Sequencing Through a Global Decision Instance Based on a Neural Network

Thomas Dasbach, Johannes Olbort, Felix Wenk, Reiner Ander

Sequencing through a global decision instance based on a neural network

Thomas Dasbach[1][0000-0003-3059-7576], Johannes Olbort[1], Felix Wenk[1] and Reiner Ander[l][0000-0001-7681-2118]

TU Darmstadt, DiK, Otto-Berndt-Straße 2, Germany
[1] dasbach@dik.tu-darmstadt.de

Abstract. An existing concept for sequence planning in production planning and control was extended by a global decision instance based on neural networks. Therefore, information regarding the state of the production and available orders were normalized and analyzed by one agent. In contrast to a partially observable Markow Decision Problem one single agent was allowed and used to process all available information. Feasibility and problems were examined and compared with a concept for decentralized decisions. The implementation consists of two parts, which continuously interact with each other. One part is a simulation of a job shop, including multiple machines. The other parts tackle the Markow Decision Problem with the use of double Q reinforcement learning in order to estimate the best sequence at any given time. Later, problems due to scaling and comparisons to the usage of multiple agents are given.

Keywords: Sequence Planning, Artificial Intelligence, Neural Network

1 Introduction

A continuous trend in industry is the attempt to reduce the total time to market of a new product. While both production planning and design try to support that goal, their impact relies heavily on each other. The decisions made in the design phase have a huge impact on the reduction potentials in the production planning. Therefore, it is necessary to give the design departments via PLM tools ways to estimate the impact of decisions in later stages of the product life cycle. The first step in creating such tools is to automate decisions that are currently done manually within the production planning.

At the same time, production systems are evolving into Cyber-Physical Systems that are in constant exchange with central servers and with each other. Large amounts of data are available that provide information about the status of resources, such as production machines or logistic vehicles, as well as about the progress of orders within the production systems.

New technologies and application scenarios also create new challenges. One challenge that can only be solved by the complete networking of production systems and products is the dynamic creation of sequence planning based on the current situation. In manufacturing companies, this is still done manually and by using simple priority

rules [1]. One approach to solve this problem is using multi-agent systems. Deep neural networks are used, which receive information about the state of the production and can then influence the sequence. These networks are called agents. A distinction can be made between approaches in which a single agent coordinates the entire production and approaches in which several agents are used for different production sections.

A further distinction must be made between the optimization target. Sequencing can be optimized either for the shortest possible production time or for maximum delivery reliability. Existing research often refers to the shortest possible production time, which means that there is a wide range of approaches. In the case of the most exact delivery reliability possible, which in the context of a just-in-time industry has to take a higher priority, the amount of research is smaller. This is due to the fact since no absolute mathematical models are available in this case. In this paper, we consider sequencing as a Markow Decision Problem. To solve this problem, neural networks are used to insert a global decision instance into the process. The goal is to achieve the highest possible delivery reliability rather than the lowest possible lead-time.

This paper will therefore start with a brief introduction into sequence planning and machine learning. Both technologies are the foundation of the later concept and implementation. After this literature review, the concept is presented. Fundamentally, the concept is separated in two distinct parts. Both parts interact with each other, as the results of either are the input of the other. Right after the concept an overview of the implementation is given. The technologies used within the prototype are shown, so that the reader can develop a more detailed understanding of the approach. In the end a summary concludes the findings and the current limitations of the approach.

2 Literature

2.1 Sequence planning

Sequencing is concerned with finding an optimal arrangement for a given number of orders and a given number of machines. But pptimality is subject to the selected goal. In the theoretical consideration of sequence planning, three assumptions are usually made [2]:

- A machine can only process one order at a time
  - Consequently, a single order can contain more than one product. In this case, all products must always be treated as a single entity from a sequencing perspective.
- After processing an order, a machine is immediately available for a new order.
  - Set-up times and maintenance work must be recorded in the processing time of an order in each processing step.
  - Planned maintenance work can be included in a schedule by a separate order.
- An order can never be processed by two machines at the same time
  - Machines working in cooperation with each other are represented by a single machine within the simulation.

In almost all problem definitions, several orders are accepted. However, the problems can be differentiated by the number of machines included. If there is only one machine in the planning, it is called a single machine model. This solution of a single machine problem can be solved mathematically and is therefore widespread. In industry, this problem only occurs on machines that represent a bottleneck in production.

More complex are models with multiple independent machines. In such a problem, each job must be processed by a different number of machines in any order. Problems of this type are NP-hard and thus no longer mathematically solvable as complexity increases. Here, different heuristics are used to create approximations.

A further distinction is made between flow shops and job shops. In flow shops, each job must be processed once on each machine. The processing sequence on the machines is the same for all orders and there is a storage area with infinite capacity in front of each machine, so that any number of orders can wait there to be processed. The difference in a job shop is that the order of processing is defined individually for each order and not a processing sequence is fixed for all orders.

Whereas in the past the utilization of operating resources was the most important optimization criteria in production planning and control, today delivery reliability and a short delivery time are the decisive target variables [3]. In practice, sequence rules, also called priority rules, are still the most common method for determining the processing sequence of orders:

- First In - First Out
- Earliest plan start date
- Earliest plan end date
- Least remaining slip
- Earliest Due Date
- Shortest operation time

More detailed information on these rules is provided by HERMANN and SWAMIDASS. [4, 5]

2.2 Reinforcement Learning

Reinforcement learning is the most general form of machine learning. There is no historical data available at the beginning of the problem to which the methods of supervised or unsupervised learning could be applied. Thus, part of the task is to develop a basic strategy. One form of implementing reinforcement learning is a Markow Decision Problem. Here, a machine learning based agent interacts with the environment [10].

The interaction possibilities are defined by the influence possibilities provided by the environment, also called action space. The agent regularly receives information from the environment about the current situation, the observation space and possibly a reward for his actions. These can be positive or negative. The goal of the agent during a run is to maximize the achieved reward. Over the course of many simulations, called episodes, the different situations, the possible actions and the rewards are put into context so that situations are valued accordingly.

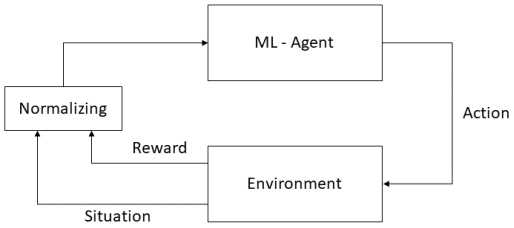

Fig. 1. Setup for reinforcement Learning

Within a Markow Decision Problem all Situation an agent encounters are stored within a graph. The different possible actions an agent can perform in the situations lead to transitions through the graph. Specific actions within a situation deterministically lead to a transition into a specific other situation, which means that situations can be put into context to each other. Since the overall goal of the agent is to achieve situations in which high rewards are obtained or low rewards are avoided, different situations can be valued differently. Even if they do not explicitly contain a reward, but only enable the transition to such. This transfer of rewards is achieved by different algorithms, in the context of this paper mainly double Q-learning is to be mentioned. Two images of the situation space are used to compute the rewards of each situation. One of the two images does not yet take into account all episodes. The estimated reward of an action is stored as the Q-Value. [11]

It is often inefficient or even impossible to store the action values Q for each state. In problems with a large number of state dimensions, this can quickly exhaust the computer's memory. Storing many individual values also has the disadvantage that when a state is reached for the first time, no estimate can yet be made about its action value. Neural networks allow a more efficient approach here. Artificial neural networks have a similar structure to the human brain. Artificial neural networks were first described in the form of the so-called perceptron by Frank Rosenblatt in 1958. The work was based on the function of a human eye with its biological neurons [12].

An artificial neural network consists of neurons and weighted connections between them. The neurons are typically arranged in columns, so called layers, and follow a Feed Forward approach. Feed Forward in this context means that connections only ever go from one layer to the next and no connection goes from one layer back to a previous layer. If all neurons of one layer are connected to all neurons of the following layer, it is called a densely connected network. The agent uses a deep, densely connected neural network, a network with multiple layers, to reduce the amount of memory required and

to increase processing speed. The training is done in episodes. Meaning one or multiple simulations are run and then the results are processed in batches. After each episode, the weights within the neural network are adjusted to allow the network to approximate the correct decisions.

3 Concept

The concept can be divided into two parts, as shown in Figure 3. First, there is the sequencing simulation that the agent must serve. Previous work in the field of sequencing has shown that it is necessary to create a separate simulation. Prototypical implementations with the software Anylogic could show that these cannot fulfil all requirements for the application. During the training of the neuronal network a large number of runs are simulated. While existing software can easily represent very large production runs in great depth, it is not optimized for high throughput.

A simplified UML diagram of the simulation setup is shown in Figure 2. The training of neural networks for even simple problems requires that the production is run through completely several hundred times. With increasing problem size, the amount of necessary runs also increases, which is not the central design criterion for industrial simulation software.

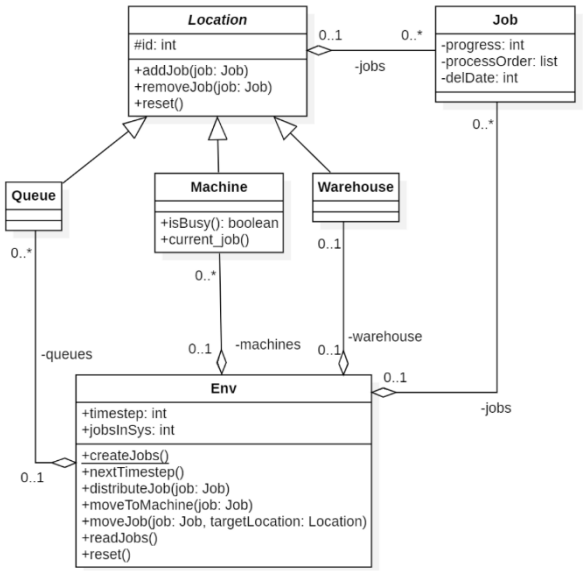

Fig. 2. UML-Class diagram of the Job Shop Simulation

The creation of simulation underlies several design decisions. These decisions are described and justified in the following. In order to go beyond the state of the art in the

context of sequence planning, it is necessary to simulate several machines simultaneously. This is the only way to deal with a multi-machine problem.

The simulation has to be able to represent a real workshop production. A distinction can be made between a flow shop application and a job shop application. Within a flow shop, the individual production steps between all orders are arranged in an identical sequence. This means that the sequence of machines that has to process an order is fixed. This case is close to most real applications, since most products require a fixed sequence of processing steps. A job shop on the other hand allows swapping the sequence within the different jobs. This increases the complexity of the solution space, but at the same time allows more interaction possibilities. Modelling as a job shop problem is more general and can be simplified to a flow shop simulation. The jobs in the simulation must be processed in an individually stored order on the machines in the simulation. Therefore, the job shop approach was chosen for the simulation.

Another feature of simulation and a special feature when considering sequencing is the possibility of not manufacturing on a machine. As already mentioned in the state of the art, most sequencing problems optimize for the fastest possible processing of orders.

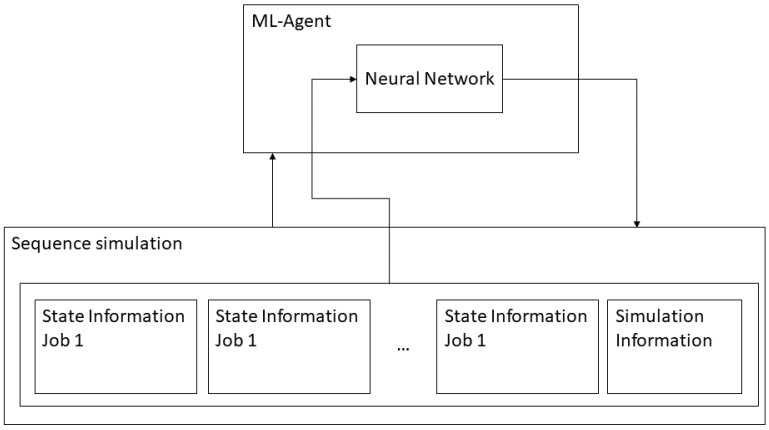

Fig. 3. Schematic of information flow between machine learning - Agent and Simulation

However, as will be explained later, the goal of the systems is to fulfil the orders as timely as possible. Therefore, it is necessary that the individual machines can also simulate waiting.

The created concept for the optimization of sequence planning by a global machine learning agent is outlined in Figure 3. The machine learning agent interacts with the sequence simulation via an interface, which provides information about the status of all orders. In addition, the machine learning agent receives a reward for each action, which can be used to adjust the weights of the neural network. Earlier works deal with the use of local machine learning agents, and the results were not sufficient. Therefore, the approach of a global agent is used here. A comparison of the two approaches is discussed in the abstract.

The concept is divided into two parts based on the components of a Markow Decision Problem. The sequence simulation must represent a workshop production. The environment thus consists of the basic building blocks of a workshop production. The workshop production consists of machines, each of which has a queue. In a queue are orders, which must be processed in the next step at the assigned machine. In order to be able to make this assignment, the processing sequence must be stored for each order. In addition, the progress within the sequence simulation must be saved for each order. Since finished orders should neither be stored in a queue nor in a machine, an additional storage is necessary for complete orders.

4 Implementation

The implementation is based on the programming language C++ and Python. Tensorflow 2 with the Keras API was used as the basis for the neural networks.

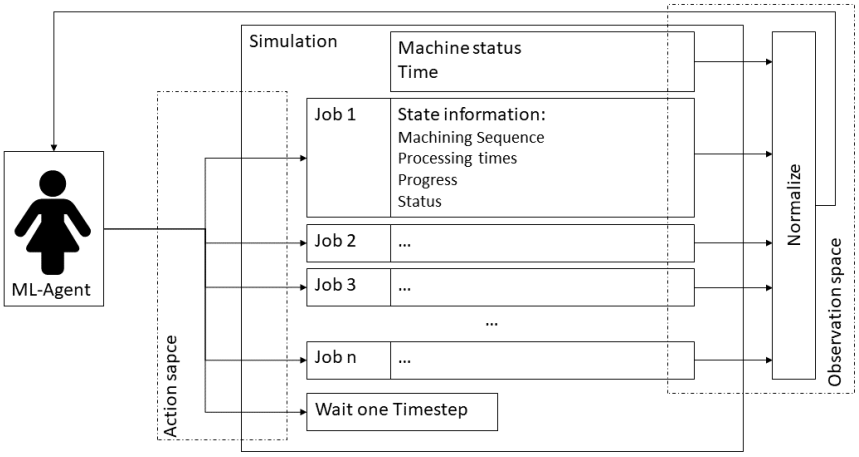

Fig. 4. Implementation of an machine learning -Agent for sequence planning

Figure 4 shows the relationship between the machine learning agent and the simulation and what information is exchanged between these. All actions that the agent can trigger are summarized under the action space. The information that the agent recieves from the simulation is called observation space. The entire simulation does not take place in real time, but is an iterative cycle between simulation and machine learning agent. The simulation is divided into single time steps and after each time step a response of the machine learning -agent is expected.

The response of the agent is limited by the action space. The agent can select a task to be processed. Alternatively, the agent can decide to wait a time step. This option is

necessary because situations can arise in which waiting is the most strategically sensible action in order to maximize the long-term reward.

Since the output of the simulation is to be used directly as input for a neural network, it makes sense that all values have a similar order of magnitude. For this, the value range [0, 1] is suitable, to which the values are linearly scaled in each case. For this purpose, they are normalized by their maximum value. This value is constant over all episodes and is based on a maximum value analysis. For the information of the time step, therefore, the highest possible value for the episode length must always be selected and not the random value, which originates from this uniform distribution and determines the maximum length of the episode.

The simulation state is to be completely described by an array. 2 * number of machines + 3 numerical values are required per job. For each machine an order has the information, which processing step it is and how long the order must be processed at this machine. The information about the processing time on a machine is divided by the largest possible time step in an episode, so that the value is always < 1. The information about the progress of each job is contained in the state-describing array. In addition, it is indicated for each job whether it is currently being processed or not. Finally, the current time step of the simulation is contained in the state array. This array is passed onto the agent as the observation array.

Many simulations must be performed as part of the training. The respective runs are referred to as episodes. An episode ends when all orders have been completed or the maximum number of time steps in an episode has been exceeded. After an episode has been completely run through, the weights within the neural network are adjusted according to the described double Q-learning.

The core of this adaptation is the reward that the machine learning agent can accumulate in the course of an episode. Here, the form of the reward is crucial. The stated goal is to achieve the most accurate delivery fidelity possible. Thus, other than a total time as short as possible, the reward function must also include penalties if an order is completed too early. Here, bell functions have established themselves in order to achieve an exact result.

However, bell functions based on e-functions produce very low values far outside the target value. Since the simulations can run for a relatively long time, this leads to the phenomenon that especially at the beginning of the training hardly any training progress is achieved, since the reward function does not have a significant slope. This is due to the small differences in the function values, which disappear completely due to rounding errors. Therefore, in addition to the traditional bell function, an overlay with a linear function was used. This superposition could only enable the training when the number of simulation steps increased.

The reward function used is therefore as follows for orders that are completed close to their actual target time:

$$R = \propto * e^{\frac{-(C-d)^2}{K}} + (1-\propto) * \frac{C}{d} \tag{1}$$

For orders where C > d, the reward changes, resulting in a linear decrease as the deviation continues to increase:

$$R = \propto * e^{\frac{-(C-d)^2}{K}} + (1-\propto) * (2 - \frac{C}{d}) \quad (2)$$

- C: Delivery date of an order
- d: Finishing date of an order
- K: Factor for manipulating the width of the bell function
- α: Factor for the balance between bell function and linear part

Figure 3 shows that the machine learning agent is based on a neural network for decision making. To implement the machine learning agent, an algorithm that uses double Q-learning is implemented in Python. The algorithm is based on an implementation by Andy Thomas [6]. The training of the transitions takes place offline, the transitions are first stored and used at another time to approximate over the neural networks. This requires storing a transition and retrieving a certain number of transitions from memory.

As illustrated in the explanation of double Q-learning, the agent actually works with two neural networks. Both networks are fully connected feed-forward networks with hidden layers and an output layer. One network serves as the target network and the second one as the primary network. In each time step, the weights of the primary network are adjusted as suggested by Lillicrap [7]:

$$\theta^{Q_{Target}} \leftarrow \tau * \theta^{Q_{primary}} + (1-\tau) * \theta^{Q_{Target}} \quad (3)$$

Double Q-learning was chosen because this approach is advantageous when there is a state action space that is too large to be completely traversed and at the same time there is no pre-trained system. These characteristics are present in this case. [8]

5 Conclusion

To test the system, simulations were first performed with one machine and one job. With this setup, the machine learning agent was able to converge quickly and the basic functionality could be proven. Figure 5 shows that the neural network does not change further after 600 episodes.

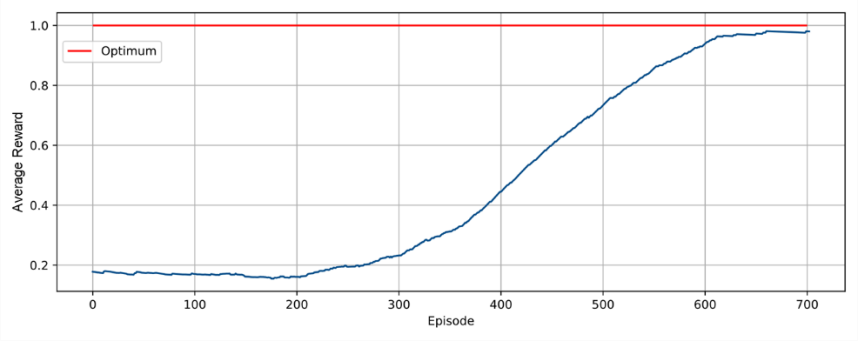

Fig. 5. Average reward with one machine and one job

In the next step, the scope of the simulation was expanded. The number of machines and the number of orders were each increased by one. Two effects occurred. As expected, the training time required to reach a stable state increased. The time required tripled. As can be seen in Figure 6, however, uniform training no longer takes place. These are local optima, which are approximated by the system structure. An example of this can be observed around episode 720. Experiments with different parameters regarding the ratio of how greedy the agent should act could show that a convergence to an optimal solution is not guaranteed.

Similarly, not every configuration could be guaranteed to maintain a steady state near the optimum, but as can also be seen in this iteration, fluctuations occur around the target value. Further tests regarding the scaling of time and the stability will be conducted in further research.

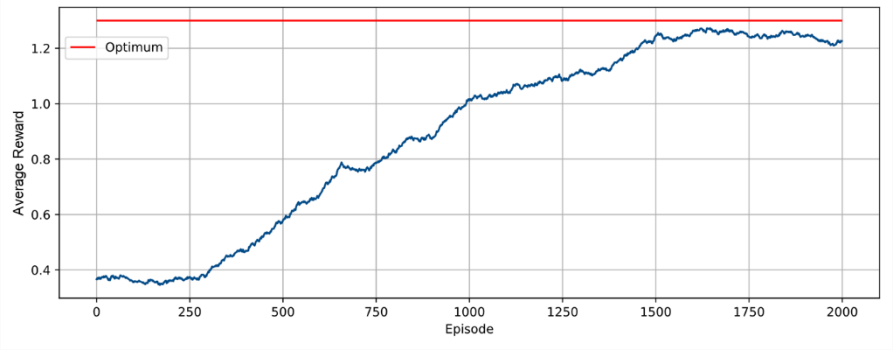

Fig. 6. Average reward with two machines and two jobs

The structure of the implementation further results in an asymmetric network. This can be recognized by the fact that the order of jobs and how these are put down within the observation space, has an influence on the processing. By a rotation of the same orders, the same net can produce different processing orders.

In order to give a final conclusion, a comparison to other disciplines is necessary. The concept is in competition with systems that use priority rules. Furthermore, there is a variation of the concept described here where machine learning agents are not used globally, but locally. Earlier research has shown that it is not generally possible to create convergence of the individual agents in a local application, but that they are in a permanent competition with each other [9].

From this point of view, the concept shows that it is basically possible to implement sequencing by means of a global agent. In contrast to a distribution of the machine learning agent to the individual machines, all orders in the system can be considered. It can be taken into account that an already delayed order, which is currently still being processed on a machine, must be released for processing immediately. This advantage of a global machine learning agent is not only compared to a concept with split machine learning agents, but also to the use of priority rules. Rules such as the least remaining slip rule can only eliminate delays if the machine used for processing is not currently processing another order and processing has already been completed for the delayed order in the previous step.

However, this advantage is also the biggest disadvantage for a global machine learning agent compared to the use of multiple machine learning agents, each of which has a smaller observation space. Accordingly, a global machine learning agent scales poorly with the number of orders, which is expressed by an unstable training.

The Just in Time manufacturing-like option of not releasing an order and waiting for a time step makes the job shop problem many times more complex. Many transitions occur in which no evaluation of the current state of the sequence simulation can be given, so the machine learning agent must wait for a reward until an order is released for the last processing step. This leads to a delayed adaptation of the neural networks. While specific solutions can be implemented here to evaluate orders differently, these are not generally applicable options. Instead, it is necessary to analyse the orders separately in order to reward the agent for every processing step.

References

1. Schuh, G., Stich, V.: Produktionsplanung und –steuerung, Springer, Heidelberg (2012)
2. Jaehn, F., Pesch, E.: Ablaufplanung. Einführung in Scheduling, Springer, Heidelberg (2014)
3. Wuendahl, H.-P.: Betriebsorganisation für Ingenieure, Hanser, München (2010)
4. Swamidass, P. M.: Encyclopedia of Production and Manufacturing Management, Kluwer Academic Publishers, Boston (2000)
5. Lödding, H.: Verfahren der Fertigungssteuerung, Springer, Heidelberg (2016)
6. LNCS Homepage, http://www.springer.com/lncs, last accessed 2016/11/21.
7. Lillicrap, T.: Continuous control with deep reinforcement learning, on ICLR (2016)
8. Thomas. A., https://adventuresinmachinelearning.com/double-q-reinforcement-learning-in-tensorflow-2/, last accessed 2020/03/29
9. Zhou, H.: Concept for Optimization of Scheduling in Production Planning and Control using Machine Learning, TU Darmstadt (2019)
10. Ertel, W.: Computational Intelligence, Springer Vieweg, Wiesbaden (2016)
11. Sutton, R. S., Barto, A. G.: Reinforcement Learning, MIT Press, Cambridge (2018)

12. Rosenblatt. F.: The perceptron: a probabilistic model for information storage and organization in the brain.“ In: Psychological review 65.6 (1958)