Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 55-2 (2022) 156–162

Reinforcement Learning-based Scheduling of a Job-Shop Process with Distributedly Controlled Robotic Manipulators for Transport Operations

Simon Jungbluth [*] Nigora Gafur [**] Jens Popper [***] Vassilios Yfantis [**] Martin Ruskowski [***]

[*] *Technologie-Initative SmartFactoryKL e.V., 67663 Kaiserslautern, Germany, (e-mail: simon.jungbluth@smartfactory.de)*
[**] *Chair of Machine Tools and Control Systems, Technische Universität Kaiserslautern, 67663 Kaiserslautern, Germany*
[***] *Innovative Factory Systems, German Research Center for Artificial Intelligence, 67663 Kaiserslautern, Germany*

Abstract: Job-shop scheduling problems are important in the industrial context to achieve high machine utilization. Heuristics offer a possibility to solve these problems with moderate computational effort. However, they might be associated with a high development effort and generalization to other tasks is difficult. We use a reinforcement learning approach (deep Q-learning) to solve a job-shop problem in our production environment. A production process is considered where jobs are transported to allocated stations by two collaborative robots. To this end, a learning environment and a simulation environment are developed to evaluate the feasibility of an obtained schedule. The results are compared to a First In - First Out heuristic. The main objective is to consider the motion of the robots and to avoid collisions without losing unnecessary time. First, a fixed scheduling problem is analyzed to verify that a feasible solution can be obtained. Second, arbitrary instances of the scheduling problem are solved. The presented method leads to feasible schedules. An increased training and a more stable convergence process are necessary for an efficient use.

Keywords: Scheduling, Machine learning, Intelligent manufacturing

1. INTRODUCTION

Scheduling problems have received widespread attention in industry and are found in areas such as transportation, distribution and manufacturing. An industrial setting requires the optimization of large scale job-shop scheduling problems (Cunha et al. (2020)). A job-shop scheduling problem (JSSP) describes an allocation of jobs to a limited number of resources over a certain period of time. The critical decision is to plan how to assign tasks to the machines so that an optimal target criterion is achieved. Depending on the type of optimization criterion, a scheduling problem can lead to the most difficult optimization problems. As the JSSP belongs to the class of NP-hard problems, an optimal solution can not be obtained with a clear rule of action (Brucker (2007); Martínez et al. (2011); Cunha et al. (2020)).

Conventional approaches to solving JSSPs are heuristics or priority rules. These are designed to solve a specific problem, with defined mathematical rules. The advantages include them being computationally fast, intuitive and easy to implement. The generalization to other problems and the effort to develop these techniques belong to the current challenges (Cunha et al. (2020); Zhang et al. (2020)). An alternative approach for solving the presented problem is machine learning or, more precisely, reinforcement learning (RL). Deep reinforcement learning (DRL) provides successful results for computer and board games (Mnih et al. (2015); Silver et al. (2017)). However, deep learning has been applied only on few applications in the processing industry (Waschneck et al. (2018); Cunha et al. (2020); Zhang et al. (2020)). Panzer and Bender (2021) provided a detailed literature review as an introduction into DRL supported production systems. The authors highlighted that in most examinations, conventional approaches are exceeded but the findings have to be transferred to real-word systems to evaluate the reliability under existing requirements.

In flexible manufacturing systems (FMS's) deadlocks may occur either during allocation processes (assigning resources to jobs) or between transportation units. For instance, to prevent deadlocks during allocation, Banaszak and Krogh (1990) proposed to utilize Petri net models to prevent deadlocks by defining a restriction policy that restricts resource allocation decisions. The second class of deadlocks, i.e., deadlocks during transportation task execution, usually stems from insufficient motion planing. In this case, intersecting trajectories prevent the task execution assuming sufficient measures for collision avoidance are in place. Zhou et al. (2017) introduced a distributed algorithm, where robots exchange their next two consecutive states and check if a deadlock occurs. Depending on the manufacturing system, both types of deadlocks need to be considered.

In this paper, we use DRL to solve JSSPs for an FMS and transfer the solutions into a simulation environment. We sched-

[⋆] The authors gratefully acknowledge financial support by the Federal Ministry for Economic Affairs and Energy of Germany in the project "Modular Smart Manufacturing Gaia-X Testbed" (SmartMA-X, 13I40V005A).

10.1016/j.ifacol.2022.04.186

ule the processing as well as the transporting to check the feasibility of our generated plans. We set up a learning environment for our production plant and integrate deep Q-learning. Two collaborative robots (UR3) are considered to transport the assigned resources to different stations. The robots share a common workspace and collisions between the robots are imminent. An online motion control is employed to provide optimal and collision-free trajectories for each robotic manipulator. We want to emphasize that especially transport operations of robotic manipulators are associated with many restrictions and scheduling the process operations alone does not lead to a sufficient result. Thus, theoretical scheduling meets our system requirements. We determine whether deep Q-learning is a suitable method to solve our JSSP and evaluate the schedules against a FIFO heuristic.

2. RELATED WORK

The use of RL methods in the area of optimization and scheduling has already been investigated in 1995. Zhang and Dietterich (1995) presented a successful application of RL combined with neural networks to solve JSSPs. Based on a critical-path schedule, RL was used to find a short conflict-free schedule. Aydin and Öztemel (2000) focused on dynamic scheduling with an RL agent to select the most suitable priority rule in real-time. The dynamic behavior was achieved with a simulation environment communicating with the agent when a job needed to be assigned to a machine. The results outperformed the traditional alternatives. Subsequently, Gabel and Riedmiller (2007) developed techniques to solve JSSPs and showed that solutions with multiple agents are advantageous to get nearly optimal results. The agents were associated with resources and decided which job has to be processed next on the resource. Martínez et al. (2011) combined learning and optimization to solve flexible JSSPs. Two-stage Q-learning was applied to find the routing and sequencing with priority rules-related rewards. Then the mode optimization procedure was used to minimize the makespan. In 2018, Waschneck et al. (2018) applied cooperative deep Q-network agents to production scheduling. The authors used discrete event simulation to observe the factory state and distributed agents at different workcenters optimizing a global reward. The obtained results were on par with the expert benchmark. Wang et al. (2019) presented a multi-agent RL setting with a deep Q-network to control the scheduling of multi-workflows while outperforming the baseline algorithms. Chen et al. (2020) solved flexible JSSPs with a self-learning genetic algorithm. The approach combined genetic algorithm as the substantially optimization with RL to customize the key parameters resulting in outperforming its competitors. Zhang et al. (2020) mentioned the importance of priority rules. Since developing these methods is time-consuming, the authors focused on the question of whether they can automate the process of designing priority rules using DRL. Priority rules trained by the authors' policy perform significantly better than existing manually designed ones. Popper et al. (2021) developed a RL multi-agent system combining job scheduling and vehicle planning. The authors explained that the location of the means of transport is usually omitted in scheduling but new research should consider these aspects. The presented concept performed up to 100 times better than the compared heuristics.

Especially, Waschneck et al. (2018), Zhang et al. (2020), Panzer and Bender (2021) and Popper et al. (2021) highlighted that the numerous real-world applications of JSSPs are widely unexplored with respect to RL. Popper et al. (2021) used the inclusion location of the transport units to increase the transferability to real production systems. Whereby the authors investigated automated guided vehicles, robotic manipulators are linked with significantly more restrictions and we concentrate on the feasibility of our production plant.

3. METHODS

We use a deep Q-learning method based on a global state to train in the learning environment and to solve the JSSP. We consider a transportation system consisting of two robotic manipulators sharing a common workspace. The robots perform pick and place operations, where trajectories are performed using distributed model predictive control (DMPC). Hereby, collision and deadlock avoidance between the robots are considered.

3.1 *Job-Shop Scheduling Problem*

The JSSP is classified by a set of n jobs $J = \{J_1, ..., J_n\}$ and m machines $M = \{M_1, ..., M_m\}$. Each job $J_i (i = 1, ..., n)$ consists of n_i operations $O_i = \{O_{i1}, ..., O_{i,n_i}\}$. The processing time p_{io} defines the processing time of the operation O_{io} $(o = 1, ..., n_i)$. Each operation O_{io} $(i = 1, ..., n, o = 1, ..., n_i)$ is linked to a set of $M_{io} \subset M$ machines on which it may be executed. Each job has to be processed in an individual sequence of machines, in which a single machine might be used for more than one operation (Brucker (2007); Martínez et al. (2011)). Heuristics can be used to solve the JSSP, for example, a FIFO heuristic where the prioritization is based on arrival (Haupt (1989)).

3.2 *Reinforcement Learning*

RL describes an approach, where an agent learns its own behavior by interacting with a dynamic environment. The agent makes certain decisions that result in reward or punishment. With the environment in state S_t, the agent can perform a set of predefined actions A_t from the set of all actions A. Depending on the selected action A_t and its subsequent state S_{t+1} of the environment, the agent receives a reward R_{t+1}. The main objective is to find a policy to maximize the sum of the rewards. The agent has two general learning strategies: a decision must be taken to explore previously poorly examined state space regions or to check the correctness of the previous states. One way to handle this problem is the ε-greedy strategy with a parameter $\varepsilon \in [0, 1]$. Before an action A_t is performed, a random number x is generated that decides whether the agent follows its strategy $(x \geq \varepsilon)$ or performs a random action $(x < \varepsilon)$ (Sutton and Barto (2018); Lapan (2018)). Q-learning describes a possible algorithm of model-free learning to find an optimal policy, (Watkins and Dayan (1992)). The expected benefit of an action A_t in the state S_t is described by the Q-function $Q(S_t, A_t)$ in the following form,

$$\begin{aligned} Q_{\text{new}}(S_t, A_t) = &Q_{\text{old}}(S_t, A_t) + \eta [R_{t+1} + \gamma \max_A Q_{\text{old}}(S_{t+1}, A) \\ &- Q_{\text{old}}(S_t, A_t)]. \end{aligned} \tag{1}$$

The corresponding values are stored in the Q-table. The new Q-value denoted as $Q_{\text{new}}(S_t, A_t)$ consists of the old value $Q_{\text{old}}(S_t, A_t)$, the reward R_{t+1} and the maximum Q-value of all the next states $\max_A Q_{\text{old}}(S_{t+1}, A)$. The parameter η denotes the learning rate and defines the step size at each iteration. The discount factor γ controls the influence of future decisions. Q-learning is applicable if the number of possible states S_t

and actions A_t is manageable (Sutton and Barto (2018); Lapan (2018); Dutta (2018)). For this reason, neural networks are used in deep Q-learning to approximate the values of the Q-function (Mnih et al. (2015); Fan et al. (2019)). The learning process of deep Q-networks is facilitated by two essential steps:

(1) A virtual memory (replay memory) is created. The memory e consists of all transitions,
$$e = (S_t, A_t, R_{t+1}, S_{t+1}).$$
Random samples of the memory can be used for training (Lin (1992); Mnih et al. (2015); Fan et al. (2019)).

(2) A target network is used to approximate the Q-values of the next state $\max_A Q_{\text{old}}(S_{t+1}, A)$. The target network serves to determine an uninfluenced estimator for the error function. The step size τ is defined, whereby the weights of the target network are updated with those of the policy network (Dutta (2018); Fan et al. (2019)).

Depending on the strategies explained so far, the algorithm of deep Q-learning is presented in Mnih et al. (2015). We adopt this procedure to our problem statement in Algorithm 1. In Sec. 4.2, a detailed description of Algorithm 1 is provided during the introduction of the training procedure.

3.3 Online Motion Control

Online motion control is realized by applying the approach of Gafur et al. (2021a,b) which is based on the concept of distributed model predictive control (DMPC). Each robotic manipulator plans its own trajectory after receiving a sequence of setpoints from the scheduling algorithm. As robotic manipulators are operating in a shared workspace, they have to account for potential collisions between each other. Therefore, each robotic manipulator shares its predicted trajectory with all the other manipulators to compute a collision free trajectory. Furthermore, deadlocks between the robots are locally detected by each robot and communicated to a coordinator that resolves the occurring deadlocks. More details on the applied algorithm are provided in Gafur et al. (2021a).

Algorithm 1: Training for solving JSSPs

```
Initialize the replay memory $e$ with capacity $N$;
Initialize the policy network $Q$ with random weights $w_{ij}$;
Initialize the target network $Q^*$ with random weights and $w^*_{ij} = w_{ij}$;
Loop
    Initialize $\varepsilon$ with $\varepsilon = \varepsilon_{\text{start}}$;
    for Sequence = 1,seq do
        Initialize the JSSPs for $y$ learning environments;
        for Step = 1,Steps do
            Reset state $S_1$ for every learning environment;
            for env = 1,y do
                while JSSP not solved or not terminated do
                    With $\varepsilon$ choose random action $A_t$ considering
                      normal action space and reduced action space or
                      $A_t = \text{argmax}_A Q(S_t, A; w_{ij})$;
                    Run action $A_t$, get reward $R_{t+1}$ and state $S_{t+1}$;
                    Store state transition $(S_t, A_t, R_{t+1}, S_{t+1})$ in $e$;
                end
                Get random stack $(S_j, A_j, R_{j+1}, S_{j+1})$ from $e$, calculate
                  the error with Q-function and train the neural
                  network with gradient descent procedure for $x$ times;
                Every $\tau$ steps reset $Q^* = Q$;
            end
            if $\varepsilon > \varepsilon_{end}$ then
                $\varepsilon = \varepsilon - \varepsilon_{\text{decay}}$;
            end
        end
    end
    Initialize the JSSPs for $z$ validation environments;
    for env=1,z do
        while JSSP not solved or not terminated do
            With policy ($\varepsilon = 0$) choose action
              $A_t = \text{argmax}_A Q(S_t, A; w_{ij})$;
        end
    end
EndLoop
```

4. SYSTEM CONFIGURATION

4.1 Learning Environment

The production plant is shown in Fig. 1. It consists of two UR3 robots, five machine stations (Fig. 1: black objects) and 5 jobs (Fig. 1: red objects), each with 4-5 operations. In addition, there are two buffer stations in the middle of the workspace (Fig. 1: circled black objects) that serve as an intermediate storage. A starting and ending tray (Fig. 1: blue trays) are used as interface to the working area. Within the context of discrete event simulation time is not modeled explicitly. Instead, discrete event points are defined and their timing is computed within the simulation by accounting for the duration of processing and transport times between consecutive events. Set up times s and transport times t between the stations are taken into account. For simplification, the set up time is considered as a constant with 20 time steps. Every time a robot picks a job the set up process is triggered. The set up process is carried out as soon as possible and retrospectively unused processing time could be taken to speed up the operation. Also, the transport times are specified to be constant with an initial value of 12 time steps. The transport is divided into two processes, the pick operation (transport 1) and the place operation (transport 2). An exception occurs when the robots avoid collisions between each other in the way, when one robot has to choose a trajectory above the other one in order to reach its target. Through collision avoidance, the transport time is increased to 20 time steps. The processing time is chosen between 20 and 50 time steps depending on the JSSP. The goal is to bring the different jobs from the respective starting tray through a predefined station sequence to the end tray. The target criterion is to minimize the makespan $C_{\max}$, i.e., the time required to complete all jobs. We use a neural network to assign the robots to transport operations and to sequence the operations in the production units.

Since deadlocks between the robots can occur during grasping procedure, our scheduling algorithm has to take restrictions

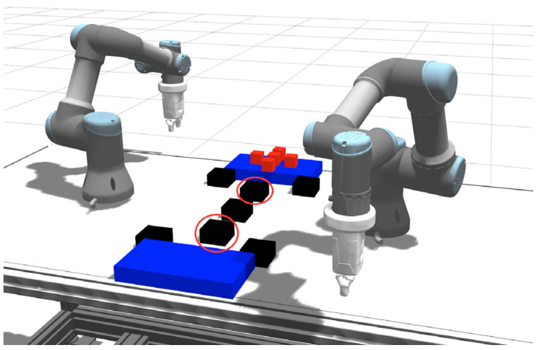

Fig. 1. Simulation environment of the production plant in Gazebo.

into account, i.e., a minimum distance of 0.12 m between two objects that have to be grasped at the same time. In our case, this leads to the fact that not every station can be assigned, depending on the current location of the robots. It is only possible that one robot grasps objects from the starting or ending tray. Also, it is not possible that the outer stations, which are next to the trays, can be reached on both sides of the table at the same time, i.e., that one robot moves from the bottom station to the upper outer station while the other robot does it reversed. Furthermore, resource allocation can lead to a deadlock since the progress of the process is always dependent on a subsequent machine or buffer station being free. In a deadlock situation, each job is waiting for a resource occupied by another job. This leads to a significant increase in the complexity of the JSSP. At this point, it should be noted that these constraints are not considered in the FIFO heuristic. Thus, the RL approach is compared to a heuristic one with far fewer boundary conditions.

In the learning environment, the robots can perform three different actions. A job can be placed at a machine station, at a buffer station or a robot can wait 5 time steps at the current location. Thus, for n jobs $2n+1$ actions follow. After completion of the action, another action is specified until the JSSP is solved or a defined simulation time or number of actions threshold has been reached. These boundaries are determined empirically. The actions are based on a global state that is normalized between 0 and 1 before passing through the neural network. This state contains an entry for each next operation, each machine, each buffer, each robot, the time for each robot and the jobs. The next operation includes the current location of the job as well as the next machine and the associated processing time. The machines cover the remaining processing time, set up process notifications and information if the machine is idle, processing or blocked. The robots track their movement as well as the required time for the transport operations. Also, the overall simulation time and the number of performed actions are tracked. The jobs initially describe the general JSSP, whereby the corresponding entry is set to 0 after a completed operation.

The presented approach and the associated actions result in different rewards and punishments. A distinction is made between local rewards after each action and global rewards when the JSSP has been solved. The statements and values are listed in Tab. 1. Local negative rewards result in actions that can not be executed. This includes a job that is already allocated to the other robot or a possible deadlock situation between the transportation units due to the current state. Another negative reward is received when a job is chosen which next station is already occupied. Thus, in a resource allocation deadlock, the agent always receives a punishment until the termination criterion is reached and it therefore has to learn how to avoid these situations. Local positive rewards follow up from actions that lead to a successful pick and place process. The main objective is to minimize the makespan, resulting in a time dependent global reward. Since the optimal makespan is not known, the robots strive for a makespan of 0. The reward is scaled between 0 time steps and the time steps of the termination criterion. A high makespan results in a global reward of 1, while a schedule with a theoretical makespan of 0 results in a reward of 11.

Table 1. Explanation of the rewards

Statement	Value
Action can not be executed (e.g., movement of robot is not possible, job is already assigned to a transportation task)	-2.5
Action can not be executed currently (e.g., buffer stations are occupied, target machine station is occupied)	-1.25
Wait a time step	-0.05
Job was placed at buffer station	0.04
Job was placed at machine/end station	0.08
All jobs were processed	$R \in \{r \vert 1 \leq r \leq 11\}$

4.2 Training Procedure

Two different case studies are conducted to evaluate the presented approach. In the first case, we train the network with a fixed JSSP, where the objective is to find a feasible schedule and to evaluate the chosen parameters. In the second case, we use arbitrarily generated JSSPs to train the network and to solve random JSSPs.

Most hyperparameters of the learning process are fixed. Only the learning rate, the batch size and the size of the neural network are adjusted. These values are further explained in the results section (Sec. 5). The values of the other parameters are as follows: discount factor $\gamma = 0.99$, $\varepsilon_{\text{end}} = 0.1$, step size $\tau = 50$. Adam optimizer (Kingma and Ba (2014)) is applied to adjust the neural network. As the error function the least squared error method is used. The rectified linear unit activation function is applied in the hidden layers of the network and the linear activation function at the output. To speed up the process, a distinction is made during exploration between a normal action space and a reduced action space. The reduced action space excludes actions that lead to obviously negative rewards based on the current state. This is useful since in some cases only few actions lead to a positive change of the system. The probability of selecting the appropriate action space is 50 % in each case. Considering the methods presented in Sec. 3.2, we employ a deep Q-learning algorithm presented in Algorithm 1. The *seq*-parameter (sequence) defines how many times the number of learning environments y is reset and JSSPs are generated. At the end, there is a validation phase with z environments to evaluate the current policy. The algorithm is explained for the two use cases in the following paragraphs.

Fixed JSSP To solve a fixed JSSP one learning environment ($y = 1$), one validation environment ($z = 1$) and one sequence is considered ($seq = 1$). The JSSP is saved for both environments and restored in each sequence. The robots solve the JSSP with the ε-greedy strategy. After the JSSP is solved, the training takes place 5 times ($x = 5$). Within one episode, the JSSP is solved several times. In the first episode, the JSSP is solved 2500 times with an unchanged $\varepsilon_{\text{start}} = 1$. As long as the policy does not lead to a solution in the validation phase, the $\varepsilon_{\text{start}}$ value is set to 1, the JSSP is solved 1000 times and ε decreases by $\varepsilon_{\text{decay}}$ = 1e-3. If the policy solves the JSSP, $\varepsilon_{\text{start}} = 0.5$ and ε decreases with $\varepsilon_{\text{decay}}$ = 5e-4.

Arbitrary JSSPs To solve arbitrary kinds of JSSPs 10 learning environments are created ($y = 10$). Within an episode, different kinds of JSSPs are solved. After each JSSP is solved, the training takes place 5 times ($x = 5$). Each JSSP is solved 100 times then new ones are created. The validation phase consists of 200 unseen JSSPs ($z = 200$). In the first episode, 16 sequences ($seq = 16$) and therefore 160 JSSPs are solved with an unchanged $\varepsilon_{\text{start}} = 1$. As long as 75 % of the JSSPs are not solved in the validation phase, $\varepsilon_{\text{start}} = 0.9$ holds and decreases with $\varepsilon_{\text{decay}} = 5.625\text{e-}4$. If 75 % of JSSPs are solved, the number of sequences is reduced to 8 ($seq = 8$). In this case, $\varepsilon_{\text{start}}$ is set to 0.5 and decreases with $\varepsilon_{\text{decay}} = 6.25\text{e-}4$.

The solutions of the validation phase are used to present the results of the current policy in every episode in Sec. 5.

5. RESULTS

The learning environment is developed within the Python programming language. Discrete event simulation is applied to model the operations. To this end, the Python library SimPy (2021) is used. The neural network is trained with TensorFlow 2 (Abadi et al. (2015)). The simulation environment is set up in Gazebo (Koenig and Howard (2004)). Two UR3 manipulators are employed and controlled by ROS (Stanford Artificial Intelligence Laboratory et al. (2021)).

5.1 Fixed JSSP

In the first step, a fixed JSSP is considered. This should help to assess whether the presented approach is at all capable of generating a feasible solution. We start with a neural network that contains 4 hidden layers, each with 512 hidden neurons. The learning rate η is set to 1e-5 and the batch size to 256. The learning procedure includes 3.875 million training steps. In every episode, the current policy is used to evaluate the model. The makespan of the JSSP created by the policy in each episode is presented in Fig. 2. It takes over 200 episodes until the policy can create feasible schedules. The makespan decreases strongly from episode 200 to 300. After episode 300, the gradient decreases and only a slight change in makespan is detected. However, up to episode 700, the tendency is falling. The robots reach the lowest makespan of 633 time steps in episode 703. The FIFO heuristic achieves a makespan of 756 time steps. Although the FIFO is a very simple heuristic, the difference is very high, considering that the FIFO heuristic works without significant restrictions. The resulting schedules of episode 703 of the different jobs and the robots are visualized in Fig. 3 and Fig. 4. The numbers within the rectangles represent the job J_i.

Fig. 3 shows the utilization and the job sequence of the different stations. The stations are divided into machine stations (M) and buffer stations (B). For the sake of clarity, the starting and ending trays are omitted. Set up times are represented by the

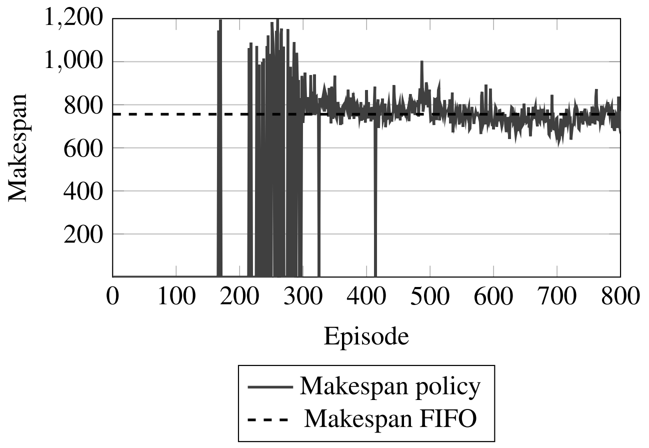

Fig. 2. Makespan of a fixed JSSP with 5 machine stations.

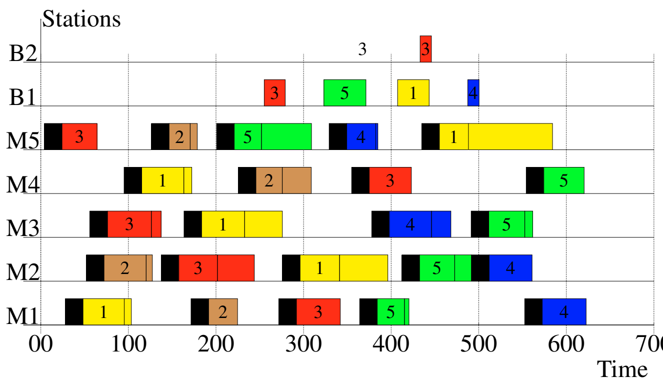

Fig. 3. Schedule of the stations of a fixed JSSP with 5 machines.

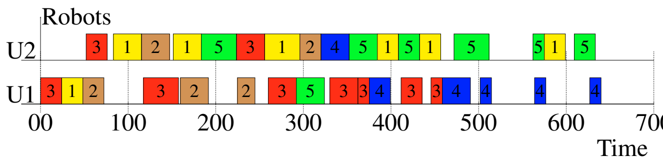

Fig. 4. Schedule of the robots of a fixed JSSP with 5 machines.

black rectangles within the figure. If no transport capacities are free or other jobs are prioritized, it might be possible that a job remains on a machine longer than necessary. This is symbolized by a line dividing a rectangle. It is noticeable that the jobs often remain at the machines longer than necessary. Especially the jobs J_1 (at M_2, M_3 and M_5), J_3 (at M_2) and J_5 (at M_5) remain idle for a long time after processing. This can be useful since the prioritization of other jobs can lead to a lower makespan. Furthermore, the policy leads to the fact that the buffer stations are used rarely. This occurs due to the time required for transportation between the stations and staying on unused machines leads to a better overall schedule. It should also be mentioned that the job J_3 is not placed at the buffer station B_2 before time step 400. After moving to the buffer station, the job is transported away directly. However, a reason for this might be to collect the additional reward for moving a job to a buffer station (see Tab. 1). A similar behavior is also noticed for the jobs J_3 (at B_1) and J_1 (at B_1). The jobs are transported to the buffer station although their next machine station is already free. To solve the problem, the local reward for transporting a job to a buffer station might be reduced.

Fig. 4 visualizes the job sequence of the two robots (U_1 and U_2). A process of a robot includes the transportation task to a specific job, the pick operation, the transportation task to the next station and the placing operation. These operations amount to overall time needed to carry out a specific operation with the job J_i. If a robot has no immediate consecutive task, it moves to its initial pose. It should be mentioned that up to time step 350 the jobs are well allocated between the two robots. After time step 350, robot U_1 only carries the jobs J_3 and J_4 and robot U_2 only operates the jobs J_1 and J_5. This behavior is a consequence of the boundary conditions. The remaining operations of these jobs are spatially separated. During this time, the robots can work undisturbed and need less time for transport operations. Only the transport of the job J_4 from the machine M_3 to the buffer station B_1 and the almost simultaneous transport of the job J_5 between the machines M_2 and M_3 require additional time steps to avoid collisions.

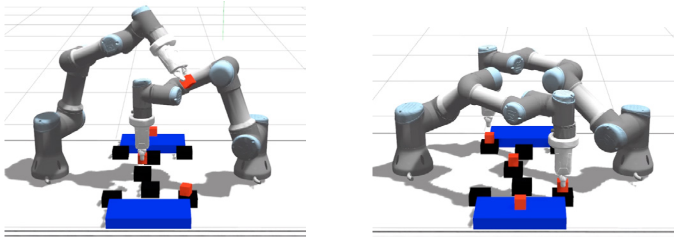

Fig. 5. Selected time frames from Gazebo simulation.

The schedule generated by the RL agent is transferred to the simulation environment with the two robotic manipulators. We can avoid deadlock situations and execute the obtained schedule. Selected time frames from the simulation environment can be seen in Fig. 5. The real benefit of dealing with RL occurs when arbitrary problems can be solved. This challenge is faced in the next subsection, where arbitrary and unseen JSSPs are considered. The objective is to solve complicated schedules in real-time after the training. So the required computation power is only needed during the training phase.

5.2 Arbitrary JSSPs

In the second step, arbitrary and therefore unseen JSSPs are considered. Since this leads to a larger complexity, the size of the neural network is increased. The network consists of 6 hidden layers, each with 1024 neurons. The learning rate and the batch size are adapted during the training. Starting with a learning rate of 1e-5 and a batch size of 256, the batch size is reduced in episode 120 to 128. In episode 170, the learning rate is also reduced to 1e-7 and the batch size is reset to 256. This should make the parameters more conservative with increasing training time and promote stability. The training process leads to 250 episodes and 12.8 million learning steps.

The results are calculated based on the schedules of the current policy in the validation phase. In Fig. 6, two different characteristics are considered: the average makespan of the 200 JSSPs at the left axis and the number of solved JSSPs on the right axis. The average makespan decreases until episode 100 rapidly. Then, the gradient decreases and only a small change in makespan is observed. The average makespan of the schedules is below 800 time steps. Looking at the number of solved JSSPs, 75 % of the problem instances are solved after 100 episodes. Until episode 170, the oscillations are high. After the change of the learning rate to 1e-7, the number of solved JSSPs is nearly constant. This is associated with the low learning rate and a small step size when updating the network. A generalization of 75 % is satisfactory but still needs improvement. From this point on, higher learning rates lead to divergences, suggesting that the training process should be improved. In the end, the policy can not reach the average makespan of the FIFO heuristic with around 670 time steps. Nevertheless, since the schedules of the FIFO heuristic are calculated without noteable constraints, the schedules can not be executed in the simulation environment. Thus, the policy is compared to infeasible schedules. If we take this fact into account, it becomes clear that the level of the policy is not worse, as the FIFO heuristic merely provides a lower bound on the makespan. It should also be mentioned that the tendency of the makespan is still falling and it can not be excluded that better schedules could be achieved with longer training. To achieve significantly better schedules, another exploration strategy than ε-greedy could be used. The policy is based on random solutions and therefore unfavorable schedules. The policy reaches the level of exemplary random solutions after 75 episodes. However, after a certain point there are no major improvements. Another exploration type like prioritized experience replay (Schaul et al. (2015)) or the use of noisy networks (Fortunato et al. (2017)) might result in a more efficient training. Another approach is the use of priority rules during exploration to achieve better initial solutions. In the end, the presented approach leads to feasible schedules for most instances of the presented JSSP.

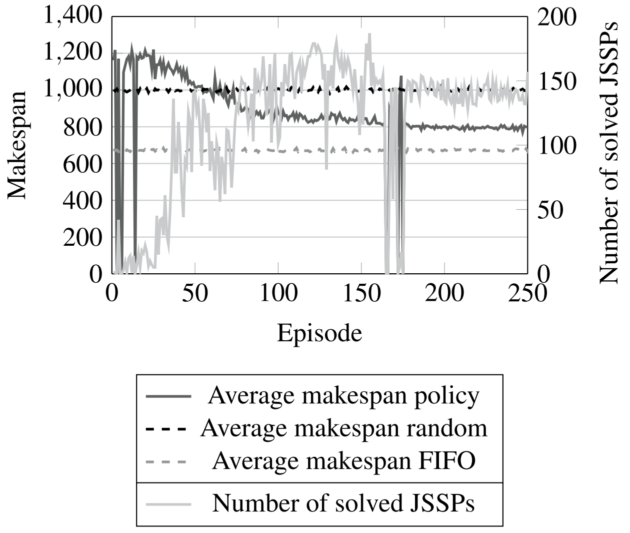

Fig. 6. Number of solved JSSPs and average makespan of arbitrary JSSPs with 5 machines.

6. CONCLUSION

The results imply that the presented approach leads to feasible schedules in the developed simulation environment. A specifically trained JSSP outperforms the FIFO heuristic by 20 %. For solving more complex arbitrary JSSPs, a difference in the performance of the applied deep Q-learning method and the FIFO heuristic has been observed. Since the FIFO heuristic can not always generate feasible schedules, the discrepancy is not worse but the generalization of the policy should be improved. The approach is able to solve 75 % of the problem instances. Due to combinatorics, another exploration type might be promising. Exploration strategies like prioritized experience replay or noisy networks offer possibilities for a more successful process. Furthermore, the RL approach offers other methods like double (van Hasselt et al. (2016)), dueling deep Q-learning (Wang et al. (2016)) or deep deterministic policy gradient (Lillicrap et al. (2015); Liu et al. (2020)) with possibly faster convergence. Application of the presented approach offers high potential for industrial processes since computation times for schedules can be decreased and more dynamic creation of schedules is possible. Feasible schedules with a trained neural network can be generated within short computation times. In future works, we plan to test different learning algorithms and integrate them into our production environment.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., and et. al (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. URL `https://www.tensorflow.org/`.

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

Banaszak, Z.A. and Krogh, B.H. (1990). Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. *IEEE Transactions on robotics and automation*, 6(6), 724–734.

Brucker, P. (2007). *Scheduling Algorithms*. Springer, Dordrecht.

Chen, R., Yang, B., Li, S., and Wang, S. (2020). A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Computers & Industrial Engineering*, 149, 106778.

Cunha, B., Madureira, A.M., Fonseca, B., and Coelho, D. (2020). Deep reinforcement learning as a job shop scheduling solver: A literature review. In A.M. Madureira, A. Abraham, N. Gandhi, and M.L. Varela (eds.), *Hybrid Intelligent Systems*, volume 923 of *Advances in Intelligent Systems and Computing*, 350–359. Springer International Publishing, Cham.

Dutta, S. (2018). *Reinforcement Learning with TensorFlow: A beginner's guide to designing self-learning systems with TensorFlow and OpenAI Gym*. Packt Publishing, 1st edition edition.

Fan, J., Wang, Z., Xie, Y., and Yang, Z. (2019). A theoretical analysis of deep q-learning. *arXiv preprint arXiv:1901.00137*.

Fortunato, M., Azar, M.G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. (2017). Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*.

Gabel, T. and Riedmiller, M. (2007). On a successful application of multi-agent reinforcement learning to operations research benchmarks. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 68–75. IEEE. doi:10.1109/ADPRL.2007.368171.

Gafur, N., Kanagalingam, G., and Ruskowski, M. (2021a). Dynamic collision avoidance for multiple robotic manipulators based on a non-cooperative multi-agent game. *arXiv*. URL `https://arxiv.org/abs/2103.00583`.

Gafur, N., Yfantis, V., and Ruskowski, M. (2021b). Optimal scheduling and non-cooperative distributed model predictive control for multiple robotic manipulators. *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Haupt, R. (1989). A survey of priority rule-based scheduling. *Operations-Research-Spektrum*, 11(1), 3–16.

Kingma, D.P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3, 2149–2154.

Lapan, M. (2018). *Deep reinforcement learning hands-on: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing, Birmingham, UK.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lin, L.J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4), 293–321.

Liu, C.L., Chang, C.C., and Tseng, C.J. (2020). Actor-critic deep reinforcement learning for solving job shop scheduling problems. *IEEE Access*, 8, 71752–71762.

Martínez, Y., Nowé, A., Suárez, J., and Bello, R. (2011). A reinforcement learning approach for the flexible job shop scheduling problem. In C.A.C. Coello (ed.), *Learning and Intelligent Optimization*, volume 6683 of *Lecture notes in computer science*, 253–262. Springer Berlin Heidelberg.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540), 529–533.

Panzer, M. and Bender, B. (2021). Deep reinforcement learning in production systems: a systematic literature review. *International Journal of Production Research*, 1–26.

Popper, J., Yfantis, V., and Ruskowski, M. (2021). Simultaneous production and agv scheduling using multi-agent deep reinforcement learning. *Procedia CIRP*, 104, 1523–1528.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676), 354–359.

SimPy (2021). Documentation of simpy. https://simpy.readthedocs.io/en/latest/. (Visited on 24.09.2021).

Stanford Artificial Intelligence Laboratory et al. (2021). Robotic operating system. https://www.ros.org. (Visited on 24.09.2021).

Sutton, R.S. and Barto, A. (2018). *Reinforcement learning: An introduction*. Adaptive computation and machine learning. The MIT Press, Cambridge, MA and London, second edition edition.

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2374–3468.

Wang, Y., Liu, H., Zheng, W., Xia, Y., Li, Y., Chen, P., Guo, K., and Xie, H. (2019). Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning. *IEEE access*, 7, 39974–39982.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, 1995–2003. PMLR.

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

Watkins, C.J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4), 279–292.

Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P.S., and Xu, C. (2020). Learning to dispatch for job shop scheduling via deep reinforcement learning. *arXiv preprint arXiv:2010.12367*.

Zhang, W. and Dietterich, T.G. (1995). A reinforcement learning approach to job-shop scheduling. In *IJCAI*, 1114–1120. Citeseer.

Zhou, Y., Hu, H., Liu, Y., and Ding, Z. (2017). Collision and deadlock avoidance in multirobot systems: A distributed approach. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(7), 1712–1726.