UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

HAW HAMBURG

TFG del Grado en Ingeniería Informática

Report: Potential of Reinforcement Learning in Flexible Modular Production Systems in Theory and Practise

Presentado por Helen Haase
en Universidad de Burgos
January 17, 2022
Tutor: Bruno Baruque Zanón

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D.Bruno Barque Zanón, profesor del Departamento Ingenería Informática, Àrea de Ciencia de la Computacíon e Inteligencia Artificial

Expone:

Que el alumno D. Helen Haase, con id L1PP0F92X, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, January 17, 2022

V^{o}. B^{o}. del Tutor:	V^{o}. B^{o}. del co-tutor:
D. nombre tutor	D. nombre co-tutor

Resumen

Este trabajo se centra en la investigación de estrategias para resolver un problema de programación del taller flexible (FJSP), como problema concreto de los conocidos como sistemas de fabricación flexibles (FMS) y en el desarrollo de un prototipo basado en el aprendizaje por refuerzo (RL) como prueba de concepto para mostrar las posibilidades de optimización del ajuste de los hiperparámetros.

En la parte teórica de este trabajo se muestran diferentes estrategias que se han utilizado en la literatura científica para intentar resolver los problemas de programación. Estos enfoques comprenden algoritmos exhaustivos y aproximados, así como el aprendizaje por refuerzo.

En el apartado práctico se construye un prototipo que puede ser visto como una demostración de una posible solución con Aprendizaje por Refuerzo. Incluye un sistema de producción modelado en Siemens Plant Simulation (SPS) y la optimización de políticas proximales (PPO), así como un agente de doble Deep Q-Network DQN, que son entrenados y comparados. Los experimentos muestran la relevancia del ajuste de los hiperparámetros y los rendimientos resultantes.

Descriptores

Double Deep Q-Network (DQN), Flexible Manufacturing System (FMS), Hyperparameter tuning, Matrix Production, Proximal Policy Optimization (PPO), Reinforcement Learning (RL), Siemens Plant Simulation (SPS)

Abstract

This work focuses on investigating strategies to solve a Flexible Job Shop Problem (FJSP) scheduling problem regarding a Flexible Manufacturing System (FMS) and developing a Reinforcement Learning (RL) based prototype as proof of concept to show optimization possibilities of hyperparameter tuning.

The theoretical part of this work shows different strategies, that have been used to try and solve the scheduling problems by going through the search space with different techniques. These approaches comprise exhaustive and approximate algorithms as well as Reinforcement Learning.

In the practical part a prototype is built, than can be seen as demonstration of a possible solution with Reinforcement Learning. It includes a production system modelled in Siemens Plant Simulation (SPS) and Proximal Policy Optimization (PPO) as well as a double Deep Q-Network (DQN) agent, that are trained and compared. Experiments show the relevance of hyperparameter tuning and the resulting performances.

Keywords

Double Deep Q-Network, Flexible Manufacturing System (FMS), Hyperparameter tuning, Matrix Production, PPO, Reinforcement Learning (RL), Siemens Plant Simulation (SPS)

Contents

Contents **iii**

List of Figures **v**

List of Tables **vi**

List of Acronyms **vii**

1 Introduction **1**
1.1 Outline of the Documentation 2

2 Project Objectives **5**

3 Theoretical Concepts **7**
3.1 Flexible Manufacturing System (FMS) 7
3.2 Flexible Job Shop Problem (FJSP) 8
3.3 Reinforcement Learning (RL) 8

4 Tools and Technologies **9**
4.1 Simulation Environment 9
4.2 Reinforcement Learning Libraries 10
4.3 Other Tools 10

5 Relevant Aspects of the Project Development **11**
5.1 Motivation 11
5.2 Project Idea 11
5.3 Related Work 12
5.4 The Main Components: Agent and Simulation 13
5.5 Data Exchange 14

5.6 Experimentation 15

6 Conclusion and Future Work 17

6.1 Tuning the Agent 17

6.1.1 The State Space 18

6.1.2 The Action Space 18

6.1.3 The Agent 18

6.1.4 The Reward Function 19

6.2 The Production System 19

6.2.1 The software 19

6.2.2 Biasing information 20

6.2.3 Efficiency of the Schedule within the Planning Pyramid 20

6.2.4 Application in real world scenarios 21

6.2.5 Exploding computation costs 21

6.3 Experimentation 21

Bibliography 23

List of Figures

1.1 Document structure, the topic "Motivation and Definition of the Problem", "Related Work and Theoretical Concepts", Experiments and Results" are included in Annex B, the "Software Development of the Prototype" can be found in Annex A 3

3.1 Example of a production system with two product types, one source, one drain and six working stations, oriented on work of Greschke [5] 7

List of Tables

List of Acronyms

A2C Advantage Actor Critic

A3C Asynchronous Advantage Actor Critic

AGV Automated Guided Vehicle

DRL Deep Reinforcement Learning

DQN Deep Q-Network

EDD Earliest Due Date

ERM Experience Replay Buffer

FIFO First In First Out

FMS Flexible Manufacturing System

FJSP Flexible Job Shop Problem

IID independent and identically distributed

JSP Job Shop Problem

KPI Key Performance Indicator

KBGA Knowledge Based Genetic Algorithm

MDP Markov Decision Process

ML Maximal Lateness

MP Matrix Production

NN	Neuronal Network
NTJ	Number of Tardy Jobs
ODBC	Open Database Connectivity
OPC UA	Open Platform Communications Unified Architecture
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
SGA	Simple Genetic Algorithm
SRPT	Shortest Remaining Process Time
SPS	Siemens Plant Simulation
SPT	Shortest Processing Time
TD	Temporal Difference
TCT	Total Completion Time
TL	Total Lateness
TRPO	Trust Region Policy Optimization
TT	Total Tardiness
UU	Uptime Utilization
WIP	Work in Progress
XML	Extensible Markup Language

Chapter 1

Introduction

This work focuses on investigating the potential of Reinforcement Learning to solve a Flexible Job Shop Problem (FJSP) regarding a Flexible Manufacturing System (FMS) by building a prototype as proof of concept based on different recent approaches. Furthermore it is used to show optimization possibilities of hyperparameter tuning.

The background is the need to design and control production systems, that fulfill modern requirements and which shift the original way of planning production to a highly complex control.

Since 1913 the setup of a general line production system, as originally designed by Ransome Eli Olds for the the car industry[13], has barely been adjusted, while the demands on modern production system has changed significantly.

While in the early 20th century product versatility and customization was not demanded, modern product variety requires a number of different and more complex products. *According to Audi, there are theoretically* $1.1x10^{38}$ *possible configurations of the A3 model alone*[10]. But this is not the only change, that has occurred. The duration of the market cycle continues to decrease while planning uncertainty increases due to global expansion and complex supply chains [6].

Additionally line production systems have always been very sensitive to malfunctions and breakdowns of conveyor belt or other production assets, as it can lead to a partly or entire blockade of the entire production system. This alone would call for an alternative approach to existing production systems, that are designed to be more modular. A system is required to adapt to changing demands and to the uncertainty of daily businesses of a fast past world, that we find ourselves in today.

Flexible Manufacturing System (FMS) or Matrix Production (MP) could be an answer to the emerging requirements. The system is still relatively new, which is why it is still confronted with a variety of challenges like the allocation of work centers (layout planning), product design, material flow, route planning or schedule of production [6].

The scheduling problem is one of the most analysed issues in this context. [3]. Being highly challenging, it has always attracted not at least due to the NP-hardness of the problem.

Current Machine Learning approaches are often used to try and solve scheduling problems [3]. This trend probably originates from the increase of computing capacity and the availability of information (Moore's law), that provides an option to explore NP-hard problems. In this work also Reinforcement Learning has been chosen as a further mean to solve a scheduling problem.

The presented work is research oriented and therefore includes an overview of the underlying theoretical concepts, the related work ranging from early days to state of art approaches. Examining current solutions serve as a baseline for the author's own attempt to build a prototype as proof of concept. Using this prototype different experiments have been performed to shed light on the topic from a practical point of view.

1.1 Outline of the Documentation

The documentation of this project is split into three different documents. While this document gives a general overview over the entire project, the other two documents cover the theoretical background and the practical prototype as a proof of concept. As a guide the figure 1.1 shows the topics covered in Annex A and B.

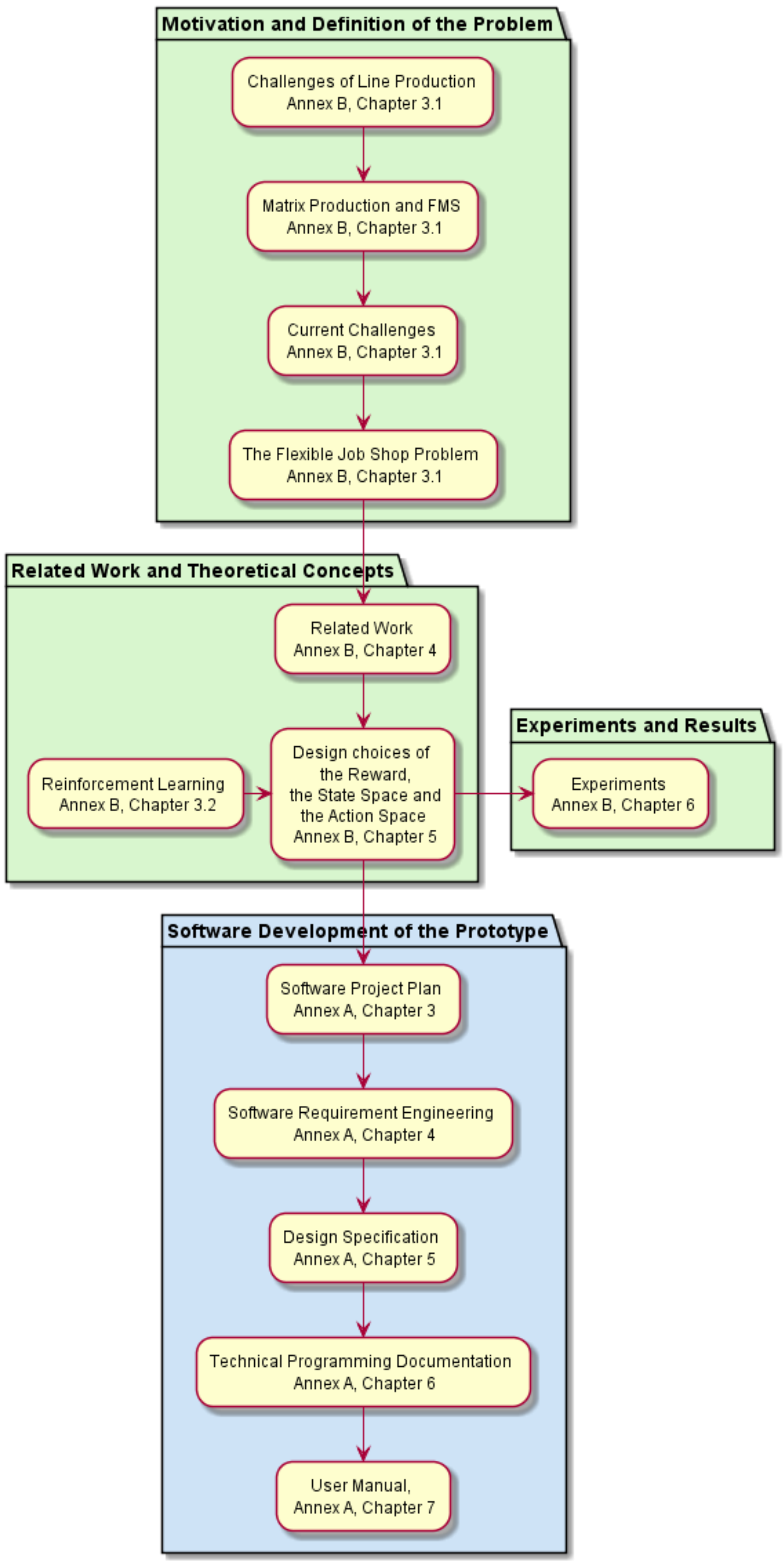
Motivation and Definition of the Problem
Challenges of Line Production Annex B, Chapter 3.1
Matrix Production and FMS Annex B, Chapter 3.1
Current Challenges Annex B, Chapter 3.1
The Flexible Job Shop Problem Annex B, Chapter 3.1
Related Work and Theoretical Concepts
Related Work Annex B, Chapter 4
Reinforcement Learning Annex B, Chapter 3.2
Design choices of the Reward, the State Space and the Action Space Annex B, Chapter 5
Experiments and Results
Experiments Annex B, Chapter 6
Software Development of the Prototype
Software Project Plan Annex A, Chapter 3
Software Requirement Engineering Annex A, Chapter 4
Design Specification Annex A, Chapter 5
Technical Programming Documentation Annex A, Chapter 6
User Manual, Annex A, Chapter 7

Figure 1.1: Document structure, the topic "Motivation and Definition of the Problem", "Related Work and Theoretical Concepts", Experiments and Results" are included in Annex B, the "Software Development of the Prototype" can be found in Annex A

Annex A: Software Development Plan outlines the different aspects relevant for developing the prototype.

Annex B: Theoretical Backgrounds and Experimentation covers the necessary technical concepts for the project and includes the evaluation of the performed experiments.

This document offers a general overview of the project covering the main topics and the results elaborated in Annex A and B.

Chapter 2

Project Objectives

The aim of this project is to evaluate whether Reinforcement Learning is a possible approach to optimize a Flexible Manufacturing System (FMS) with regard to the minimal makespan for a defined demand of products. To accomplish this an analysis a theoretical and a practical analysis will be performed. This project is mostly research oriented which explains the slightly different approach and documentation.

The following list covers the main aspects of the presented work.

1. State of the art analysis of selected algorithms used for solving scheduling problems of a flexible modular production system [refer to Annex B chapter 4 and chapter 4.5 of the Annex A]
2. Development of a simulation for a Flexible Manufacturing System that can be used as an environment for training reinforcement algorithms, including researching common representations and constraints of production systems [refer to Annex A and chapter 5 of Annex B]
3. Implementation and optimization of an Reinforcement Algorithm [refer to the repository, that can be obtained from following this link: `https://github.com/xHelenx/matrix-optimization`]
4. Evaluation of the results and outlook of future projects [refer to Annex B chapter 6 and chapter 6 of the Report]

Chapter 3

Theoretical Concepts

3.1 Flexible Manufacturing System (FMS)

A FMS describes a production system, which is very flexible, as it offers, in contrast to line production, a variety of working stations. At each working station different production steps of the production of a product can be performed. Hence it is modularly built system, making it easy to switch working stations on and off depending on the current demands. Working stations can not only complete multiple working steps but can also be redundant in the system. Figure 3.1 visualizes a possible setup. A thoughtful control of the schedule is required. chapter 3.1 of Annex B examines more deeply differences of line production and a FMS, showing also current challenges and defining the vocabulary.

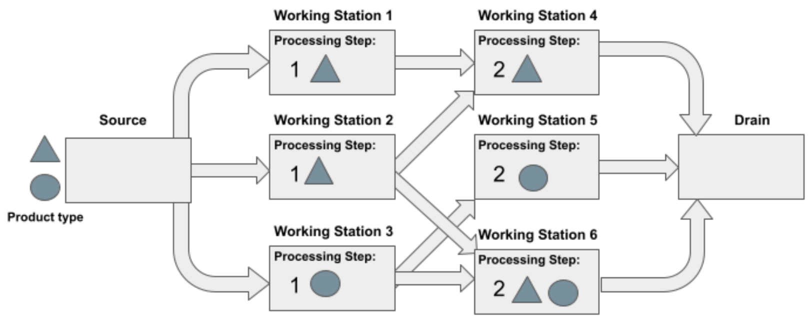

Figure 3.1: Example of a production system with two product types, one source, one drain and six working stations, oriented on work of Greschke [5]

3.2 Flexible Job Shop Problem (FJSP)

A FJSP is the definition describing the problem, that the RL agent tries to solve within the context of a FMS during this project. It extends the Job Shop Problem (JSP) problem.
The JSP describes the problem of assigning different jobs to time slots, such as example working stations or machines. This is done while aiming to reduce the total runtime. The FJSP extends this problem insofar, that not only one machines can be chosen but one or more. It is a strongly NP-hard problem.
The FJSP is described in chapter 3.1.3 of Annex B.

3.3 Reinforcement Learning (RL)

RL is a subfield of Machine Learning, in which an agent tries to learn from the interaction with an environment. The decision making process makes it suitable for optimization problems, as it tries to find a solution in the solution space, without trying every option, but using a smarter search strategy.

The interaction of the agent and the environment begins as the agent receives a state of the current environment. It then decides on an action, that the environment shall perform. Based on the action, that was performed in the current state the environment gives a reward or a punishment to the agent as feedback. The agent tries to maximize the reward, which is calculated through a reward function, that is mappable towards the actual optimization goal.
In chapter 4 of Annex B, the concept is explained in more detail. It includes a definition of a Markov Decision Process (MDP), that is used to define the problem. Afterwards the state space, action space and reward system are defined and an overview of different algorithms is given.

Chapter 4

Tools and Technologies

4.1 Simulation Environment

As the Reinforcement Algorithm was supposed to be implemented in python, both *OpenAI Gym* [4] and *SimPy* [17, 11] libraries were examined. A third possible option is *Siemens Plant Simulation* [15], which was ultimately the chosen one. In the following the reason for this choice will be shown.

OpenAI Gym offers a few predefined environments, but these were not suitable for the problem set.
On the other hand *SimPy* is a framework, that is used for discrete-event simulation. A clear disadvantage was the missing visual representation, that might prove useful for analysis and demonstration purposes. Since the author had to be trained in the Siemens Plant Simulation (SPS) software anyway, it made sense to use this tool for the project as well. The view was, that a graphical interface for modelling the production system would be more intuitive and better to comprehend.

As the use of a student license proved necessary, only few interfaces for data exchange were offered, limiting the software design. A file interface and a XML-interface is offered for this license type[14]. A prototype has been built as proof of concept for the interface to check the possibility of interchanging information, even though it is usually used to import and export files before beginning the simulation. However when the project started and the communication increased, the XML-interface and the absence of wider synchronization tools within SimTalk - the simulation programming language - made the implementation more difficult.

4.2 Reinforcement Learning Libraries

The RL agent was implemented using the Tensorforce [7] library, provided by Kuhnle et al.. When choosing the most suitable library Keras, Tensorforce and OpenAI Baselines have been compared [1, 16], even though there also other libraries. The most important requirements were, that the library provided state of the art RL methods, an easily customized environment and a relatively intuitive approach, to quickly get started.

Keras does not offer a lot of RL methods, nor is it easy to use your own environment. OpenAI Baselines was lacking documentation for implementing customizing environments or getting started easily.

But *TensorForce [...] targets both industrial applications and academic research. The library has the best modular architecture we have reviewed so far. Therefore, it is convenient to use the framework to integrate customized environments, modify network configurations, and tweak deep RL algorithms* [12].

4.3 Other Tools

Tools to organize the research, project and documentation further tools were used. These are listed in chapter 3.1.2 of Annex A.

Chapter 5

Relevant Aspects of the Project Development

5.1 Motivation

Before starting the project, the author had already decided that a major part of the project should be research oriented. One reason being that it offers the opportunity to spend time on practising how to learn new concepts and deepening the understanding of a so far unknown topic, while being challenged to share and condense the newly gained information. Reinforcement Learning had already caught the author's attention for a while, so having a project with the ability to finally dive into the capabilities of RL was exciting.
The second reason is, that the author is generally interested in learning more about current research in this field.
Reading a variety of different papers and publications also provides insight into research work.

5.2 Project Idea

The problem to solve is the following. Imagine, that one type of product has to be produced in a production system. Therefore for instance three processing steps have to be performed. The second one takes the longest time, so having redundant machines performing the second processing step makes sense.

Imagine now, that the production system not only offers redundant machines. Adding further complexity such as larger product palettes, more processing steps or more machines makes it more and more difficult to find a schedule reducing the makespan for a defined demand. For this reason it makes sense to automate creating a schedule.

5.3 Related Work

In order to gain knowledge about common scheduling approaches the literature has been reviewed. The following sections summarizes the different approaches encountered.

These approaches range from using exhaustive searching algorithms, to approximate searching algorithms to learning algorithms.

In the past mathematical programming methods were popular, as they could encounter optimal solutions. But when the problem size increased, computational cost exploded and it was no longer possible to reach optimal solutions. Some algorithms still aimed to find optimal solutions and tried strategies to reduce the search space by cutting out already established sub-optimal solutions.

When sub-optimal solutions were sufficient, approximate searching algorithms could be used. These algorithms use heuristics to find a solution. These are often problem specific since the heuristics include domain knowledge. This led to the question, on how the optimization problem in general could be improved.

So Hyper-heuristic and Meta-heuristics approaches became more relevant, as they abstract from the specific problem and try to optimize the used algorithms and their configurations.

In the end RL has been selected to try and find a good schedule within a defined production system

Another reason for the extended research study is, that comparing different RL approaches was helpful to gain knowledge about the previous attempts to configure the agent. This made it easier to make design choices for this project.

For a better understanding chapter 3.2 of Annex B explains the most important concepts of Reinforcement Learning, like the reward action and state space. Chapter 4 of Annex B shows the related work more in depth.

5.4 The Main Components: Agent and Simulation

In order to solve the given problem two main components are required. One is a simulation of the production system and the other is the agent learning the schedule for this system.

For the simulation it is first of all important to define its setting. Questions regarding the size, amount of processing steps and product portfolio have to be answered. After doing so it is important to consider which limitations are set for the simulations. For example it has to be decided, whether machines break down or whether the allocation of transportation vehicles is considered. Are transportation times, material allocation as well as tool allocation included in the scope of the problem? These design choices ultimately influence the performance but also differently reflect how close the simulation is to reality.

The second component to be considered is the agent. Also here different questions appear regarding its design. The following list gives an overview of these:

- Which type of actions can the agent perform? A typical approach is to allow the agent to move items from a origin to a target. It is possible to allow the agent to also wait for the system to continue before choosing an action
- What are valid actions within the agent's action space? For example an agent may suggest to move a product to processing step, that has already been performed. The questions is if the agent should have

the possibility to perform this action or how the system reacts to performing impossible actions?

- Also depending on the current state of the production systems some action considered as possible may not be valid. For instance it may be correct to move a product from processing step 2 to 3, but what if the machine performing the third processing step is occupied? The action itself is possible, just not at the current moment. Should a solution like this be suggested to the agent or excluded from the agents space altogether? Does it makes sense to dynamically change the action space?

- Which information about the production system are required for the agent to choose the next action?

- Which information could be added additionally to help the agent learn important dependencies within the system?

- How much information is helpful, when does information rather confuse the agent?

- How does the agent learn to improve its behaviour?

- How can a reward function reflect the goals the agent should optimize towards? How do the different aspects have to be balanced? How can this be done avoiding too much domain knowledge letting the agent explore unique solutions?

The list shows that one of the most important parts of this project was to design the agent. That is why chapter 5 of Annex B explains the design choices for the action space, state space and the reward function in detail.

After designing the agent and the simulation, the training environment for the agent, it is important for both components to exchange information.

5.5 Data Exchange

The data exchange had to use XML-files, as the student license only included this option. Also the internally used programming language SimTalk on the production system's side did not offer any synchronization possibilities making this part of the project more difficult as anticipated.

In chapter 5 of Annex A the data exchange is explained and a sequence diagram shows the process in detail.

At last the agent could train from performing action in the environment.

5.6 Experimentation

After gaining a theoretical understanding as well as a practical point of view, which includes learning about different libraries, best practises and common problems, the experimentation part including hyperparameter tuning was started. It offers the opportunity to combine theory and practice, as the tuning of the agent requires a good understanding of the underlying mechanisms.

Three different categories for the experiment have been selected. These include parameters regarding the training duration, the agent configuration and problem specific configurations.

The methodology, detailed experiment configurations and results are presented in chapter 6 of Annex B.

Chapter 6

Conclusion and Future Work

This project has analyzed the potential of RL for the optimization of the schedule of a FMS. For this goal some of the current research regarding the JSP in Production Systems has been collected. Furthermore a prototype has been developed to show the possible interaction between an RL agent and a simulation software. The agent was then improved with hyperparameter tuning, showing clearly the relevance of the different parameters as shown in chapter 6 of Annex B.

On a personal note the author acquired new skills and knowledge during this project by exploring previously unknown fields of Machine Learning and Scheduling. Additionally basic user skills with the software SPS were gained.

All in all the work fulfilled its purpose on being a first prototype for evaluating the potential of RL for the given problem set. In the following section the author outline further considerations as well as suggestions to continue the project. These are split regarding the agent, the production system and the experimentation.

As the results of the experiments have already been outlined, they are not repeated here, but can be found in chapter 6 of Annex B.

6.1 Tuning the Agent

There are several ways to continue tuning the agent.

6.1.1 The State Space

It would be interesting to experiment with different amounts of accessible information for the agent regarding the state of the production system. The question is, does more information help or confuse the agent? Depending on the chosen data representation, it would be possible to include some information multiple times directly or indirectly to stress their importance.

Luckily SPS offers nearly every characteristic information of the system at any given time, so extending this information to the state space is no time consuming task.

6.1.2 The Action Space

In this approach the agent had to learn to distinguish between valid and invalid actions. Depending on the scale of the environment, it is possible to avoid, that the agent has to learn the difference. For example the action space could be dynamic and only the valid actions are presented to the agent. Another idea would be to store all actions in a ring buffer. Whenever an invalid action is chosen, the next valid action is taken from the ring buffer.

Another idea is to add a waiting action, as suggested by Kuhnle et al. [9], so that the system can choose not to move parts instantly.

6.1.3 The Agent

As shown in the results, the current parameters were not suitable for the double DQN. So another grid search could be implemented, focusing on double DQN instead.

It also might be interesting to tune further parameters on the existing agents. For the PPO agent for example it could for instance be of relevance to experiment with the clipped probability rate, which defines the range for the difference between the current and the new policy. Tensorforce offers to adjust this with the *likelihood_ratio_clipping* parameter.

The double DQN also has at least one further parameter that would be interesting to experiment with. Tensorforce offers the option to vary the size of the Experience Replay Buffer (ERM).

In general an even deeper understanding of the adjustable parameters would help to tune the agents even better to the current environment.

Furthermore other algorithms could be tried out as well.

6.1.4 The Reward Function

There are also more reward functions that could be tested. For instance it was not tested how important the bonus that is given, when a part is moved to a drain, is.

Also it might be interesting to change the importance of the *totalTime* and the recognition of valid actions over time. Perhaps punishing invalid actions strongly in the beginning, but decreasing the punishment over time, would also shift the agent's priorities of optimization.

It could also be interesting to punish the agent not linearly for the used system, but with a logarithmic scale, in order to not punish relatively small *totalTimes*, but heavily weigh a very high *totalTime*.
Now that some data has been acquired in the range of the different Key Performance Indicators (KPIs), generally changing the weights of the reward function becomes easier.

6.2 The Production System

6.2.1 The software

The computation time of the training of an RL agent is naturally very high. The current implementation of the communication between agent and environment probably also played into this and additionally increased the time additionally. If further experimentation with this setup shall be done, it is strongly advised to improve the communication. In this case it was not possible, as the upgraded software license was not free of charge.

However if this project will be further developed, it would make sense to invest in a better license and faster hardware or servers for training the agent.

Maybe it would be worth to write a python simulation for production systems to keep the communication to a minimum. The author also found a possibly

suitable implementation called *SimRLFab* close to the end of the project, that was developed by Kuhnle and his teams [8].
The offered features appear sufficient for a similar sized project.

6.2.2 Biasing information

Designing the state and action space as well as the reward function, always leads to biasing the agent, since the developer uses her or his own domain knowledge to weight the importance of different information. In fact, *by using domain knowledge, the risk arises that the agent can no longer find the optimal strategy, as that is usually not known by any domain expert*[9]. This also means it is difficult to try avoiding to bias the agent and at the same time to provide an amount of information that does not fully confuse the agent or makes it impossible to train the agent in finite time.

6.2.3 Efficiency of the Schedule within the Planning Pyramid

In general it is to say the higher the level of the planning pyramid, the more inefficient are its outcomes. The planning steps include product planning, planning of the production system, planning of the production and scheduling. So even though improvements are made on the top level of the planning pyramids, there are still many more fields that are adjustable and intersect. This makes it impossible to only consider one subproblem.

For instance considering different subproblems like material handling and tool allocation individually has two main shortcoming *First, focusing on the machines while generating a schedule and disregarding other influencing resources results in a production schedule with a lower potential to be fulfilled. Second, the machine-oriented optimization of the production scheduling leads to a decreased utilization of the other resources* [2].

The variety of subproblems that exist might be solvable with different learning algorithms, if they are not to heavily impacted by human bias, so that they offer new ways of looking at the problem, when yielding unusual solutions.

It needs to be mentioned, that Matrix Production is still a very new approach, which offers a unique opportunity to work on different subproblems with less constraints, than an alternative system, that only has the goal of exchanging one part of their production chain.

6.2.4 Application in real world scenarios

The current system only included 10 machines and two types of products, which is nowhere near realistic quantities of a real production system. Unfortunately trained RL agents are not easy to transfer to different production systems, as they are trained for a specific setup. Additionally there is the problem, that there is no clear way to optimize RL agents. It is not possible to change one value of the hyperparameters and then be sure, that the solution will scale up.

Transfer learning might offer an interesting approach depending on the differences between the production system.

When continuing to work in the current setup, it seems a good idea to slowly increase the complexity by dropping certain constraints. For instance breakdown and reparation times could be added.

6.2.5 Exploding computation costs

When increasing the size of the state and thus the agent space, the calculation time will increase tremendously. Additional to using parallel trainable algorithms and having stronger hardware, it might be possible to at least slightly decrease the action space with combinatorial analysis.

In this software for example actions, that were not possible, like moving an item back to the source, were all together excluded from the action space, that the agent considers.

6.3 Experimentation

For future experiments some aspects can be improved. First of all when an agent has an early stop, it makes more sense to set all future values to a very high ones. This makes it easier to evaluate the experiments. Alternatively another KPI could be introduced tracking the produced demands of an episode. This could then mask the early stopping effect.

Furthermore it could be interesting to store the chosen actions to analyse encountered patterns.

Bibliography

[1] Mauricio Fadel Argerich. 5 frameworks for reinforcement learning on python. `https://towardsdatascience.com/5-frameworks-for-reinforcement-learning-on-python-1447fede2f18`, 2020. Last accessed: 12-01-2022.

[2] Iman Badr and Peter Göhner. An agent-based approach for scheduling under consideration of the influencing factors in fms. *IECON Proceedings (Industrial Electronics Conference)*, pages 2405 – 2410, 2009.

[3] Massimo Bertolini, Davide Mezzogori, Mattia Neroni, and Francesco Zammori. Machine learning for industrial applications: A comprehensive literature review. *Expert Systems with Applications*, 175(2):114820–114849, 2021.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] Peter Greschke, Malte Schönemann, Sebastian Thiede, and Christoph Herrmann. Matrix structures for high volumes and flexibility in production systems. *Procedia CIRP*, 17:160–165, 2014.

[6] Peter Immanuel Greschke. Matrix-produktion als konzept einer taktunabhängigen fließfertigung. pages 8–15, 97–99, 2016.

[7] Alexander Kuhnle, Michael Schaarschmidt, and Kai Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. `https://github.com/tensorforce/tensorforce`, 2017. Last accessed: 12-01-2022.

[8] Andreas Kuhnle. Simrlfab. `https://github.com/AndreasKuhnle/SimRLFab`.

[9] Andreas Kuhnle, Jan-Philipp Kaiser, Felix Theiß, Nicole Stricker, and Gisela Lanza. Designing an adaptive production control system using reinforcement learning. *Journal of Intelligent Manufacturing*, 32:866–876, 2021.

[10] Daniel Küpper, Christoph Sieben, Kristian Kuhlmann, Yew Heng Lim, and Justin Ahmad. Will flexible-cell manufacturing revolutionize carmaking? pages 1–16, 2018.

[11] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017.

[12] Ngoc Duy Nguyen, Thanh Nguyen, Hai Nguyen, Doug Creighton, and Saeid Nahavandi. Review, analysis and design of a comprehensive deep reinforcement learning framework. *ArXiv*, abs2002.11883. pages of journal unknown.

[13] W. Nickel. Autos am laufenden band - 100 jahre fließbandfertigung. `http://www.zeit.de/auto/2013-04/ford-fliessband-massenproduktion`, 2013. Last accessed: 12-01-2022.

[14] Siemens. Available features of the individual licenses. `https://docs.plm.automation.siemens.com/content/plant_sim_help/15/plant_sim_all_in_one_html/en_US/tecnomatix_plant_simulation_help/setting_up_and_starting/plant_simulation_license_types/available_features_of_the_individual_licenses.html`, 2019. Last accessed: 12-01-2022.

[15] Siemens. Plant simulation & throughput optimization. `https://www.plm.automation.siemens.com/global/en/products/manufacturing-planning/plant-simulation-throughput-optimization.html`, 2022. Last accessed: 12-01-2022.

[16] Thomas Simonini. On choosing a deep reinforcement learning library. `https://blog.dataiku.com/on-choosing-a-deep-reinforcement-learning-library`, 2020.

[17] SimPy. Overview. `https://simpy.readthedocs.io/en/latest/`, 2020. Last accessed: 12-01-2022.