Simulation in Produktion
und Logistik 2021
Jörg Franke & Peter Schuderer (Hrsg.)
Cuvillier Verlag, Göttingen 2021

An Approach for Deep Reinforcement Learning for Production Program Planning in Value Streams

Ein Nutzungsansatz für Deep Reinforcement Learning zur Produktionsprogrammplanung in Wertströmen

Nikolai West, Florian Hoffmann, Lukas Schulte, TU Dortmund, Dortmund (Germany), nikolai.west@ips.tu-dortmund.de, florian.hoffmann@ips.tu-dortmund.de, lukas.schulte@ips.tu-dortmund.de

Victor Hernandez Moreno, University of Technology, Sydney (Australia), victor.hernandezmoreno@student.uts.edu.au

Jochen Deuse, TU Dortmund, Dortmund (Germany) and University of Technology, Sydney (Australia), jochen.deuse@ips.tu-dortmund.de,

Abstract: The application of Reinforcement Learning (RL) methods offers a potential for improvement in operational Production Program Planning. Numerous influences and domain-specific practices characterize the multi-dimensional planning paradigm. RL can support human planning personnel in the determination of optimal production parameters. This requires a suitable abstraction of the overall system by means of simulation and subsequent optimization by a self-learning agent. In this paper, the authors present an application example for sequence planning using RL. The case study includes a discrete-event simulation built with *SimPy* that is trained by a Duelling Deep-Q-Network implemented in *PyTorch*. Finally, the suitability of two reward functions is discussed. The authors fully provide the case study via GitHub.

1 Introduction

During *Production Program Planning* (PPP), a company must define a market-driven and future-oriented selection of products to manufacture in a certain period according to type and quantity. While in long- and mid-term PPP a general program outline of products is defined, short-term PPP defines specific production quantities and variant sequences. A major goal is to find optimal product sequences and batch sizes for a satisfactory production program. To enable optimal and short-term PPP, it is no longer sufficient under increasing competitive factors to operate purely based on real-world planning data. Due to variability-influenced behaviour, anticipation of a real-world production system is a challenging task that requires in depth domain and system knowledge. Computer-based approaches to support planning production capacities often include more complex solutions, such as event-discrete simulation models or

mathematical replications. *Machine Learning* (ML) represents an increasingly popular approach to achieve targeted results in such systems. In this paper, we focus on the usage of *Reinforcement Learning* (RL), a sub-domain of ML, together with a simulation model to identify beneficial scheduling plans for mixed product lines. Our goal is to develop and compare two different reward functions for demonstration.

2 Fundamentals and related-work

2.1 Short-term production program planning

Within the manufacturing domain, there are numerous tasks to solve with regard to PPP, starting from aggregated PPP, line configuration, followed by PPP and the subsequent sequencing for companies. Line configuration represents the deterministic basis for the new or reconfiguration of a production line with regard to its capacity. PPP carries out the scheduling of customer orders. Both planning problems are subject to a medium-term time horizon, which is why planning processes here have a direct influence for achieving on-time delivery in a production system. Sequence planning, on the other hand, serves a short-term time horizon, whereby corresponding planning errors show rapid effects on production, which can lead to undesirable developments with regard to on-time delivery or temporary capacity requirements (Volling 2009). The efficiency of production is thus significantly influenced by the choice of sequence planning (Dörmer and Günther 2013). As of yet, the choice of a planning approach is highly dependent on the respective target variable selected for optimization. Boysen et al. (2009b) and Gujjula et al. (2011) differentiate existing scheduling approaches according to the three groups of *car sequencing, mixed-models* and *level scheduling*. Car sequencing was formulated by Parrello et al. (1986) and, similar to mixed-model sequencing that was formulated by Kilbridge (1963), it focuses on the necessary working time requirements of orders. The level scheduling of the Toyota production system, on the other hand, is designed to achieve a uniform material requirement (Boysen et al. 2009a). Existing approaches can also be assigned with regard to the distinction according to logistical target variables. With regard to on-time delivery, classic methods such as *First-in-Frist-Out*, short FIFO, (Wiendahl 1997) and *minimum residual slack* (Lödding 2016) can be mentioned. On the other hand, methods such as setup *time-optimizing sequencing* (Lödding 2016) and the *extended work in next queue* method (Conway 2012) focus on optimizing the performance target. According to Conway (2012) the *shortest operation time rule*, there are additional possibilities to reduce the inventory, the average lead time, the average delivery delay and to increase the delivery reliability.

Summarizing, PPP can benefit from a multitude of industrial methods that guide and support the determination of optimized sequencing with regard to different target conditions. In addition, numerous methods simplify the management of potential solution spaces. A mayor task is to narrow down the problem in terms of the targeted solution and solve it in an appropriate duration. Due to model abstraction, the risk of simplifying disturbing influences of the solution parameters remains. In this paper, we target the use of automated algorithms based on RL for it offers a promising approach to enable computer-aided sequencing in industrial value streams. RL enables experience-based planning recommendations through computer-based learning without the need for deep subject or domain knowledge. However, while other

approaches such as evolution strategies or genetic algorithms offer promising solutions as well, we deliberately chose to implement a RL model for the case study.

2.2 Reinforcement learning in production program planning

The research area concerning RL and the related *Deep Reinforcement Learning* (DRL) is rapidly evolving. It finds applications in numerous areas, such as the gaming industry, robotics, natural language processing, computer vision or system control (Li 2018). Along with *Supervised Learning* and *Unsupervised Learning,* RL is the third branch of ML, which provides automated methods to detect patterns in data and to use them to fulfil organizational tasks (Bishop 2006). In the domain of ML, RL is concerned with the task of learning how agents ought to take sequences of actions in an environment in order to maximize cumulative rewards (François-Lavet et al. 2018). A network for DRL is characterized by a succession of multiple processing layers, where each layer consists of a non-linear transformation and the sequence of these transformations leads to learning different levels of abstraction (Erhan et al. 2009). According to their learning policy, methods for DRL can be divided into three groups: V*alue-based methods*, e.g. Q-Learning or Deep Q-Networks, *policy gradient methods*, e.g. Stochastic Policy Gradient or Actor-Critics-Methods, and *model-based methods*, e.g. purely model-based or hybrid methods (François-Lavet et al. 2018).

Using a virtual replication of the real world, RL helps to find solutions to complex problems, where a problem's solution does not need to be known a priori. Due to its widespread application, there are numerous examples of successful industrial usage. For example, Freitag and Hildebrandt (2016) address multi-objective optimisation in order to improve dispatching rules for production planning and control. Scholz-Reiter and Hamann (2008) optimise the material flow to significantly reduce inventories, lead times and its variation coefficient. Wang and Usher (2005) address the agent-based finding of a dispatching rule for single machine jobs. Zhang et al. (2011) develop a multi-step RL algorithm to solve a scheduling problem with a throughput-based objective. Qu et al. (2016) adapt RL to optimally schedule a multi-step manufacturing system for multiple product types of diverse machines and multi-skilled workforce. Wang (2020) considers a job-shop scheduling problem solved weighted Q-learning algorithm. The selected cases are exemplary, for we do not aim to provide a comprehensive review of RL application. For this, we refer to Shyalika et al. (2020) who provide a detailed review of existing approaches for RL techniques in scheduling tasks. Due to the technical challenge and the economic potential, we derive a need for research that led to the following approach.

3 Reinforcement learning approach

The principle of RL is based on the training of an ML algorithm, hereafter referred to as an agent, that uses constant feedback to interact with an environment (François-Lavet et al. 2018). We display the general principle of RL systems in *Figure 1*. An agent acts independently during the learning phase and it tries to successively optimise its own behaviour by having its actions rewarded or punished by the environment. Starting from a discrete point in time ($t = 0,1,2 \dots, n$), the agent is in a observable state s_t. Then, the agent can select an action a_t from predefined set of actions to perform in an environment. Each action may influence the environment. Through this action, the environment enters a new state s_{t+1} at the following time $t + 1$. In case

the state is considered beneficial, the agent receives the reward r_t to reinforce the selected action (Sutton and Barto 1998). The changes of the environment are related to the selected actions of the agent. This allows the training of reward maximizing strategies, also called a policy in order to maximise its total numerical reward (Sim et al. 2003). Through repeated agent-environment interactions, the agent learns a useful strategy that leads to more rewarding states as they are reinforced by the environment.

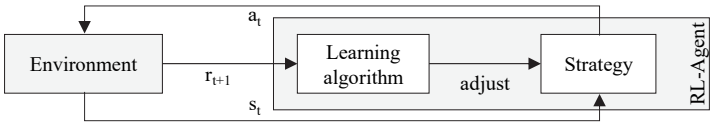

Figure 1: *Operating principle of Reinforcement Learning (Sutton and Barto 1998)*

For application in operational PPP, the main task of the system is to determine an ideal sequence of products for the PPP. From a production engineering perspective, the main task lies in the correct mapping of the overall system by means of simulation and the definition of state and reward. From a data analytics perspective, the main task is to select an appropriate policy and to specify a function to determine the reward. We explain and demonstrate the approach to DLR in PPP in practice with a case study.

4 Case study

In the following section, we present an exemplary application of DRL in operational PPP using a case study. This section is divided into three parts: First, we present the design of the use case. Second, we explain the structure of the chosen DRL model. Third, we show and discuss the results obtained from two different reward functions.

4.1 Design and implementation of the environment

To demonstrate the potential of DRL in operational PPP, we designed a production system with an implied sequence scheduling trade-off (see *Figure 2*). The exemplary system manufactures two products A and B on a flow line with five machines M_i. While the process time $pt_{M_i}^A$ for manufacturing product A decreases further down in the value stream, the process time $pt_{M_i}^B$ for manufacturing product B increases. All stations are linked via a buffer b_i, with an additional buffer r_1 with raw material at the start of the line. As the last buffer in the value stream, b_5 acts as the transfer point to the customer. No inventory limits are set for the buffers and the system starts empty. All machines operate according to the FIFO principle. For the software implementation, we use the package `SimPy` (V. 4.0.1) for Event Discrete Simulation in Python (Matloff 2008). In the code documentation of this paper (West 2021), the simulation of the described factory system is implemented as *factory_simulation.py*.

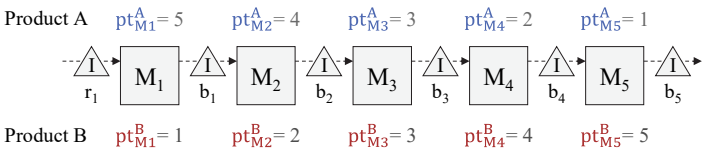

Figure 2: *Structural layout of the designed use case for Reinforcement Learning*

The chosen study design aims at solving the following planning task: The total output of products A and B has to be maximized for a given time period. If the PPP forces a sequence of one product type, either A or B, we expect an output of one product in five time steps. This is associated with availability losses, as the potential overall output decreases due to the respective bottleneck, either $pt_{M_1}^A$ for a monotonous production of A, or $pt_{M_5}^B$ for the same case of B. Alternating the production of A and B can balance the varying production times, and thus reduce the downtimes of the machines M_i. This strategy allows for the production two products in six time steps. We show the expected ideal state as the overall goal in *Figure 3*. After an initial settling period, we expect an alternating output of product A and B in the optimal PPP case. For manageability, we limit the simulation duration t to a total of 60 time steps.

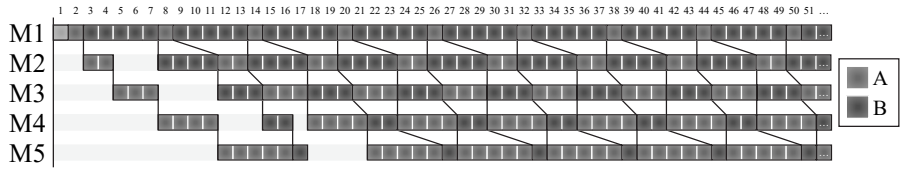

Figure 3: *Expected sequence for efficient product sequencing in the use case*

4.2 Model selection and implementation for job scheduling

The next task is to select and implement a model. Each of the three groups of DRL provides a potential approach. For the case study, we implement a value-based method, notwithstanding the fact that policy-gradient or model-based methods offer equally appropriate ways to approach the planning task (Beitelspacher et al. 2006). As a model-free approach, value-based techniques allow us to leverage the simulation model without the need to learn an accurate model of the real-world system behaviour.

Due its success in recent years, we select a *Deep-Q-Learning* (DQN) based approach for the case study. *DQN* represents a further development of so-called *Q-Learning* with an extension using deep neural networks and a method referred to as experience replay (Mnih et al. 2015). For our case study, we chose to implement a variant of DQN with a duelling network architecture (DDQN), 48 neurons per layer and rectified linear unit as activation function. We refrain from an in-depth description of the DDQN and instead refer to François-Lavet et al. (2018) for an overview of DQN and to Wang et al. (2015) for an explanation of the duelling architecture. For software implementation, we use the package `PyTorch` (V. 1.8.1) as ML framework in Python. We modelled the interaction between agent and factory according to Allen and Monks (2020), and follow the structure of the established *OpenAI Gym* environments. In the code of this paper (West 2021), the DDQN and the training program for the agent-factory-interaction is implemented as *factory_agent.py*.

As illustrated in *Figure 1*, we need to define an action space $\vec{a}$, an observation state $\vec{s}$ and a reward function rf for the interaction between the simulation model and the DDQN agent. At each time step of the simulation, the agent has to choose between one of three actions to influence the environment: It either triggers the production of product A, or the production of B. Additionally, it may remain in an idle state and thus manufacture no product. The process runs via virtual release of raw material for the two products A and B for the raw material buffer r_1. We define $\vec{a} = [0,1,2]$, with

- $a = 0$: The DDQN agent choses the production of product A ('A'),
- $a = 1$: The DDQN agent choses the production of product B ('B'), and
- $a = 2$: The DDQN agent chose to be idle and produce no product ('0').

In principle, all variable parameters in the simulation model are possible observations for $\vec{s}$, with buffer levels b_i being the most promising from a production engineering point of view. Other influencing variables such as machine states, e.g. running, blocked or starved, or production key figures, e.g. average cycle time or throughput, would also be plausible observations. Since we refrain from influencing the process times pt_{M_i} by a variability factor in the use case, the demand characteristic for semi-finished goods is deterministic. Thus, it is sufficient to consider the initial buffer r_1 in $\vec{s}$ and we avoid additional influences by the buffer levels b_1 to b_5. To better distinguish between demands for product A and product B, we track the available amount of raw material for A and B in R1 with two separate variables. We define $\vec{s} = [r_1^A, r_1^B]$, with

- $r_1^A = 0$: Amount of raw material for Product A in buffer r_1 (initially set to zero),
- $r_1^B = 0$: Amount of raw material for Product B in buffer r_1 (initially set to zero).

Selecting the action 0 ('A') releases one unit of raw material r_1^A for A in r_1 and enables production. Analogous, action 1 ('B') releases raw material for B in r_1. Technically, we implemented the separation of material types using sets of dual container in *SimPy*.

In the section 4.3, we use and compare two different reward functions rf_1 and rf_2. Both functions enable the determination of a time-dependent reward r during the training runs and they operate on the same production cost factors within the environment. We set material cost for a raw material release at 35 *reward units* (RU). We also assume an inventory holding cost of 1 RU per product and buffer. Material costs are applied in the moment the agent chooses to release raw material through the appropriate actions, i.e. 0 or 1. Storage costs are applied once per time step. According to the use case, machine M_5 acts as transfer point to the customer. Thus, after completion of a product, either A or B, rf_1 returns a fixed reward of 300 RU, analogous to a product sale. We build rf_2 to reflect a customer preference for changing product deliveries. The function returns a reward at the same stage, but the value of the reward is based on the prior deliveries. Starting at 200 RU and set in the limit of 100 to 400 RU, rf_2 repeatedly applies a factor of 1,5 for each consecutive switch between products, as well as a factor of 0,75 for each consecutive production of the same product type, either A or B. Both reward functions should be able to identify profitable production sequences, with rf_2 specifically designed to induce the sequencing behaviour that we displayed in *Figure 3*.

4.3 Training results using two reward functions

Next, we show the training results of the DDQN agent in the designed case study after 1000 training runs. We consider the results of both reward functions separately.

4.3.1 Simulation and DDQN agent using reward function rf_1

Figure 4 shows the exploration rate and total reward of 1000 training runs of the DDQN agent in the simulation environment with reward function rf_1. Each run represents a full simulation over the set training duration of $t = 60$. Following an *Epsilon-greedy* exploration strategy, after 10 random runs, we decrease the exploration rate using a decay of $0{,}9995$ to a minimum rate of $0{,}01$. After initial

fluctuations due to exploration, the overall reward reaches a relatively stable plateau at circa 850 RU. Besides random downward outliers, the reward curve also shows recurrent solutions around 1000 RU. It is therefore apparent that the agent has found a reliable strategy to achieve a positive reward, but it may not be the optimal strategy.

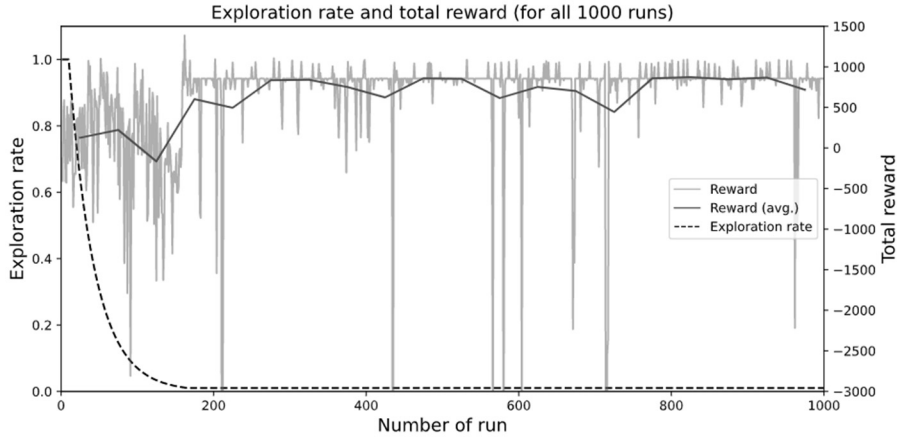

Figure 4: *Exploration rate and total reward of run 1000 with reward function* rf_2

The plot of an exemplary run after 1000 training runs in *Figure 5* confirms this assumption. After initial costs, products *A* are continuously completed after about 15 time steps. The running costs become minimal the accumulated reward curve takes on a staircase-like shape. The agent uses a strategy of exclusively producing product *A*, to reliably achieve a positive reward. It has learned the systems output behaviour and the agent thus triggers the production of product A in time intervals that result in minimal inventories while finishing A in the shortest possible succession.

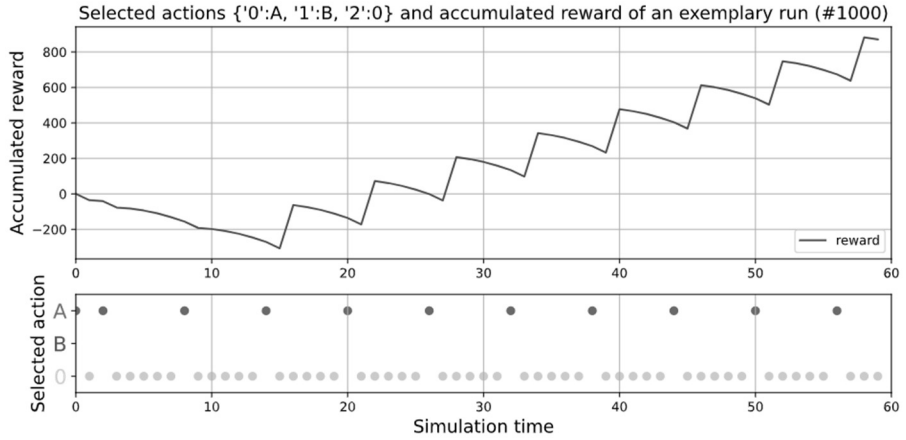

Figure 5: *Exemplary results after 1000 training runs, displaying all selected actions and the accumulated reward with reward function* rf_1

Due to our preliminary considerations, we know a more optimal solution of the planning task. Furthermore, the upward deviations in Figure 5 indicate better

solutions. For full disclosure, we emphasize that such solutions might also be found using rf_1 by further tuning the training parameters, e.g. by using a lower exploration decay or by increasing the number of training runs. Additionally, from the number of alternative DRL models, some might also provide a more targeted selection of actions.

4.3.2 Simulation and DDQN agent using reward function rf_2

For comparability with rf_1, we again use the same exploration strategy and present exploration rate and absolute rewards in *Figure 6*. Again, there are initially expected fluctuations in reward due to exploration. However, unlike before, the total reward reaches a stable range much later, about 150 training runs later. Downward outliers also occur much more frequently, whereas there are almost no spikes above a recognizable maximum value of around 2700 RU. A higher average reward for $rf_2$was to be expected due to the design of the reward function, however, the achieved threshold indicates the identification globally optimal strategy.

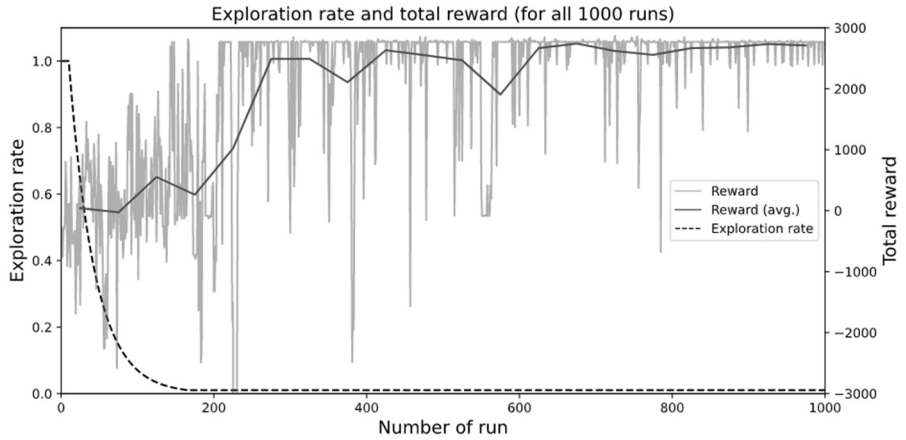

Figure 6: *Exploration rate and total reward of run 1000 with reward function rf_2*

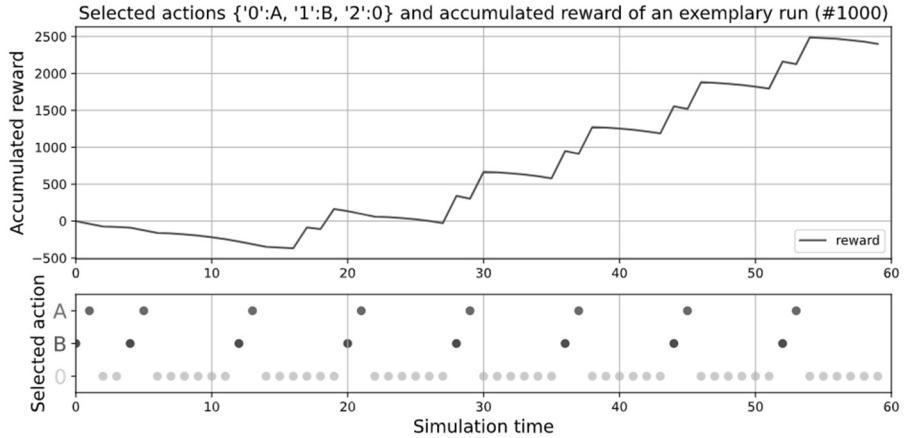

Figure 7: *Exemplary results after 1000 training runs, displaying all selected actions and the accumulated reward with reward function rf_2*

In *Figure 7*, we show a visualization of an exemplary run after 1000 training steps. The plot of the accumulated reward again shows a phase with initial costs up to time step 15. The subsequent reward course has a double staircase-like structure with two successive ascents after another descending phase. The double increases are due to alternate completion of product B and A. In the consideration of chosen actions, it becomes apparent that the agent finds a seemingly ideal sequence of possible actions. The chosen sequence corresponds to the expectation laid out in *Figure 3*. In addition to the ideal sequence, using rf_2, the DDQN agent additionally succeeds in releasing orders at cost-maximizing time points.

5 Conclusion and outlook

In this paper, we demonstrated an exemplary approach to use DRL for tasks of PPP. Thereby, this contribution serves to affirm the potential benefits of combining DRL with PPP tasks, as stated at the beginning. From both a scientific and an economic point of view, there is a need for further development. The technical realization of a generally valid reward function for a target-oriented and variability-proof determination of ideal planning states is required. In addition, the approaches that have to date only been realized as theoretical implementation examples should be confirmed in real-world scenarios with economically measurable benefit potential.

In addition to its postulated use as a supporting tool in product sequence planning during PPP, the demonstrated DRL approach offers additional usefulness for advanced control methods that follow an inventory-oriented order release strategy.

References

Allen, M.; Monks, T. (2020): Integrating Deep Reinforcement Learning Networks with Health System Simulations. In *arXiv:2008.07434*, pp. 1–6.

Beitelspacher, J.; Fager, J.; Henriques, G.; McGovern, Amy (2006): Policy gradient vs. value function approximation. A reinforcement learning shootout. In *Technical Report of the School of Computer Science, University of Oklahoma* (06-001).

Bishop, C. (2006): Pattern recognition and machine learning. New York: Springer.

Boysen, N.; Fliedner, M.; Scholl, A. (2009a): Level scheduling for batched JIT supply. In *Flexible Services and Manufacturing Journal* 21 (1-2), pp. 31–50.

Boysen, N.; Fliedner, M.; Scholl, A. (2009b): Sequencing mixed-model assembly lines. Survey, classification and model critique. In *EJOR* 192 (2), pp. 349–373.

Conway, R. W. (2012): Theory of scheduling: Dover Publications.

Dörmer, J.; Günther, H. (2013): Production program planning with multi-variant flow production. Dissertation. Wiesbaden: Springer Gabler.

Erhan, D.; Bengio, Y.; Courville, A.; Vincent, P. (2009): Visualizing higher-layer features of a deep network. In *University of Montreal* 1341 (3).

François-Lavet, V.; Henderson, P.; Islam, R.; Bellemare, M.; Pineau, J. (2018): An introduction to deep reinforcement learning. In *Foundations and Trends in Machine Learning* 11 (3-4), pp. 1–102.

Freitag, M.; Hildebrandt, T. (2016): Automatic design of scheduling rules for complex manufacturing systems by multi-objective simulation-based optimization. In *CIRP Annals* 65 (1), pp. 433–436.

Gujjula, R.; Werk, S.; Günther, H.-O. (2011): A heuristic based on Vogel's approximation method for sequencing mixed-model assembly lines. In *International Journal of Production Research* 49 (21), pp. 6451–6468.

Kilbridge, M. (1963): The assembly line model-mix sequencing problem. In *Proceedings of the International Conference on Operations Research* 3.

Li, Y. (2018): Deep reinforcement learning. In *arXiv:1810.06339*, pp. 1–150.

Lödding, H. (2016): Production control method. Basics, description, configuration. 3. Edition. Berlin, Heidelberg: Springer Vieweg.

Matloff, N. (2008): Introduction to discrete-event simulation and the SimPy language. Davis, CA: UC Davis.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.; Veness, J.; Bellemare, M. et al. (2015): Human-level control through deep reinforcement learning. In *Nature* 518 (7540), pp. 529–533.

Parrello, B.; Kabat, W.; Wos, L. (1986): Job-shop scheduling using automated reasoning. A case study of the car-sequencing problem. In *J. o. A. Reasoning* 2 (1).

Qu, S.; Wang, J.; Govil, S.; Leckie, J. O. (2016): Optimized adaptive scheduling of a manufacturing process system with multi-skill workforce and multiple machine types. In *Procedia CIRP* 57, pp. 55–60.

Scholz-Reiter, B.; Hamann, T. (2008): The behaviour of learning production control. In *CIRP Annals* 57 (1), pp. 459–462.

Shyalika, C.; Silva, T.; Karunananda, A. (2020): Reinforcement learning in dynamic task scheduling. A review. In *SN Computer Science* 306 (1), pp. 1–17.

Sim, S.; Ong, K.; Seet, G. (2003): A foundation for robot learning. In *IEEE International Conference on Control and Automation* 4, pp. 649–653.

Sutton, R.; Barto, A. (1998): Reinforcement learning. An introduction. Cambridge: MIT Press.

Volling, T. (2009): Order-related planning for multi-variant series production. Zugl.: Braunschweig, Techn. Univ., Diss., 2008. 1. Edition. Wiesbaden: Gabler (Gabler Edition Science Production and Logistics).

Wang, Y.-F. (2020): Adaptive job shop scheduling strategy based on weighted Q-learning algorithm. In *Journal of Intelligent Manufacturing* 31 (2), pp. 417–432.

Wang, Yi-Chi; Usher, John M. (2005): Application of reinforcement learning for agent-based production scheduling. In *Engineering Applications of Artificial Intelligence* 18 (1), pp. 73–82.

Wang, Z.; Schaul, T.; Hessel, M.; van Hasselt, H.; Lancot, M.; Freitag, N. de (2015): Dueling Network Architectures for Deep Reinforcement Learning. In *arXiv:1511.06581*, pp. 1–15.

West, N. (2021): Code to Case Study. Available online at github.com/ nikolaiwest/2021-reinforcement-learning-asim.

Wiendahl, H. P. (1997): Production control and Logistical control of manufacturing processes based on the Trichtermodell. München: Hanser.

Zhang, Zhicong; Zheng, Li; Hou, Forest; Li, Na (2011): Semiconductor final test scheduling with Sarsa(λ,k) algorithm. In *European Journal of Operational Research* 215 (2), pp. 446–458.