Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

SIMULATION-BASED DEEP REINFORCEMENT LEARNING FOR MODULAR PRODUCTION SYSTEMS

Niclas Feldkamp
Soeren Bergmann
Steffen Strassburger

Information Technology in Production and Logistics
Technische Universität Ilmenau
P.O. Box 100 565
Ilmenau, 98684, GERMANY

ABSTRACT

Modular production systems aim to supersede the traditional line production in the automobile industry. The idea here is that highly customized products can move dynamically and autonomously through a system of flexible workstations without fixed production cycles. This approach has challenging demands regarding planning and organization of such systems. Since each product can define its way through the system freely and individually, implementing rules and heuristics that leverage the flexibility in the system in order to increase performance can be difficult in this dynamic environment. Transport tasks are usually carried out by automated guided vehicles (AGVs). Therefore, integration of AI-based control logics offer a promising alternative to manually implemented decision rules for operating the AGVs. This paper presents an approach for using reinforcement learning (RL) in combination with simulation in order to control AGVs in modular production systems. We present a case study and compare our approach to heuristic rules.

1 INTRODUCTION

Industrial production finds itself in a trade-off between productivity and flexibility. In the automotive industry, assembly lines are traditionally optimized for productivity. This has been the case since the establishment of the very first assembly line for the mass production of automobiles by Henry Ford. Nowadays, the importance of flexibility becomes increasingly important because of the increasing demands on quality, variety, and customizability (ElMaraghy et al. 2013). Production companies struggle to find solutions that guarantee the reliability of supply as well as stable delivery times. Many car companies therefore started pilot projects to investigate the modernization of the traditional assembly line concept towards a more flexible, job shop alike production scheme. Those attempts can be summarized under the term *modular production system* (Grunert et al. 2019; Hüttemann et al. 2016; Hüttemann et al. 2019) In practice, many other terms that all describe the same approach can be found, including “modular assembly” (Audi AG 2019), “Flexi-Line” (Mayer 2018), “Full-Flex-Plant” (Daimler AG 2018b) or “Factory 56” (Daimler AG 2018a). The goal of these approaches is to produce nearly any vehicle type including their variants in the same plant, so that even the introduction of new products does not require a lengthy modification or reconstruction of the production facility. At best, new products can even be introduced during operation without pausing the current production. This shall be achieved by the implementation of a self-regulating production in the proper sense of the Industry 4.0 approach. Production jobs are transported through the system by automated guided vehicles (AGVs) through a system of matrix-style arranged stations. Those stations are differently equipped regarding the tools and production steps that they can offer. Products can decide dynamically and individually on the production

step they want to have executed next and the station that shall perform it. This entire control strategy is only limited by technical restrictions of the production program regarding the order of some of the production steps. Implementing predefined heuristics for the decision-making of the AGVs is the most straight forward method. However, incorporating the dynamically changing environment and system status in those heuristics can be quite difficult in a complex system and may not lead to a good decision in every possible scenario. Furthermore, using ex ante optimized production plans is not sufficient as well because of the highly dynamic nature of self-regulating modular production systems. Therefore, using an artificial intelligence (AI)-based control logic that is trained on a large number of samples and that can then decide in real time yields a promising approach. On the other hand, for modelling, planning, and controlling production systems, especially in the automotive industry, discrete-event simulation is a well-established methodology. Therefore, training an AI-based control logic using a proper simulation model is standing to reason. In this paper, we present an approach for using Deep Reinforcement Learning (RL). The RL-algorithm can train using a simulation model of a modular production system. The remainder of this paper is structured as follows: In section 2, we introduce related work regarding modular production systems and deep reinforcement learning, complemented by a short review of work regarding reinforcement learning in combination with simulation. In section 3, we introduce our general approach and describe our preliminary tests and experiments, followed by the description of the actual case study implementation and its results. In section 4, we give some concluding remarks and a discussion of possible future work.

2 RELATED WORK

2.1 Modular Production Systems

The term modular production system is synonymous for a number of terminologies that all describe the same principle. In this approach, flexibilization of the automotive assembly is the main goal. The traditional principle of a fixed assembly line is dissolved in favor of a more job-shop-alike production scheme (Grunert et al. 2019; Hüttemann et al. 2016). The actual assembly of products is conducted on assembly stations (often dubbed “assembly isles”), that are very flexible and can execute various tasks, depending on their function, tools equipped, and personnel. On the other hand, each product is defined by a certain set of tasks, so products can feature a very high degree of individualization. Individual products thus need to pursue different stations and can therefore pursue individual routes through the system. The assembly stations are not linked by a fixed conveying system. The transport of products as well as the transport of material is carried out by AGVs (Kern et al. 2017). Flexibility arises therefore on multiple levels: Firstly, the assembly steps to be executed can differ greatly and are not bound to a global uniform system cycle time, as shown in Figure 1.

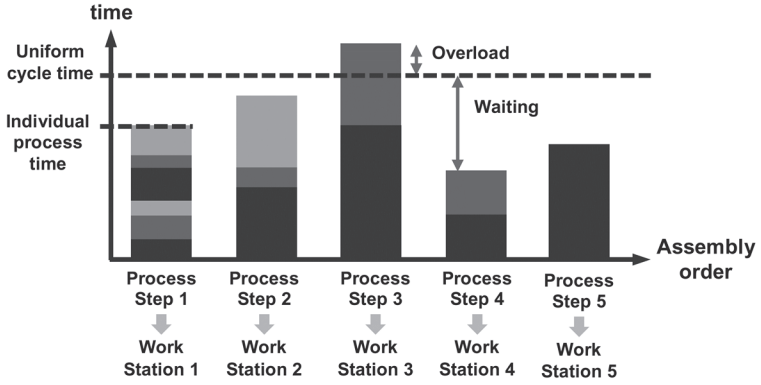

Figure 1: Individual process times vs. uniform cycle time (Greschke et al. 2014).

Secondly, the available stations offer different tasks that they can perform. And lastly, as a consequence of the removal of a fixed conveyor system, products can move freely and therefore assembly steps do not necessarily have to be executed in a fixed order, unless technical restrictions apply. This results in a high degree of freedom for products to move autonomously through the system and having their assembly steps executed, leading to more flexibility and improved balancing of cycle times (Greschke et al. 2014; Hüttemann et al. 2016).

On the other hand, this high degree of freedom comes at the price of an increased complexity and challenges regarding coordination, communication, and organization between system components. This refers to one of the main visions of the Industry 4.0 concept that proclaims an interconnection between machines, devices, sensors, and humans (Kagermann et al. 2013). A few publications already addressed the simulation related aspects and modelling techniques of modular production systems (Feldkamp et al. 2019; Schönemann et al. 2015) and others concluded that a proper simulation model is essential to develop heuristics for optimizing planning and scheduling problems in those systems, which still is an ongoing challenge (Filz et al. 2019; Mueller et al. 2018). Therefore, using AI-assisted control logics for guiding products to find their way through the system contributes to solving those challenges.

2.2 Deep Reinforcement Learning

The basic idea behind reinforcement learning is that an agent can interact with its environment. The agent can choose to perform an action from a given set of certain actions based on the state of the environment that it can perceive. As the consequence of its action, the agent gets a feedback from its environment in return. This feedback is granted as a reward. The reward is calculated by the environment and can be positive or negative (punishment instead of reward), dependent on the given state of the environment and the action that the agent chose to perform (Sutton and Barto 2018). This relationship can be formalized as a so called Markov decision process (MDP), which is shown in Figure 2. Based on the perception of the environmental state S_t from all possible states S at discrete points in time $t = 1..n$, the agent chooses an action A_t from the set of possible actions A. This results in the change of the state to S_{t+1}.The agent then gets the new state as well as the reward R_{t+1}.

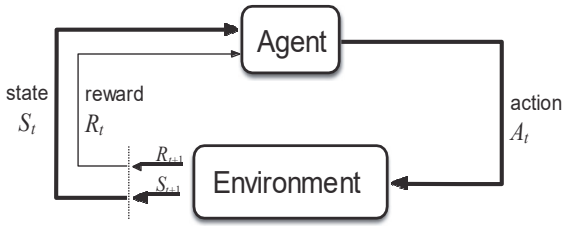

Figure 2: Agent-Environment-Relationship in a Markov decision process (Sutton and Barto 2018).

The agent aims at choosing its actions in such a way that it can maximize the sum of its rewards, thereby ultimately developing a policy for solving problems (Sutton and Barto 2018; Wiering and van Otterlo 2012). In fact, reinforcement learning is an umbrella term for a variety of different methods and algorithms. The majority of those algorithms can be divided into two categories: Model-based and model-free approaches. Model-based approaches aim to learn a model of underlying rules and dynamics of their regarding environment in order to derive an optimal policy. Model-free approaches on the other hand aim to find the best action for every state in order to build a global policy over all states, or to reach a desired state as quickly as possible (Achiam and Morales 2018; Kaelbling et al. 1996). For complex and dynamic systems that may face stochastical uncertainty (like modular production systems do), model-based approaches are usually not very suited because the state space is too large, the internal model building of the algorithm might be too inaccurate, and computational effort is not feasible (Gosavi 2015a). A popular example for model-free approaches is *Q-Learning*. Here, the agent learns a value for every state-action-

pair (so called *Q-Value*) so that it can approximate a value function $Q(s,a)$ from which it can derive an optimal policy (Watkins and Dayan 1992). An adequate extension of the traditional Q-Learning is Deep-Q-Learning (DQL), where the relationship between states and actions is mapped by a deep convolutional neural network. This enables the principle of Q-Learning even for large state and action spaces (Mnih et al. 2015; Shrestha and Mahmood 2019).

2.3 Combining DQL and Discrete-Event Simulation for Solving Optimization Problems

Gabel and Riedmiller could show that DQL is able to solve textbook optimization problems in the context of production management. In an job-shop scheduling scenario with parallel machines they were able to beat classical heuristic rules (Gabel and Riedmiller 2008). There is only little research available regarding the combination of DQL and discrete-event simulation models, which is the preferred typed of simulation for modelling modular production systems (Feldkamp et al. 2019), leaving a large gap for further research. Gosavi indicates the complex challenge for synchronization of simulation progress and learning algorithm in this context (Gosavi 2015a). For solving simple optimization problems, he successfully applies DQL algorithms onto a simulation model (Gosavi 2015a, 2015b). This work relates to finding a global optimal solution to fixed problem. On the other hand, DQL in combination with simulation yields also very promising possibilities regarding the dynamic and on-demand decision making in real-time in a dynamically changing and uncertain environment. Zhang et al. successfully use DQL and simulation for determining the next job to be processed on a machine (Xie et al. 2019; Zhang et al. 2017) as well as for real-time batching (Zhang et al. 2018). In this context, Xie et al. point out that there are many challenges in combining simulation and DQL that remain not conclusively solved, for example the quantification of states and actions, the design of the reward function, and ensuring the convergence of the learning algorithm (Xie et al. 2019).

3 USING SIMULATION AND REINFORCEMENT LEARNING FOR CONTROLLING AGVS IN MODULAR PRODUCTION SYSTEMS

3.1 Concept for an Reinforcement-based Control Systems

Hereinafter, we assume, that an AGV picks up a product/job at the source and releases it after all necessary assembly tasks have been executed before picking up a new one. Therefore, product and AGVs form a unified whole leaving all the decision making in the process to the AGV and its control logic. The decision-making-process for AGVs can then be divided into multiple levels as shown in Figure 3.

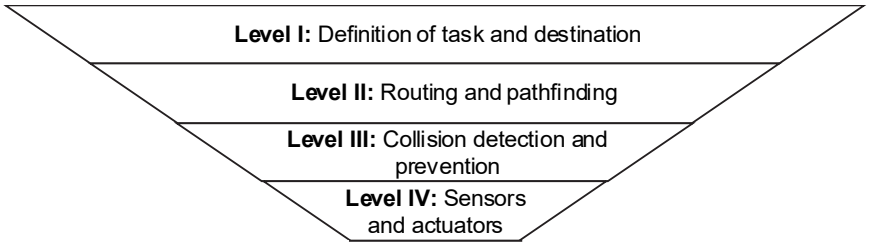

Figure 3: Hierarchy of decision-making.

The first level of decision-making is responsible for selecting the next task (which step to do next) and destination (which station to choose for performing the task). This decision cascades into subsidiary decisions regarding routing and pathfinding, collision detection and prevention down to the actual sensors and actuators of the AGV. Our approach is targeted at the first level of decision making, leaving lower tier levels to separate systems or the simulator, respectively. However, this does not prevent those levels to be controlled by AI-supported systems as well. In our approach, the AGV needs to make a decision

after every finished task or after picking up a new job at the source. When a decision is pending, the simulation is stopped and the current system state is sent to the RL-agent. The agent then makes the decision that determines the next station to drive to and sends it back to the simulation, setting the next target for the AGV and continuing the simulation until a new decision is needed. In our approach, we use DQL for the learning of the RL-agent, as explained above. This means, we use a multilayer neural network to model the value function between system states and actions as shown in Figure 4. The input of the network is represented by the current system state. A large number of possible system parameters could theoretically be passed. In our first approach, we reduced the input to the tasks that are currently possible to perform, the stations that are available (meaning that they are properly equipped to perform any of the available tasks), and the current length of each queue. The output layer then represents the stations, so that the output neuron with the highest value determines the next station for the current decision.

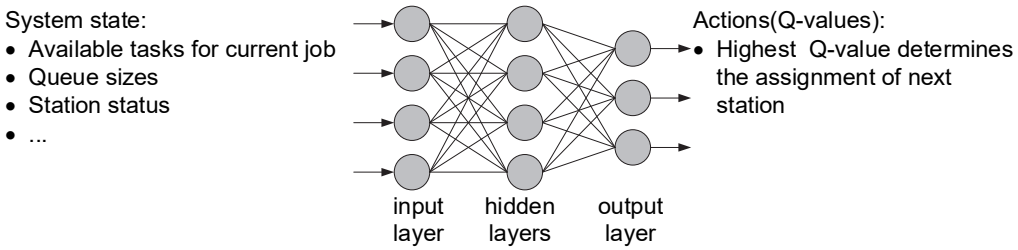

Figure 4: Schematic construction of a potential DQL setup for training AGVs.

In order to further increase the performance of the learning algorithm, we use Double Deep Q-Learning (DDQN). Here, two identical neural networks are used: One local network that predicts the next action, and a target network that computes the Q-value for the training step. This method prevents the agent to overestimate suboptimal policies and leads to a better overall learning performance (van Hasselt et al. 2016). The learning process is outlined in Figure 5.

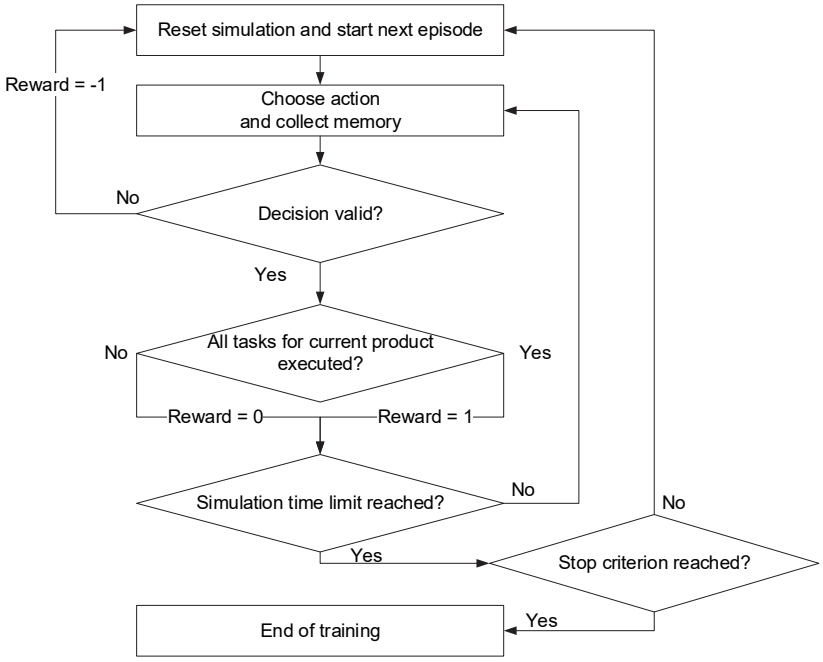

Figure 5: Learning process for solving assembly jobs.

The agent learns over the course of multiple episodes. One episode ends if the agent takes an invalid decision or the simulation time limit is reached. The actual training process that we developed resembles a popular learning problem where an agent tries to find the exit of a maze without falling into a trap door (Atienza 2020). For each correct decision that the agent makes, it gets a reward of 0. If a product is completely assembled and released, a reward of 1 is given (exit of the maze is considered being found), which encourages the agent to get to the exit as fast as possible. If an invalid action is chosen (which means that the chosen station cannot perform any of the currently available tasks), a reward of -1 is given, the episode ends and the simulation starts from the beginning. This ensured the learning of the correct assembly rules and restrictions, so that we could add additional performance-related components to the reward function after a stable learning process could be established. Updating the neural networks weights and therefore the Q-function directly after each decision is not advisable though. Temporal correlations between learning from experience and gaining experience can cause undesirable bias, letting the agent tends forget older occurrences of certain state/action pairs. Therefore, we implemented an memory replay mechanism, that stored each experience that the agent gains in a large buffer. Each memory represents one experience that the agent has made, which in turn represents the system status, the agents actions, the new system status after the action was taken and the gained reward. After each episode, a random sample of memories is taken from this buffer in order to train the neural network (Kalyanakrishnan and Stone 2007).

3.2 Case Study

3.2.1 Simulation Model and Implementation

For a case study, we implemented an example model that represents a typical layout for automotive modular production systems. The model was implemented in Siemens Plant Simulation, which is frequently used in the German automotive industry. Figure 6 shows a screenshot of the model. The model contains a 50m x 35m factory floor that hosts 9 assembly stations. The number of AGVs can be set variably, but in the final experiments we set the number to 12. AGVs can queue up before the stations when waiting for processing. There are 4 different product types, each containing up to 20 individual assembly tasks with distinct processing times. Each of the stations offers a portfolio between 2 and 4 tasks that they can perform. Each product has a specific underlying assembly priority chart, so some tasks have to be done in a certain order, and for other tasks the order can be chosen freely. The type of job is chosen randomly before being assigned to a vacant AGV that picks it up at the source.

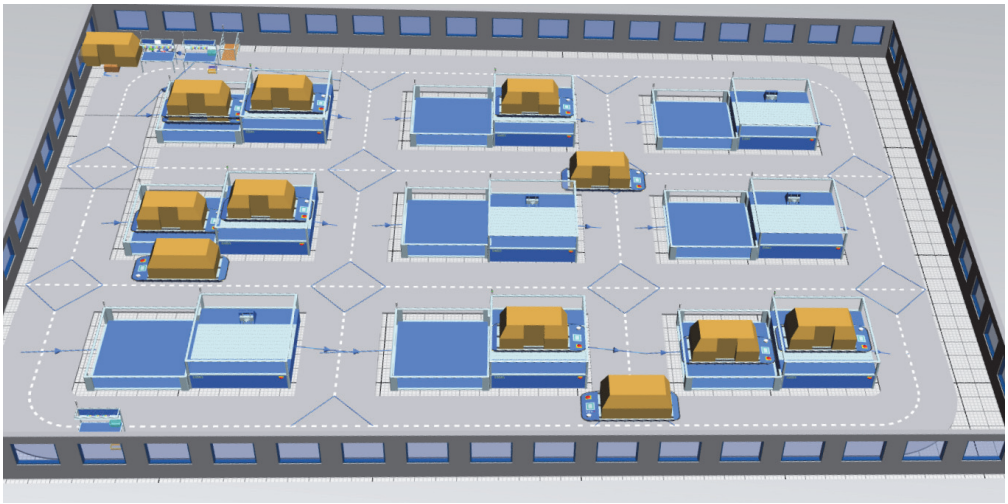

Figure 6: Screenshot of the Plant Simulation model.

The implementation of the DQL-algorithm was done in Python's Tensorflow/Keras library. The connection between Plant Simulation and Python was established via a TCP/IP-based interface. In this setup, the python side acts as a server. When starting the server, the required elements including the neural network are initialized. Then, the server goes into a loop where it listens to incoming requests from the simulator. At each request, a string of data is transmitted including the state of the system as well as some metadata like simulation time and performance values. The state vector is send to the RL-algorithm, receiving an action that is then sent back to the simulator as response to TCP/IP-request. The server side is also responsible for calculating rewards as well as stopping, resetting, and starting the simulation via keywords embedded in the TCP/IP-response.

3.2.2 Preliminary Test

We performed a large number of preliminary tests before executing the actual case study in order to fine-tune the setup and enabling the agent to learn reasonable policies. One of the biggest challenges was the management of the agent memories in combination with the progression of learning episodes. When learning, RL-agents are in a dilemma between exploration and exploitation. Exploration means, that the agent tries out new actions while exploitation means that the agent is sticking to policies that it already learned. To ensure exploration and preventing the agent to be stuck in suboptimal policies, decisions should be chosen randomly with a certain probability Ɛ. This probability should be high at first (usually 90%) and then decrease over the span of episodes.

With our developed concept (Figure 5) of resetting the simulation after the agent making an invalid decision (according to the assembly priority chart), the agent progressed too fast through the episodes while collecting only a few memories per episode. This was especially the case at the beginning of the learning process, when the value of Ɛ is high and therefore the agent does a high amount of exploration, causing him to choose a lot of invalid decisions. We therefore increased the number of invalid decisions that the agent can make in an episode to 10, before closing the episode and resetting the simulation. This resulted in a greatly improved learning progression. To further improve the collection of memories in the high-exploration phase, we implemented a minimum amount of memories that needs to be collected before starting a new episode, so that an episode may include multiple simulation runs, depending on the time of reset, until the required amount of memories was achieved. This method resulted in a much more stable and linear acquirement of memories as shown in Figure 7. Note that the maximum number of memories is capped at 100,000, due to the implementation of a sliding window that overwrites the oldest (and probably less valuable) memories once the limit is hit.

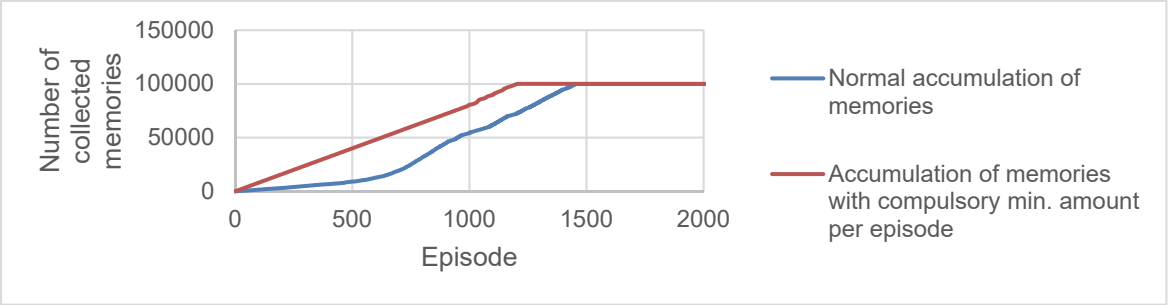

Figure 7: Comparison of memory collection methods for the experience replay.

Furthermore, we implemented a prioritized replay mechanism, transforming the simple table of memories into a tree structure, so that valuable memories can be sampled more frequently. The value of a memory or experience, respectively, is based on the magnitude of the temporal-difference error (TDE) during the training of the neural net. A high TDE means that the actual reward received was far from the agents prediction of the reward it would get when choosing an action. In other words, the more surprised

the agent is from the outcome of an action, the higher its learning progression is (Schaul et al. 2016).We found that prioritizing experiences in our scenario resulted in a drastically improved learning progression.

Regarding the layout of the neural network, a general rule of thumb recommends that one hidden layer with the number of neurons between the size of the input and the size of the output layer is sufficient for most problems (Zhang et al. 2018). However, after conducting numerous testing with various net sizes, we found that a hidden layer that is much larger than the input layer works best in our case. In our final version we used 3 hidden layers with each having 256 neurons. The size of the input layer was 29 as shown exemplarily in Figure 8, and the size of the output layer was 9, representing the stations as possible actions in the system.

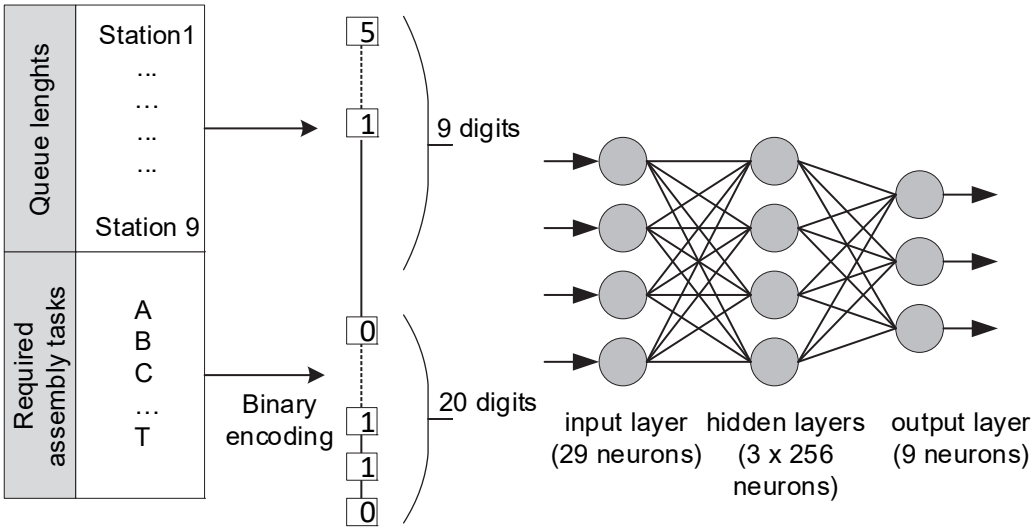

Figure 8: Construction of the input layer with example values.

After constructing a setup that ensured a stable learning of the rules of the underlying assembly tasks, so that the agent was able to learn how to successfully finish assembly jobs from source to sink, we added a performance related component to the reward function based on the lead time that the agent needed to complete the job. With this approach, the agent could actually make use of the information regarding queue lengths it is given, so that it can manifest a relation between queue length and lead time in its Q-function. After various testing we found that the best solution for this is to give a bonus on top of the finishing reward based on the deviation of the archived lead time from a fixed average lead time that was measured using only random decision making. With this method, lead times that are better than the average are rewarded, whereas lead times above the average are punished.

3.2.3 Results

In our final setup, we used a maximum simulation run time of 2 days before the next episode was started, and a batch size of 64 samples for the experienced replay training. The total time needed was roughly 40 hours until convergence of the algorithm after approximately 2200 episodes. As always when using deep learning based algorithms, over and under fitting needs to be avoided. However, the convergence of RL-algorithms is typically not easy to determine, since agents keep making bad decision caused by the exploration factor. Regarding the stop criterion, we assumed that the agent has learned sufficiently when it manages to finish an episode ten times in a row without hitting the error limit, not counting errors caused by random exploration. Figure 9 shows the average reward over the episodes.

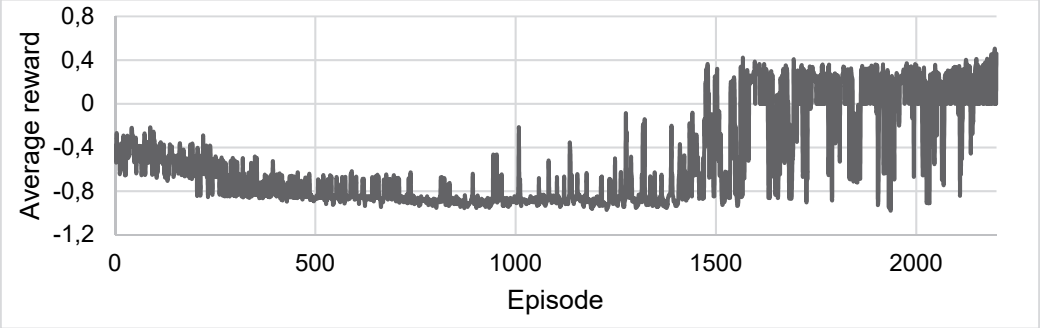

Figure 9: Learning curve of the RL-agent.

Note that this shows the average reward divided by all decisions made. Having a correct decision mostly results in a direct reward of 0, the average reward as shown in this form usually is smaller 1. We can clearly see how the agent improves over time, collecting only positive average rewards at the end which means that the agent makes nearly no invalid decision. For a performance evaluation of the agents policy, we compared the lead times of jobs of the trained agent against two heuristic decision rules: Following the first rule, the AGV simply picks a random valid station as next target. The second rule is picking the next valid station that has the shortest queue and is geographically the closest if multiple stations have the same queue lengths at the time of decision making. The result of this comparison is shown in Figure 10 represented by a Box-Whisker-Plot.

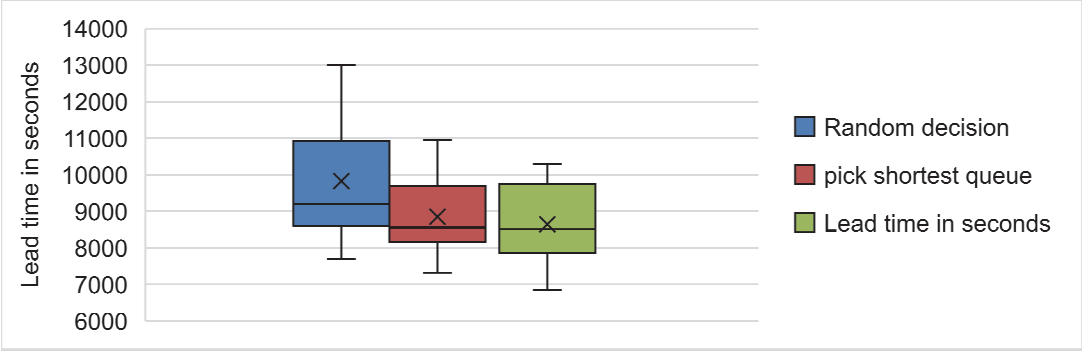

Figure 10: Comparison of decision rules.

While making a random decision obviously performs very poorly, we can see that the RL-agent's policy can keep easily up with the shortest-queue-heuristic. In detail, we can see that both policies are very similar regarding the average and median lead time (represented by the straight line and the cross mark in the boxes) with maybe even a slight edge towards the RL-agent policy. However, when comparing the datasets in total, we can see that the maximum observed lead time from the measurement following the shortest-queue-rule is much higher (around 11,000 seconds) than the maximum observed lead time following the RL-agent's policy. Furthermore, the shortest observed lead time of the RL-agent's policy beats the heuristic rule by approximately 500 seconds. This means, that the system yields potential for better lead times beyond following a simple shortest-queue heuristic, and the RL-agent, at least sometimes, found ways to do so. This can for example be the case when a station that usually exhibits a long queue has a short queue at the time of decision making. The agent would, based on its experience, select this station as its new target, even when other available stations have even shorter or no queues at this time. This prioritization of stations can only be achieved through learning and experience of the RL-agent and is difficult to embed in a static, priorly defined decision rule.

4 CONCLUSIONS AND FUTURE WORK

In this paper, we successfully used deep reinforcement learning in combination with a discrete-event simulation model of a modular production system in order to improve AGV decision making. By training an agent in an simulated environment, the agent was able to successfully learn how to finish production jobs in a sequence of 20 tasks. The obtained policy was able to beat simple heuristic decision rules. These results are very promising and show that AI-assisted control logics in a modular production system can bring a benefit, but also yield a lot of room for further research. The success of the policy depends on the complexity of the intended goal for system performance improvement.

In our case study, we only focused on the objective to improve the lead times. Multidimensional, maybe even conflicting objectives like throughput, lead times, and utilization balancing are obviously more complex to implement and are subject to further research. Not only would this increase the complexity of the reward function, but also increase the amount of information that the RL-algorithm needs to process, which in turn increases the size of the system state vector and the input layer of neural network.

Furthermore, training agents autonomously leads them to optimize their policy based on their own individual optimum only, disregarding a possible global optimal policy that may lead to better results. There are basically two ways for future research to tackle this problem: The first option would be to train an agent that controls not only one AGV, but the total system at once. This would theoretically lead to global policies, but also greatly increases the complexity of the input state vectors for the RL-algorithm. Furthermore, synchronization of the decision request and decision making between simulation and RL-agent would be more difficult. The other solution would be to implement elements of traditional multi-agent systems, so that agents can interact and negotiate with other agents. These approaches can be summarized as multi-agent deep reinforcement learning (MADRL), and have been applied in fields like swarm systems, fleet and traffic management, and energy sharing optimization (Nguyen et al. 2020). Therefore, future work could investigate possibilities for applying MADRL-algorithms to the problem of AGV decision making in modular production systems.

REFERENCES

Achiam, J., and M. Morales. 2018. OpenAI Spinning Up – Part 2: Kinds of RL Algorithms. https://spinningup.openai.com/en/latest/etc/author.html, accessed 14th April 2020.

Atienza, R. 2020. *Advanced Deep Learning with TensorFlow 2 and Keras.* 2nd ed. Birmingham, UK: Packt Publishing.

Audi AG. 2019. Die Modulare Montage - Fertigungsinseln statt Fließband. https://www.audi-illustrated.com/de/smart-factory/Die-Modulare-Montage, accessed 28th January 2019.

Daimler AG. 2018a. Factory 56 – Mercedes-Benz Cars increases flexibility and efficiency in operations. https://www.daimler.com/innovation/production/factory-56.html, accessed 15th April 2020.

Daimler AG. 2018b. Groundbreaking for first Full-Flex Plant. https://www.daimler.com/innovation/case/connectivity/full-flex-plant.html, accessed 28th January 2019.

ElMaraghy, H., G. Schuh, W. ElMaraghy, F. Piller, P. Schönsleben, M. Tseng, and A. Bernard. 2013. "Product variety management". *CIRP Annals* 62(2):629–652.

Feldkamp, N., S. Bergmann, and S. Strassburger. 2019. "Modelling and Simulation of Modular Production Systems". In *Simulation in Produktion und Logistik 2019,* edited by M. Putz and A. Schlegel, 391–401. Auerbach, Germany: Verlag Wissenschaftliche Scripten.

Filz, M.-A., C. Herrmann, and S. Thiede. 2019. "Simulation-Based Data Analysis to Support the Planning of Flexible Manufacturing Systems". In *Simulation in Produktion und Logistik 2019,* edited by M. Putz and A. Schlegel, 413–422. Auerbach, Germany: Verlag Wissenschaftliche Scripten.

Gabel, T., and M. Riedmiller. 2008. "Adaptive Reactive Job-Shop Scheduling with Learning Agents". *International Journal of Information Technology and Intelligent Computing* 24(4).

Gosavi, A. 2015a. *Simulation-Based Optimization. Parametric Optimization Techniques and Reinforcement Learning*. New York, New York: Springer.

Gosavi, A. 2015b. "Solving Markov Decision Processes via Simulation". In *Handbook of Simulation Optimization,* edited by M. C. Fu, 341–379. New York, New York: Springer.

Greschke, P., M. Schönemann, S. Thiede, and C. Herrmann. 2014. "Matrix Structures for High Volumes and Flexibility in Production Systems". *Procedia CIRP* 17:160–165.

Grunert, D., S. Jung, T. Leipold, A. Göppert, G. Hüttemann, and R. Schmitt. 2019. "Service-oriented Information Model for the Model-Driven Control of Dynamically Interconnected Assembly Systems". In *Tagungsband des 4. Kongresses Montage Handhabung Industrieroboter,* edited by T. Schüppstuhl, 23–33. Berlin, Heidelberg, Germany: Springer Berlin Heidelberg.

Hüttemann, G., A. F. Buckhorst, and R. H. Schmitt. 2019. "Modelling and Assessing Line-less Mobile Assembly Systems". *Procedia CIRP* 81:724–729.

Hüttemann, G., C. Gaffry, and R. H. Schmitt. 2016. "Adaptation of Reconfigurable Manufacturing Systems for Industrial Assembly – Review of Flexibility Paradigms, Concepts, and Outlook". *Procedia CIRP* 52:112–117.

Kaelbling, L. P., M. L. Littman, and A. W. Moore. 1996. "Reinforcement Learning: A Survey". *Journal of Artificial Intelligence Research* 4(1):237-285.

Kagermann, H., J. Helbig, A. Hellinger, and W. Wahlster. 2013. "Recommendations for implementing the strategic initiative INDUSTRIE 4.0. Securing the future of German manufacturing industry". Final report of the Industrie 4.0 working group. Berlin, Frankfurt/Main: Forschungsunion, Geschäftsstelle der Plattform Industrie 4.0.

Kalyanakrishnan, S., and P. Stone. 2007. "Batch reinforcement learning in a complex domain". In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems - AAMAS '07,* edited by E. Durfee. New York, New York: ACM Press.

Kern, W., H. Lämmermann, and T. Bauernhansl. 2017. "An Integrated Logistics Concept for a Modular Assembly System". *Procedia Manufacturing* 11:957–964.

Mayer, B. 2018. Ich bin ein Fan von Effizienz. https://www.automobil-produktion.de/interviews-734/porsche-produktionsvorstand-reimold-wir-ziehen-alle-register-126.html?page=4, accessed 28th January, 2019.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. 2015. "Human-level control through deep reinforcement learning". *Nature* 518(7540):529–533.

Mueller, D., C. Mieth, and M. Henke. 2018. "Quantification of Sequencing Flexibility Based on Precedence Graphs for Autonomous Control Methods". In *Proceedings of the 4th International Conference on Industrial and Business Engineering - ICIBE' 18,* edited by S. J. Fong and T. Hajime, 211–220. New York, New York: ACM Press.

Nguyen, T. T., N. D. Nguyen, and S. Nahavandi. 2020. "Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications". *IEEE Transactions on Cybernetics* 99:1-14

Schaul, T., J. Quan, I. Antonoglou, and D. Silver. 2016. "Prioritized Experience Replay". In *International Conference on Learning Representations 2016,* May 2nd-4th, San Juan, Puerto Rico, 1-21.

Schönemann, M., C. Herrmann, P. Greschke, and S. Thiede. 2015. "Simulation of matrix-structured manufacturing systems". *Journal of Manufacturing Systems* 37(1):104–112.

Shrestha, A., and A. Mahmood. 2019. "Review of Deep Learning Algorithms and Architectures". *IEEE Access* 7:53040–53065.

Sutton, R. S., and A. Barto. 2018. *Reinforcement Learning: An Introduction.* 2nd ed. Cambridge, MA, London, England: The MIT Press.

van Hasselt, H., A. Guez, and D. Silver. 2016. "Deep Reinforcement Learning with Double Q-Learning". In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence,* 2094–2100.

Watkins, C. J. C. H., and P. Dayan. 1992. "Q-learning". *Machine Learning* 8(3-4):280-292.

Wiering, M., and M. van Otterlo. 2012. *Reinforcement Learning and Markov Decision Processes*. Heidelberg, Germany: Springer.

Xie, S., T. Zhang, and O. Rose. 2019. "Online Single Machine Scheduling Based on Simulation and Reinforcement Learning". In *Simulation in Produktion und Logistik 2019,* edited by M. Putz and A. Schlegel, 59–68. Auerbach, Germany: Verlag Wissenschaftliche Scripten.

Zhang, T., S. Xie, and O. Rose. 2017. "Real-time Job Shop Scheduling Based on Simulation and Markov Decision Processes". In *Proceedings of the 2017 Winter Simulation Conference,* edited by V. Chan, A. D'Ambrogio, G. Zacharewicz, and N. Mustafee, 3899–3907. Piscataway, New Jersey: Institute of Electrical and Electronics Engineering, Inc.

Zhang, T., S. Xie, and O. Rose. 2018. "Real-Time Batching in Job Shops based on Simulation and Reinforcement Learning". In *Proceedings of the 2018 Winter Simulation Conference,* edited by M. Rabe, A. A. Juan, N. Mustafee, and A. Skoogh, 3331–3339. Piscataway, New Jersey: Institute of Electrical and Electronics Engineering, Inc.

AUTHOR BIOGRAPHIES

NICLAS FELDKAMP is a research associate in the Group for Information Technology in Production and Logistics. He received his Ph.D in business information systems from the Ilmenau University of Technology, and holds M.Sc. from the Ilmenau University of Technology and B.Sc. from the University of Cologne. His research interests include data science, business analytics, and industrial simulations. His email address is niclas.feldkamp@tu-ilmenau.de.

SÖREN BERGMANN holds a Doctoral and Diploma degree in business information systems from the Ilmenau University of Technology. He is a member of the scientific staff of the Group for Information Technology in Production and Logistics. His research interests include generation of simulation models and automated validation of simulation models within the digital factory context. His email address is soeren.bergmann@tu-ilmenau.de.

STEFFEN STRASSBURGER is a professor at the Ilmenau University of Technology and head of the Group for Information Technology in Production and Logistics. Previously he was head of the "Virtual Development" department at the Fraunhofer Institute in Magdeburg, Germany and a researcher at the DaimlerChrysler Research Center in Ulm, Germany. He holds a Doctoral and a Diploma degree in Computer Science from the University of Magdeburg, Germany. His research interests include distributed simulation, automatic simulation model generation, and general interoperability topics within the digital factory and Industry 4.0 context. His email address is steffen.strassburger@tu-ilmenau.de.