Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 54-1 (2021) 793–798

Modeling Production Scheduling Problems as Reinforcement Learning Environments based on Discrete-Event Simulation and OpenAI Gym

Sebastian Lang*, Maximilian Kuetgens, Paul Reichardt***, Tobias Reggelin******

**Fraunhofer Institute for Factory Operation and Automation IFF, 39106 Magdeburg, Germany (Tel: +49-391-67-57299; e-mail: sebastian.lang@iff.fraunhofer.de)*
***Otto von Guericke University, 39106 Magdeburg, Germany (e-mail: maximilian.kuetgens@ovgu.de)*
****Otto von Guericke University, 39106 Magdeburg, Germany (Tel: +49- 391-67-57323; e-mail: paul.reichardt@ovgu.de)*
*****Otto von Guericke University, 39106 Magdeburg, Germany (Tel: +49- 391-67-54980; e-mail: tobias.reggelin@ovgu.de)*

Abstract: Reinforcement learning (RL) is an emerging research topic in production and logistics, as it offers potentials to solve complex planning and control problems in real time. In recent years, many researchers investigated RL algorithms for solving production scheduling problems. However, most of the related articles reveal only little information about the process of developing and implementing RL applications. Against this background, we present a method for modeling production scheduling problems as RL environments. More specifically, we propose the application of Discrete-Event Simulation for modeling production scheduling problems as an interoperable environments and the Gym interface of the OpenAI foundation to allow a simple integration of pre-built RL algorithms from OpenAI Baselines and Stable Baselines. We support our explanations with a simple example of a job shop scheduling problem.

Copyright © 2021 The Authors. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0)

Keywords: Reinforcement Learning, Production Scheduling, Production Planning and Control, Discrete Event Modeling and Simulation, OpenAI Gym, Artificial Intelligence, Deep Learning, Neural Network,

1. INTRODUCTION

Reinforcement Learning is a paradigm of machine learning (ML) and an emerging research topic in the field of production and logistics.

1.1 Background: Reinforcement Learning in a Nutshell

Reinforcement Learning (RL) pursues to train an agent, which maps states of its surrounding environment (input) to actions (output). The agent can be any kind of parameterizable model, for instance a table or an equation. In deep reinforcement learning (DRL), however, the agent is represented by a deep neural network. The agent and environment are interacting in a loop. Thus, every time the agent performs an action, depending on the current state of the environment, the environment changes into the next state and the agent performs the next action. This process repeats, until the environment terminates in a final state. In contrast to supervised learning, RL methods do not require labeled training data (i.e. an expected output to a given input). Instead, a reward function evaluates the actions of the agent with positive or negative rewards. The agent tries to maximize the cumulative obtained reward over time by learning to predict the best action to a given state. (Arulkumaran et al. 2017; François-Lavet et al. 2018; Sutton and Barto 2018)

1.2 Related Research

Agents trained by RL are able to compute decisions in real-time, while taking into account information of their environment. Furthermore, they continuously improve by adapting their parameters, when encountering unknown system states. Both features make RL methods attractive for decision-making problems in production and logistics. Recently, researchers have been investigating the application of RL methods for production scheduling problems. This research trend might be motivated, on the one hand, by the widely recognized successes of Mnih et al. (2013; 2015) in applying DRL methods to control problems in videos games and, on the other hand, by current challenges in production and logistics, such as shorter production lead times, shorter product life cycles, a greater product diversity and increasing uncertainties in supply chains (Gershwin 2018; Lang et al. 2019).

The majority of related articles, such as (Shiue et al. 2018; Lin et al. 2019; Luo 2020; Wang et al. 2020), describe the application of RL for selecting scheduling rules depending on the state of the production system. In this case, the agent's decisions only indirectly affect the scheduling of jobs, whereas the selected scheduling rule is responsible for job dispatching. Another way of using RL for production scheduling is to let the agent decide directly about the scheduling of jobs. This approach is reported, for instance, by Stricker et al. (2018) and

2405-8963 Copyright © 2021 The Authors. This is an open access article under the CC BY-NC-ND license.
Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2021.08.093

Waschneck et al. (2018) for the semiconductor industry, where the agents are responsible for both, the allocation and sequencing of jobs.

There are many more articles about RL for production scheduling. We provide a comprehensive overview in our previous articles (Lang et al. 2020a; Lang et al. 2020b).

1.3 Objective and Outline

Most of the related research, however, provides only little details about the process of developing and implementing RL applications. According to our experience, the design of the agent's environment requires most time and effort, as there are already many open-source libraries providing pre-built RL algorithms, for example OpenAI Baselines (Dhariwal et al. 2017) and Stable Baselines (Hill et al. 2018).

Against this background, we present a method for modeling production scheduling problems as RL environments. More specifically, we propose the application of Discrete-Event Simulation (DES) for modeling a production scheduling problem as an interoperable environment. Thereafter, we implement the Gym interface of the OpenAI foundation (Brockman et al. 2016) for the DES model, which simplifies the deployment of pre-built RL algorithms of OpenAI Baselines and Stable Baselines.

The further paper is organized in four sections. Section 2 describes the procedure of modeling a production scheduling problem as DES model. In section 3, we explain how agents trained by RL can be deployed for scheduling decisions within a given DES model. Section 4 provides a concept for integrating an existing DES model with the Gym interface of OpenAI. Section 5 is dedicated to the conclusion and outlook.

2. MODELING OF PRODUCTION SCHEDULING PROBLEMS WITH DISCRETE-EVENT SIMULATION

Production scheduling is a planning and optimization process, which addresses two problems: (1) The allocation of jobs to machines and (2) the sequencing of jobs that are allocated to a machine. The goal is to determine a production schedule, which satisfies certain quality criteria, measured by a single or several objectives, for instance the makespan (i.e. the time between the start of the production and the completion of the last job). Scheduling problems can be modeled as queuing systems, in which entities visit a number of capacity-restricted resources, where they complete a set of time-consuming operations, before leaving the system. If a resource is blocked by an entity, upstream entities wait in a preceding queue until the resource becomes available. An entity might be a job to produce. A resource might be a machine or a workstation, which performs certain operations on the job.

Discrete-Event Simulation (DES) is a method for modeling and simulating queuing systems and processes. A particularity of DES is that model states change not continuously, but only at discrete and possibly random points in time. State changes are associated with pre-scheduled events, which are organized in several list and executed by an event handler (Banks et al. 2010; Schriber et al. 2017). The occurrence of events is subject to the process modeled by the user. DES is frequently used in production and logistics, since most production and logistics processes have a discrete nature (Scholz-Reiter et al. 2008).

The way DES models are developed depends on the chosen DES software. Many off-the-shelf tools, e.g. ARENA or AnyLogic, offer a graphical user interface (GUI) for creating process chart-like models via drag-and-drop. A problem of commercial DES tools is that they are usually closed-source platforms. Depending on the availability of suitable interfaces, it might be challenging to integrate commercial DES tools with one of the established deep learning frameworks Tensorflow or PyTorch, since both frameworks rely on Python as front-end language for developing ML models. Accordingly, we recommend the use of a Python-based DES library, for instance SimPy (Matloff 2008) or Salabim (van der Ham 2018), to implement production scheduling problems as simulation models. However, a drawback of both libraries might be that they do not provide a GUI for modeling, and thus the development of models is based on writing python code.

Fig. 1 on the next page therefore provides a simple example for modeling production scheduling problems with the DES method. The modeled system is a job shop scheduling problem consisting of a source, three resources with preceding queues and a sink. A job shop scheduling problem requires a certain number of entities (jobs) to undergo a series of operations on a certain number of resources. In the example given, entities can complete any operation on any resource. The design of the process logics for the source, entities and resources is closely related to the long-established programming language SIMULA (Dahl and Nygaard 1966), which also inspired the way of creating models in Salabim. A main feature of SIMULA is the ability to activate, passivate and hold objects. Any object is initially active after creation and executes its process logic in a loop. If an object passivates another object, the passive one stops the execution of the process logic at the current step for uncertain time. Passive objects are only able to continue the execution of the process logic after being reactivated by another object. Holding an object also suspends the corresponding process logic. In contrast to the concept of passivating, however, the suspension of the process is subject to a deterministic or random, yet certain and fixed time

In the example of Fig. 1, the source generates entities according to a user-defined interarrival time. Every time an entity was created, the entity performs the following steps until it completes all necessary operations. First, the entity identifies the next operation to be completed. An operation might be defined by the processing time that each of the three resources requires for performing the operation. Second, the entity requests the next resource to enter. This is implemented by assigning the requesting entity (*self*) to the flag variable 'invoking object'. At this point, the DES model may stop the execution and delegates the allocation decision to an external agent trained by RL. Third, the entity enters the queue of the resource to which it was allocated. If the selected resource is currently passive, the entity reactivates the resource. Thereafter, the entity passivates itself until it has completed the current operation on the allocated resource. A resource

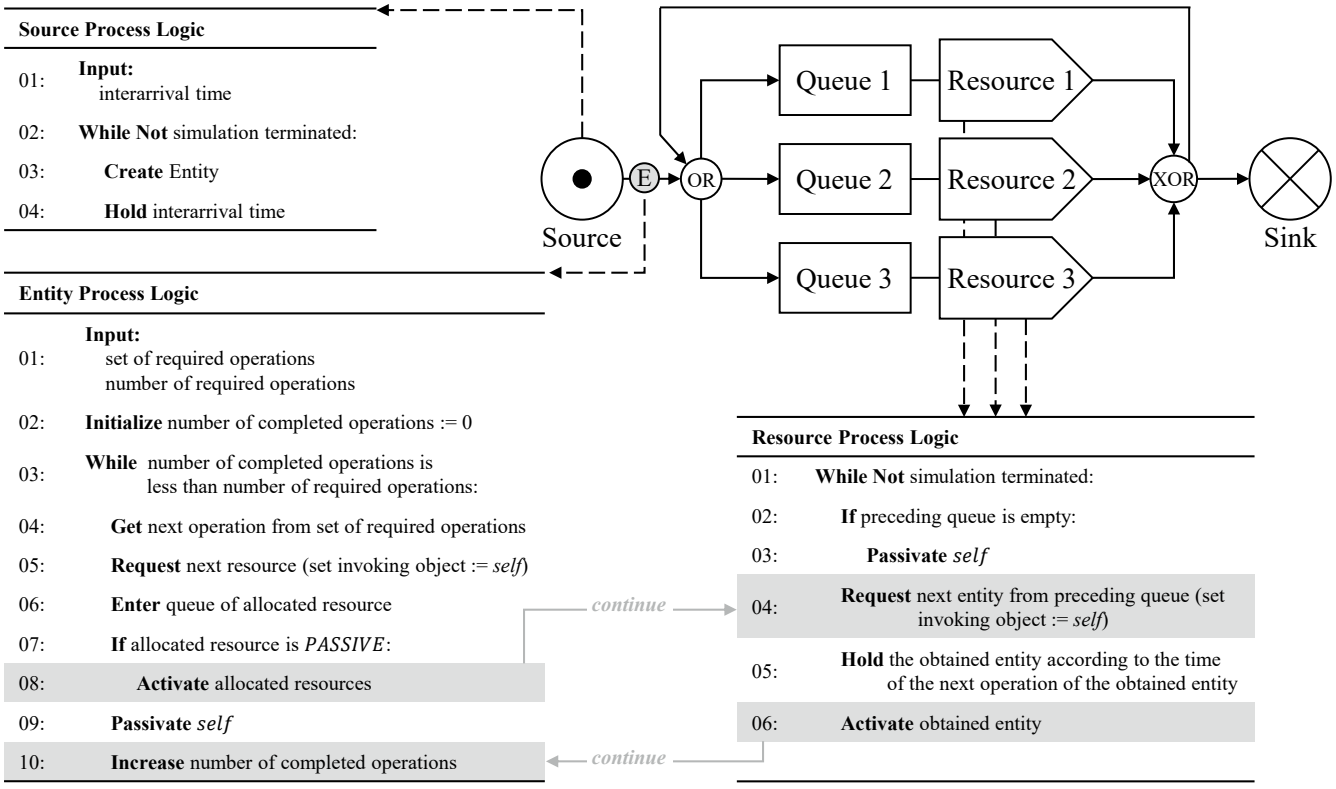

Fig. 1. Concept for implementing an exemplary job shop scheduling problem as DES model.

remains active, as long as entities are waiting in the preceding queue. If the process logic of a resource is active and the resource itself is idle, the resource requests the next entity from the preceding queue. According to the request for allocation decisions, we use the flag variable 'invoking object' to save the requesting resource (*self*) and to notify an agent about a pending sequencing decision. After completing the operation, the resource reactivates the obtained entity. In the next step, the process logic of the entity checks, whether the number of completed operations corresponds to the number of required operations. If false, the entity requests the next operation to be completed and the next resource to execute the operation. Otherwise, the entity travels to the sink, where it leaves the system.

Despite the simplicity of the given example, the modeling concept shown here is adoptable to a variety of more complex scheduling problems. In addition, it requires only a few extensions to consider further aspects as, for instance, product families and sequence-dependent setup times.

3. DECISION-MAKING IN PRODUCTION SCHEDULING WITH REINFORCEMENT LEARNING

In this section, we want to introduce a concept how agents trained with RL can be used for production scheduling decisions in a DES model. Fig. 2 shows schematically how agents for job allocation and job sequencing are interacting with the DES model. The concept is based on the assumption (but is not limited to it) that neural networks represent the agents.

From a RL perspective, we consider the allocation of entities to resources as a problem with discrete action space. Thus, the number of actions (output neurons) corresponds to the number

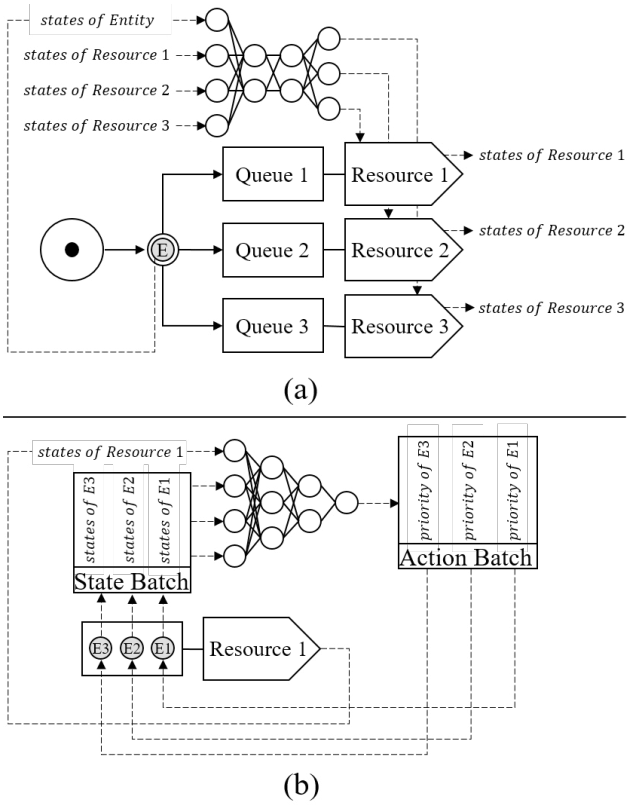

Fig. 2. Concept for integrating an RL agent with a DES model for (a) job allocation and (b) job sequencing.

of resources to which an entity can be allocated. An entity requests an allocation decision after its creation as well as every time it completes an operation and still requires to undergo further operations (code line 6 of the entity process logic in Fig. 1). The allocation of entities might be subject to several states of each resource (states of Resource 1, ..., 3) for instance, the current setup type of each resource or the total unprocessed workload allocated to each resource. Furthermore, the agent might also incorporate several attributes of the entity to be allocated (states of Entity), such as the required processing times of each resource, the required setup type to perform the next operation and the due date of the entity. The agent allocates the entity to the resource whose associated output neuron perceives the greatest activation.

For the sequencing of entities, however, we propose the training of an agent with only a single output neuron. By this means, we consider the sequencing of jobs as a problem with continuous action space. A machine invokes a sequencing event, whenever it completes a job and depending on whether there are any jobs in the preceding buffer (code line 4 of the resource process logic). In order to determine the next entity for the requesting resource, the agent generates a batch of entity states (states of E1, …, 3) to compute a corresponding batch of priorities (priorities of E1, …, 3). For calculating the priorities, the agent may additionally consider several states of the requesting resource (states of Resource 1), such as the current setup type of the resource, for each set of entity states within the batch. Finally, the resource receives the entity with the highest priority.

4. IMPLEMENTING THE OPENAI GYM INTERFACE FOR A DISCRETE-EVENT SIMULATION MODEL

In order to incorporate agents for allocation and / or sequencing decisions, the DES model requires an additional interface for exchanging information with the agents. For this purpose, we implement the unified interface of OpenAI Gym for the DES model. OpenAI Gym is an open-source library for RL research, which includes a collection of environments with a common interface. These environments can be used for developing and comparing DRL algorithms. In addition, developers can use the Gym interface to create own environments. Many libraries with pre-built DRL algorithms, in particular OpenAI Baselines and Stable Baselines, work only with environments that provide the Gym interface. Fig. 3 shows important methods of the Gym interface as well as how agents and Gym environments are integrated. The reset method is called at the beginning of each episode (simulation run), while the step method controls the execution of the simulation model.

The aim is to provide the DES model with these methods to create a unified environment for RL applications. The implementation is simple, if the DES model is based on a python library like Salabim. In this case, the developer only needs to define the OpenAI Gym class as parent class of the DES model, in order that the model inherits the functionalities of the Gym interface. Furthermore, the developer needs to create a custom step method and reset method. Fig. 4 on the next page gives a suggestion for the design of the step

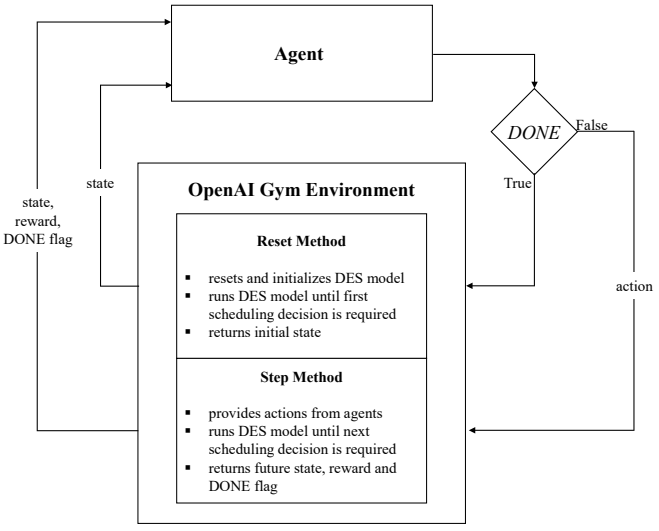

Fig. 3. Integration of an agents with the Gym environment.

and reset method for production scheduling problems. The presented step method allows the incorporation of agents for both, sequencing and allocation decisions. Furthermore, on the example of using an agent for sequencing decisions, Fig. 4 shows how the step and reset method are connected to the process logic of the DES model. The integration of both methods with the DES model for allocation decisions works in the same way. The difference is that the step and reset method are connected with the process logics of entities instead. Here, the access points to the DES model are all events, where a resource requests the next entity for processing (code line 4 of the resource process logic).

5. CONCLUSION AND OUTLOOK

In this paper, we presented a method that guides the modeling of production scheduling problems as RL environments. Our method involves the application of DES and the OpenAI Gym interface. DES allows us to model the underlying processes and dynamics of any scheduling problem. The Gym interface ensures that the development of models pursues a uniform standard, which simplifies the deployment of pre-implemented RL algorithms. Our method allows us to deploy algorithms and agents for both problems, the allocation of jobs to resources and the sequencing of jobs for a resource. Furthermore, the proposed step method allows either to train only a single agent for one of the two problems or several agents simultaneously for both problems.

Although the example we used to explain our method underlies the assumption that the environment is modeled with a python-based DES library, the method probably requires only minor specific adjustments to be applicable to any other DES tool. However, this hypothesis must be evaluated in the course of further research. In the future, we want to apply our method on different DES tools. In particular, we want to investigate whether and how the method needs to be extended to also allow an easy development of RL environments with non-python based DES tools.

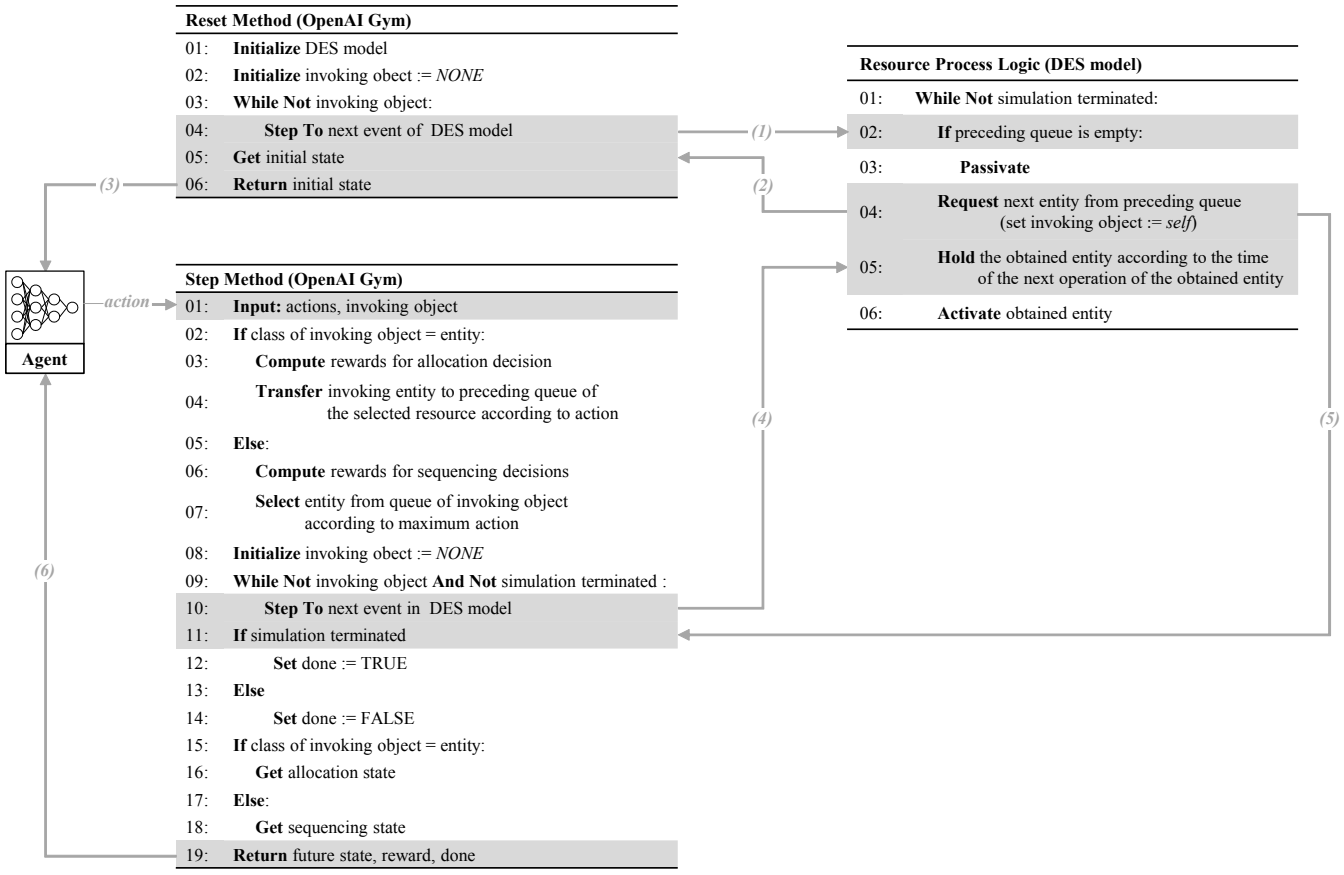

Fig. 4. Integration of the OpenAI Gym Interface with the DES model of Fig. 1 on the example of the sequencing of jobs.

REFERENCES

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine*, *34*, pp. 26–38.

Banks, J., Carson II, J. S., Nelson, B. L., and Nicol, D. M. (2010). *Discrete-Event System Simulation*. (5th ed., International version). Upper Saddle River, N.J., London: Pearson Education.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., and Tang, J., et al. (2016). *OpenAI Gym*.

Dahl, O.-J., and Nygaard, K. (1966). SIMULA: An ALGOL-based Simulation Language. *Communications of the ACM*, *9*, pp. 671–678.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., and Radford, A., et al. (2017). *GitHub repository*: GitHub. https://github.com/openai/baselines. Accessed 22.11.2020.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., and Pineau, J. (2018). An Introduction to Deep Reinforcement Learning. *Foundations and Trends in Machine Learning*, *11*, pp. 219–354.

Gershwin, S. B. (2018). The Future of Manufacturing Systems Engineering. *International Journal of Production Research*, *56*, pp. 224–237.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., and Traore, R., et al. (2018). *GitHub repository*: GitHub.

Lang, S., Lanzerath, N., Reggelin, T., Müller, M., and Behrendt, F. (2020a). Integration of Deep Reinforcement Learning and Discrete-Event Simulation for Real-Time Scheduling of a Flexible Job-Shop Production. In K.-H. G. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, & T. Roeder, et al. (Eds.), *Proceedings of the 2020 Winter Simulation Conference (WSC'20)*. Piscataway, NJ, USA: IEEE.

Lang, S., Reggelin, T., Behrendt, F., and Nahhas, A. (2020b). Evolving Neural Networks to Solve a Two-Stage Hybrid Flow Shop Scheduling Problem with Family Setup Times. In *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS 2020)* (pp. 1298–1307).

Lang, S., Schenk, M., and Reggelin, T. (2019). Towards Learning- and Knowledge-Based Methods of Artificial Intelligence for Short-Term Operative Planning Tasks in Production and Logistics: Research Idea and Framework. *IFAC-PapersOnLine*, *52*, pp. 2716–2721.

Lin, C.-C., Deng, D.-J., Chih, Y.-L., and Chiu, H.-T. (2019). Smart Manufacturing Scheduling With Edge Computing Using Multiclass Deep Q Network. *IEEE Transactions on Industrial Informatics*, *15*, pp. 4276–4284.

Luo, S. (2020). Dynamic Scheduling for Flexible Job Shop with New Job Insertions by Deep Reinforcement Learning. *Applied Soft Computing*, *91*, article-nr. 106208.

Matloff, N. (2008). Introduction to Discrete-Event Simulation and the SimPy Language. http://heather.cs.ucdavis.edu/~matloff/156/PLN/DESimIntro.pdf. Accessed 24.11.2020.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., and Wierstra, D., et al. (2013). *Playing Atari with Deep Reinforcement Learning.* Technical Report, NIPS Deep Learning Workshop 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-Level Control through Deep Reinforcement Learning. *Nature*, *518*, pp. 529–533.

Scholz-Reiter, B., Beer, C. de, Freitag, M., Hamann, T., Rekersbrink, H., and Tervo, J. T. (2008). Dynamik logistischer Systeme. In P. Nyhuis (Ed.), *Beiträge zu einer Theorie der Logistik* (pp. 109–138). Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

Schriber, T. J., Brunner, D. T., and Smith, J. S. (2017). Inside discrete-event simulation software: How it works and why it matters. In W. K. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, & E. H. Page (Eds.), *Proceedings of the 2017 Winter Simulation Conference (WSC'17)* (pp. 735–749). Piscataway, NJ, USA: IEEE.

Shiue, Y.-R., Lee, K.-C., and Su, C.-T. (2018). Real-Time Scheduling for a Smart Factory using a Reinforcement Learning Approach. *Computers & Industrial Engineering*, *125*, pp. 604–614.

Stricker, N., Kuhnle, A., Sturm, R., and Friess, S. (2018). Reinforcement Learning for Adaptive Order Dispatching in the Semiconductor Industry. *CIRP Annals*, *67*, pp. 511–514.

Sutton, R. S., and Barto, A. (2018). *Reinforcement learning: An introduction.* (2th ed.). Cambridge, MA, London: The MIT Press.

van der Ham, R. (2018). Salabim: Discrete Event Simulation and Animation in Python. *Journal of Open Source Software*, *3*, pp. 767–768.

Wang, H., Sarker, B. R., Li, J., and Li, J. (2020). Adaptive scheduling for assembly job shop with uncertain assembly times based on dual Q-learning. *International Journal of Production Research*, pp. 1–17.

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*, pp. 1264–1269.