Multi-Agent Reinforcement Learning for Job Shop Scheduling in Flexible Manufacturing Systems

1st Schirin Baer
Siemens AG
RWTH Aachen University
Nuremberg, Germany
schirin.baer@siemens.com

2nd Jupiter Bakakeu
Siemens AG
Friedrich-Alexander-Universität
Nuremberg, Germany
jupiter.bakakeu@faps.fau.de

3rd Richard Meyes
University Wuppertal
Wuppertal, Germany
meyes@uni-wuppertal.de

4th Tobias Meisen
University Wuppertal
Wuppertal, Germany
meisen@uni-wuppertal.de

***Abstract*—In this paper, we first outline the motivation and the need for a new approach for online scheduling in flexible manufacturing systems (FMS) based on reinforcement learning (RL). We then present an initial concept for such an approach. In our method, we use Deep RL agents, who have learned to efficiently guide products through the plant and achieve near-optimal timing regarding resource allocation. Each product is controlled by its own agent, which can handle unforeseen machine failures, re-configurations of the plant topology and the consideration of local and global optimization goals. We created a virtual representation of the FMS using a Petri net, modelling the plant topology and the product flow. The agents' task is to navigate the product to the corresponding machine by choosing the according transition of the Petri net. The training consists of four stages from learning rough rules in order to fulfill a job in a Single-Agent RL setup to learning thoughtful collaboration between agents in a Multi-Agent RL (MARL) setup. We proved the feasibility of the first and second training stage by applying it to the virtual depiction of a specific research plant and obtained promising results. With this concept of a self-learning control policy, online-scheduling can be applied with less effort required to customize the setup for various FMSs.**

***Index Terms*—Job Shop Scheduling, Flexible Manufacturing System, Reinforcement Learning, Multi-Agent System**

I. Introduction

Flexible manufacturing systems (FMSs) with multiple possible plant topologies and the ability to produce a huge number of product variants have been in use for several years. Such systems require a well-engineered production control method, including a proper scheduling system, which allocates the resources according to different optimization goals. The optimization problem can be formulated as a Job Shop Scheduling Problem (JSSP) [1]. It is one of the best-known combinatorial optimization problems, where the optimal sequence of processing n jobs in m machines is searched. In the JSSP, one job consists of multiple tasks in sequential order and can be scheduled in $(n!)^m$ possible ways with the assumption, that every machine can process every task [2]. The formal description of JSSP can be found in [1]. Our concept is costumized for an even more complex variant of JSSP, since we consider FMSs with a dynamic plant topology, routing flexibility, a waiting queue in front of all resources and also duplicate resources, each with different properties, e.g. processing time, quality, energy demand, or material costs. Conventional approaches to solve JSSPs are (meta-) heuristics, described in [3] and [4]. These solutions consist of static strategies [2] with the need to model all rules and constraints by a mathematical description manually. This results into high engineering effort for every setup and adaption of the FMS and therefore to high expenses for the plant operator.

Recent research in the field of Artificial Intelligence (AI) shows powerful algorithms for sequential decision-making problems that outperform humans in complex environments such as video games [5]. These algorithms use deep neural networks in order to handle long sequences of states and to find out which action in the game led to the significant advantage to win [5]. These strengths and the high manual effort in the setup of an conventional approach motivate us to apply AI algorithms to solve complex JSSPs in the manufacturing domain with the goal of creating a highly adaptable solution with less engineering effort. Reinforcement Learning (RL) is a Machine Learning method used to solve sequential decision-making problems [6], which goes well with the idea of modelling every process of a product (or task of a job) in an order stack as a sequence in which mutual interdependence decisions must be made. Furthermore, the high dimensional state space due to the high number of situations requires a solution able to generalize for similar situations. With generalization, it is not necessary to learn every single situation by its own, but the model is able to interpolate the right decision even for unseen, but similar situations. This strengthens the idea to use RL in combination with neural networks, which have the ability to generalize, called deep RL. Since only one forward pass of a trained network with very less calculation time is needed to select the action, we can obtain near-real-time decisions at runtime. In comparison to the effort needed to designing and adapt proper rules for conventional scheduling systems, our concept requires less effort to provide an updated virtual depiction of the FMS to train and retrain our self-learning system. We assume that once our concept has been implemented and evaluated in more detail, it will achieve an nearly optimal solution in resource allocation. First prototypical evaluations show such promising results. With these advantages, our approach represents a first step towards the plant operators desire to have low-effort optimal production control.

978-1-7281-4087-2/19/$31.00 ©2019 IEEE
DOI 10.1109/AI4I46381.2019.00014

II. Problem Formulation

In our concept, every product is controlled by its own agent and we consider that each product agent can follow its own optimization goal while also considering a global objective, e.g. minimal total makespan C_{max}. For the JSSP in FMSs, resources can be manufacturing modules, (CNC-) machines or workstations. Transportation between resources can be done by conveyor belts or autonomous mobile robots. The job-list consists of all tasks needed to manufacture the product. A task is described by a list of possible machines able to execute the task and the corresponding properties of these machines.

A. Markov Decision Process

We model this JSSP as a Markov Decision Process (MDP) [6] as we assume, that the selection of an action is only dependent on the current state and not on previous states [6]. With this assumption we decrease the complexity slightly in the first step to be able to proof the concept. The state definition $[J^0, ..., J^n, T^0, ..., T^n, L, PT, L_C]$ is a vector that consists of the job-list vectors $J^0, ..., J^n$ of all n products in the FMS, the vectors $T^0, ..., T^n$ encoding the current task of each job-list, the location vector L of all products, the vector PT describing the plant topology, and the vector L_C defining the location of the controlled product. The state space results from the combination of all possible states of all components of the state definition, leading to an extreme high dimensional state space. The action space results from the number of possible changes of directions to be made in the FMS. The finite state of one training episode is defined by the fulfillment of all job-lists. The reward consists of a feedback regarding the local optimization goal, e.g. energy efficient production or quality of the product and a defined share of the global optimization goal, e.g. makespan C_{max} of all products. The share of the local and global goal can help the agent to chose an action with preference to one of the goals when there are conflicts in fulfilling both goals at the same time.

B. Simulation environment

We modelled the FMS as a Petri net [7], since it is easy to design and adapt and needs less computation time, which is advantageous for training RL agents. In addition, Petri nets are a well-known modelling language to describe distributed systems in a mathematical way. It is a bipartite graph consisting of nodes and transitions and it is fully described by its incidence matrix C. [7] The marking of the Petri net gives us the current location, e.g. of the controlled product, defined by the vector L_{c_1}. After the chosen transition, defined by the vector t_2, is fired, we can easily calculate the new location, defined by the vector L_{c_2}, by the following equation:

$$L_{c_2} = L_{c_1} + t_2 \cdot \ C \tag{1}$$

Our concept considers resources and junctions as places and transportation components as transitions. The resulting simulation was implemented in a Python environment. Each job-list and product location and the plant topology is updated at every time step. Unforeseen machine failures are simulated by the randomly chose of a plant topology, including unavailable resources, in the beginning of every training episode.

III. MARL Based Approach

In our solution, we apply a Multi-Agent RL (MARL) method, with multiple agents, each controlling one product. We decided against one global agent controlling the whole FMSs for multiple reasons. The action space of a global agent would be huge, as it would consist of all combination of all possible actions of all agents at every time step in addition to the already described high dimensional state space. This would force us to use a very deep network with high complexity and with very long training time. With multiple agents controlling the according products, we shift the complexity of high state and action space to the difficulty of collaboration between agents with solely a high state space. This solution is also more flexible, as we can train sub-policies with different optimization goals more easily. In comparison to classical MARL Systems, where every agent observes just a particular part of the system's state, we provide the full state of the system to every agent, leading to an architecture with centralized agents. For this JSSP, we believe, that it is necessary to understand the situation of the whole FMS, as i.e. an agent needs to understand the availability of alternative resources to chose the right decision of resource allocation.

The training concept consists of four training stages from learning rough rules to fulfill the job-list to thoughtful collaboration of multiple agents. The goal of the different training stages is to overcome the common issues in the field of RL, i.e. the credit assignment problem, sparse global rewards and collaboration of agents in transient environments. In our JSSP, the credit assignment problem results from the consideration of a global optimization goal, which can solely be back-propagated to the agents at the end of every training episode as a sparse global reward. An transient environment appears, when updating multiple strategies at the same time.

In the first stage one single agent is trained for various product variants and topologies. Thereby the policy is trained only for a certain part of the state space consisting only the initial states defined by plant topology and the job-list, because no other product are included in the location vector L. The goal is that the agent gets to know the environment and learns valid transitions for each place in the Petri net in order to fulfill all tasks of its job-list. The agent learns to optimize on one local optimization goal, i.e. production time. With that, the agent selects the better alternative, if there are multiple machines available in the job-list for the same task, also considering transportation time to alternative machines, since the transportation time is considered in the production time of the product $t_{product}$ and therefore in the reward.

In the second stage, there are multiple agents applied with frozen policies, but still in the Single-Agent architecture, and only one agent is updated to ensure a stable environment. By including other products, the agent can explore the full state space. The goal of this stage is to adapt the policy to consider other products but without communication and e.g. to select an

alternative, if there are already products waiting in the queue in front of a machine.

In the third training stage the pre-trained policy from stage two is applied to multiple agents in a MARL architecture. A global reward is now considered. A possible MARL architecture can be an actor-critic architecture [8] in which the critic receives, in addition to the observation and action of its own actor, the actions selected by all other actors in the respective time step as input. This method should handle the credit assignment problem. By considering this information and the global reward, the agents should find out, which decision of which agent lead, e.g. to a lower accumulated reward over an episode. Our approach differs from [8] since each actor observes the global state and is therefore centralized. The aim is to learn a strategy to fulfill the job-list by considering its own optimization goal and in parallel to a global goal by making thoughtful decisions with respect to the other agents' alternatives. As every agent has the same initial policy and is trained for the same goal to process various product variants in various topologies, we can select the policy, who performed the best in the end of the training and apply it to all agents during runtime, as long as they have the same local optimization goal. For various optimization goals, sub-policies can be trained from this policy.

In the fourth stage, the trained policy is deployed to the real plant and fine tuned. The aim is to consider uncertainties, such as divergent transportation or processing times, that are not present in the virtual environment.

As we are still in the phase of proofing this concept, we validated the Single-Agent architecture using different RL-algorithms. For the first evaluation we considered the processing time as the only optimization goal, leading to a reward design according to the reverse processing time scaled in the range $[0, 1]$. If the selected action led to an invalid transition, the reward for this action was set to -1, for unneeded machines to -0.5 and for unavailable machines to -0.75. For the validation we used our modular research FMS with six different manufacturing modules and six positions, they can be located at. Our Petri net consists of 12 places and 24 transitions, including self-transitions for the opportunity to stay in a manufacturing module for two following tasks. We trained one RL agent in the first and second training stage, but for a small state space, and obtained promising results. The agent was able to process the desired product by choosing the fastest manufacturing module for every alternative, even considering transportation time. It prefers modules, which were able to process two tasks in a row. We evaluated this for various plant topologies and spontaneous module failures and it worked very well.

IV. Related Work

Heuristics and Meta-Heuristics are conventional approaches, which lead to an approximation to the optimal solution of this high complex JSSP [3, 4]. Reference [4] reviews meta-heuristic approaches like Tabu Search, Ant Colony and Bees Algorithm. [9] introduces techniques to accelerate the backtrack search procedure in the use of heuristics to solve the constraint satisfaction job shop scheduling problem. Nevertheless, due to the high complexity, these solutions are often time and computationally intensive and have static strategies [4], leading to high engineering effort for the setup and adaption to every change in the FMS. Approaches like Monte Carlo Tree Search are computationally intensive and knowledge-based systems need more computation time at runtime than trained systems [4]. Inductive learning was used in [10] to learn a state dependent scheduling policy, where dispatching rules are dynamically chosen depending on the current state of the system. RL approaches to solve JSSPs with online scheduling have been presented in [11, 12]. Both approaches use decentralized MARL architectures, trained to find proper dispatching rules for every machine. As in most RL approaches, RL agents control the machines and choose between predefined dispatching rules while communicating with each other. In contrast to the former approaches, we are using the strengths of Deep RL to solve long sequential decisions, where decentralized agents control the products. By controlling the products and not the machines, we include the consideration of transportation time in the decision making. In our approach, no direct communication between the agents and no central entity is necessary. We also hope, that with the global observation of our agents, we can achieve a near-optimal solution, as more necessary context information is used for the decision making. We do not use predefined rules, that enables us to find new and unexpected strategies.

V. Outlook

In the next steps, we plan to evaluate all training stages with the full state space. With the consideration of one module failure for every plant topology, we result 5040 different topologies. With 60.000 different product variants, which can be produced in our FMS, we already obtain a state space of $3\text{x}10^8$ for the initial state. The state space for the second state of an episode is even higher, as is contains many possibility of product locations and it increases accordingly for further states. We also plan to apply actor-critic based algorithms, as they are advantageous for stage three. We will further investigate which RL-algorithm is most suitable to handle the high dimensional state space or if it is necessary to train various sub-policies for similar groups in the state space. For more advanced scenarios in an FMS with complex constraints, where previously chosen modules effect the current action, we can adapt this problem formulation to a Partially Observable MDP and introduce recurrent policies or the explicit consideration of previous states in the decision making. With this promising concept we aim to achieve a scalable solution, adaptable to changes, for online scheduling in FMSs and we are looking forward to present an evaluation of the presented concept in a following paper.

REFERENCES

[1] Garey, M. R, Johnson, D. S, and Sethi, R., The Complexity of Flow Shop and Job Shop Scheduling. Math. of Operation Research,1976 , 117-129.

[2] Fera M., Fruggiero F., Lambiase A., Martino G. and Nenni M. E., Production Scheduling Approaches for Operations Management, 2015, pp. 119-139.

[3] Reeves, C. R., Heuristic search methods: A review, in: Operational Research Society, Birmingham, UK, 1996, pp. 122-149.

[4] Zanakis, H. S, Evans, J. R, and Vazacopoulos, A. A., Heuristic methods and applications: a categorized survey, in: European Journal of Operational Research 43, 1989.

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, et al., Playing atari with deep reinforcement learning, 2013, arXiv: 1312.5602.

[6] Sutton, R. S., and Barto, A. G., Reinforcement learning: An introduction, 2011.

[7] Zhou, M., "Petri nets in flexible and agile automation", Springer Science Business Media, Vol. 310, 2012.

[8] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., et al., "Multi-agent actor-critic for mixed cooperative-competitive environments", in: Advances in Neural Information Processing Systems, 2017, pp. 6379-6390.

[9] Sadeh, N., Sycara, K., and Xiong, Y. "Backtracking techniques for the job shop scheduling constraint satisfaction problem", in: Artificial intelligence 76 (1-2), 1995, pp. 455-480.

[10] Chan Park, S., Raman, N., and Shaw, M., "Adaptive scheduling in dynamic flexible manufacturing systems: A dynamic rule selection approach", IEEE Transactions on Robotics and Automation 13(4), 1997, pp. 486 - 502.

[11] Gabel, T., and Riedmiller, M., "Scaling adaptive agent-based reactive job-shop scheduling to large-scale problems", in: IEEE Symposium on Computational Intelligence in Scheduling, 2007, pp. 259-266.

[12] Waschneck, B., Reichstaller, A., Belzner, L., Altenmller, T., Bauernhansl, et al., "Optimization of global production scheduling with deep reinforcement learning", 2018, in: Procedia CIRP, 72(1), pp. 1264-1269.