., Song, S., Cheng, S., and Fan, W. (2024). Dynamic scheduling for multi-objective flexible job shop via deep reinforcement learning. doi: 10.2139/ssrn.4696880.