Development of a resilient Reinforcement Learning-based decision algorithm for order scheduling

Entwicklung eines resilienten Reinforcement Learning-basierten Entscheidungsalgorithmus für das Order Scheduling

Master thesis by Fabio Serra Pereira

Date of submission: October 7, 2024

1. Review: Prof. Dr.-Ing. Matthias Weigold, Technical University of Darmstadt
2. Review: Prof. Dr.-Ing. Paulo Eigi Miyagi, University of São Paulo

TECHNISCHE UNIVERSITÄT DARMSTADT

PTW TU DARMSTADT

Institut für Produktionsmanagement, Technologie und Werkzeugmaschinen

Masterthesis
für
Fabio Serra Pereira | 2873099

TECHNISCHE UNIVERSITÄT DARMSTADT

TU DARMSTADT

Institut für Produktionsmanagement, Technologie und Werkzeugmaschinen
Fachbereich Maschinenbau
Otto-Berndt-Str. 2
64287 Darmstadt

Telefon: 06151 16-20080
Telefax: 06151 16-20087

Thema:
Entwicklung eines resilienten Reinforcement Learning-basierten Entscheidungsalgorithmus für das Order Scheduling

Topic:
Development of a resilient Reinforcement Learning-based decision algorithm for order scheduling

Within the context of Industry 4.0, which characterizes an interconnected environment within the industrial realm, Artificial Intelligence (AI) applications play an important role in facilitating machine-to-machine communication and automatizing even processes, that were previously reliant on human intervention. In particular, the field of Machine Learning (ML) within AI offers the potential to design more cost-effective and resource-efficient manufacturing processes. However, to fully exploit this potential, the ML approach must also achieve the required objectives (e.g., productivity, lead time) even in disruptive situations. This requires the ability to anticipate disturbances, recognize them early, react to them, and mitigate their effects. This ability, which allows for a rapid return to the desired state after a disruption occurs, is called resilience and is central to autonomous production processes.

This thesis aims to explore a Reinforcement Learning (RL) and a benchmark algorithm to develop decision strategies for order scheduling that enhance production system resilience during disruption scenarios. To accomplish this objective, two steps for algorithm selection will be realized: The first step is a literature review collecting different algorithms, which will differ by applicability and performance. Following, algorithms identified as suitable are applied in a use case and evaluated based on key performance indicators (KPIs) as the second step.

This thesis includes the following work packages:

- Literature review of Machine Learning concepts in production control, including RL.
- Literature review of resilient decision algorithms for order scheduling (ML-based, heuristic, etc.).
- Literature review of various KPIs for measuring resilience and established objectives (e.g. throughput) in production systems.
- Understanding of the available gathered data given by the context of order scheduling, and analysis of how the identified algorithms can be applied.
- Definition and implementation of disruption scenarios in a simulation environment.
- Development of a resilient RL-based decision algorithm.
- Benchmarking the RL algorithm against another decision logic (heuristic, etc.) in the defined disruption scenarios, using selected KPIs.
- Documentation and presentation of the findings.

Beginn: 08.04.2024
Umfang: 30 CP, 900 h
Betreuer: Leonie Meldt M.Sc.

Prof. Dr.-Ing. J. Metternich

Prof. Dr-Ing. P. Miyagi (University of São Paulo)

Thesis Statement pursuant to § 22 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Fabio Serra Pereira , have written the submitted thesis independently pursuant to § 22 paragraph 7 of APB TU Darmstadt without any outside support and using only the quoted literature and other sources. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I have clearly marked and separately listed in the text the literature used literally or in terms of content and all other sources I used for the preparation of this academic work. This also applies to sources or aids from the Internet.

This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

For a thesis of the Department of Architecture, the submitted electronic version corresponds to the presented model and the submitted architectural plans.

Datum / Date: October 7, 2024

Unterschrift/Signature:

Abstract

Development of a resilient Reinforcement Learning-based decision algorithm for order scheduling

This project examines a resilient Reinforcement Learning (RL) algorithm designed to manage disruptive scenarios within the order scheduling system of the FlowFactory Lab at the *Produktionsmanagement, Technologie und Werkzeugmaschinen* (PTW) of the Technical University of Darmstadt (TUDa). A simulation environment employing Petri net models was developed to facilitate the analysis of material flows. In the process of selecting an Artificial Intelligence (AI) method, a comprehensive literature review was undertaken to identify the most suitable algorithm, leading to the selection of Q-learning. Furthermore, Key Performance Indicators (KPIs) were identified, focusing on computational effort, Makespan (the time difference between the start of one machine and the end time of another), Mean Squared Error (MSE), and Mean Absolute Error (MAE) as metrics for evaluation. The performance of the RL algorithm was compared with that of simulations relying solely on Petri net modeling, referred to as the traditional method. The findings indicate that the simulation incorporating the RL algorithm showed improved performance and enhanced resilience to disruptive scenarios.

Keywords: Resilience, Reinforcement learning, Q-learning, Order Scheduling, Simulation-based optimization

Contents

Abbreviation Index

Abbreviation	Meaning
RL	Reinforcement Learning
AI	Artificial Intelligence
MSE	Mean Squared Error
MAE	Mean Absolute Error
KPI	Key Performance Index
MDP	Markov Decision Process
PTW	*Produktionsmanagement, Technologie und Werkzeugmaschinen*
TUDa	*Technische Universität Darmstadt*
DP	dynamic programming
UAV	unmanned aerial vehicle
ARP	action-replay process
ML	Machine Learning
DQN	Deep Q-Network
SARSA	State-Action-Reward-State-Action
HEFT	Heterogeneous Earliest Finish Time

Formula Symbol

General Hints

$\hat{X}$	Amplitude Value
$\bar{X}$	Mean
$\underline{X}$	Komplexer Phasor
X^*	Complex Phasor
M	Matrix

Latin Formula Symbols

Symbol	Unit	Description
T	min	Time difference given in minutes
P	adimensional	probabilities
P_P, P_T	s	production and transport times measured in seconds

Greek Formula Symbols

Symbol	Unit	Description
θ	s	Timestamp to define a certain date with time

List of Figures

List of Tables

1 Introduction

In the contemporary dynamic and multifaceted industrial environment, enterprises must augment the efficiency and resilience of their production operations. Incorporating resilience and machine learning (ML) within the framework of autonomous production control emerges as a promising strategy to fulfill these demands.

Resilience represents a pivotal attribute of a production system, typically correlated with its robustness or capacity to adapt in the face of disruptions. Beyond conventional methods to foster resilience within production systems, this endeavor seeks to devise a solution by applying Reinforcement Learning (RL) techniques. This approach entails simulating an industrial production chain, thereby facilitating the generation of self-automated decisions and enhanced resilience.

ML is a vast domain of artificial intelligence (AI) that has grown in popularity because of its simple recipes or algorithms that give programs the capability to learn patterns, behavior, models, and functions and use that information to make better decisions or actions in the future. ML can be classified as supervised learning, where a training data set is given and the agent learns how to predict output values of certain input targets; unsupervised learning, where an agent learns a certain structural organization of the input data or the relationship of the elements of the data set; and RL, where an agent is given certain rewards corresponding to the utility of an action or decision relative to the world model, with which the agent interacts. Neural networks and deep learning are the most prominent concepts in AI due to their capacity to find more efficient solutions than heuristic approaches. [AFI17]. Regarding the problem of order scheduling in distributed systems, the ML box will use RL algorithms to schedule the orders in the given production phase.

RL technology has shown numerous successes in solving complex human tasks in recent years. The advantage of using RL for solving complex problems lies in the possibility of discovering the best actions, even without knowing all the properties of the environment. [Ris23] Considering this, the RL tool is an achievable decision tool for this work's application. It offers fast decisions over the small decisions in the production chain, not regarding a considerable amount of data to its decision, instead of being consistent to its reward function.

The Markov Decision Process (MDP) models the environment for each decision, serving as the foundational framework for RL. This approach aids in delineating states and actions required by the algorithm, thereby facilitating the creative process. A more complex component to accurately model is the reward function. This function assigns a value to each action, with the magnitude of the reward guiding the algorithm's decisions by prioritizing actions considered to be of higher relevance.

The structure of this work is delineated across six chapters, each serving a distinct purpose within the overall project. Chapter 1 provides an introductory overview, familiarizing readers with the key topics addressed in this study. Chapter 2 lays the foundation by discussing the theoretical concepts crucial for comprehending the project's scope. Chapter 3 reviews relevant literature, highlighting significant criteria used in project guideline assessments. Chapter 4 details the development process of the project's solution, specifically the step-by-step creation of the RL algorithm. Chapter 5 evaluates the efficacy of the RL algorithm by comparing it with a traditional approach, which employs Petri Nets for simulation resolution.

Finally, Chapter 6 synthesizes the findings and contributions of this work, offering conclusions and potential avenues for future research.

1.1 Motivation

In the contemporary manufacturing landscape, ensuring competitiveness and efficient resource allocation necessitates the transcendence of traditional scheduling methodologies. These conventional approaches often falter in adapting to emergent challenges such as supply chain disruptions, thereby escalating operational expenditures.
In light of these obstacles, this initiative endeavors to formulate a resilient decision-making algorithm anchored in RL principles. The objective is to cultivate a more adaptable and robust framework for order scheduling. This endeavor aims to engineer a self-adjusting system proficient in navigating fluctuating conditions and disruptions, thereby enhancing production efficiency and fortifying overall system resilience.

1.2 Definition of Project

This project was developed regarding the following research question

1. Demonstrates the development of a RL algorithm performance compared to heuristic methods in order scheduling?

2. How to develop a RL algorithm that can handle occurrences and demonstrate resilience?

The project encompasses several critical activities, including examining ML concepts and a comprehensive literature review on resilient decision-making algorithms applicable to order scheduling. It also involves identifying relevant Key Performance Indicators (KPIs) for assessing resilience within production systems. The project aims to determine the algorithm's applicability, implement disruption scenarios within a simulation framework, and conduct a comparative analysis of the RL algorithm against alternative decision-making frameworks using the established KPIs.

2 Fundamentals

This project focuses on developing a resilient algorithm using RL to improve production time simulations in a production chain. This chapter will introduce key concepts to help readers better understand the project's ideas.

2.1 Value Stream

This section is based on a thorough literature review [Dec24].

2.1.1 Value Stream and Value Stream Risks

The value stream refers to the flow of values in a company and is defined by [SM23] as: "A value stream includes all activities to produce a finished product from raw materials and deliver it to the customer." In addition to the physical manufacturing process of the product, it also involves the necessary logistics and information flows in the company, as well as all other elements of value creation. The term comes from lean management, and an analysis of the value stream can be used to improve and streamline the value creation process. A classic analysis tool is value stream mapping, defined in DIN ISO 22468. This creates an overview of the activities that the value stream contains. A map of bottlenecks, waste, or other inefficiencies is created to develop an improved process mode.
A company's value stream is a complex system that must operate under its environment's influence. External factors such as wars and other crises, prices, shocks, or sanctions, as well as internal company influences, affect and can disrupt the value creation process. The value stream is therefore subject to various risks, which [SM23] divide into the process, demand, supply, control, and environmental risks based on [CP04]. The mentioned division will be explained further.
The environmental risk considers factors in the company's environment, such as legislation or the labor market. A pronounced shortage of skilled workers or government regulation such as restricting or prohibiting certain substances the company uses for production. To a certain extent, environmental risk influences the other four types of value stream risk.
Supply risk describes risks that arise from the supplier of the supply chain. These can be caused by embargoes, wars, or natural disasters that disrupt the supply chain and lead to a shortage of resources and usually an associated price increase. This can slow or even stop the company's production
Demand risk concerns the other side of the supply chain, which is influenced by the demand for products from different companies or consumers. Similar to supply risk, disruptions in the supply chain can affect the manufacturing company.
Control and process risk are internal risks that describe the failure of machines, networks, programs, or workers. This paper addresses these risks in particular because they describe the environment in which Production Planning and Control operate. Production planning and control also consider demand and supply risks, but these processes focus more on characterizing the supply chain.

2.1.2 Resilience of the value stream

Dealing with and preventing the risks mentioned above is essential for the process in a manufacturing company. The value stream method discussed above can be used for identification, while there are standardized procedures for managing risks, such as in DIN ISO 310000:2018:10. Regardless of the methodology, dealing with risks is to be understood as part of the resilience of a production system [SR22]. Resilience in the production environment does not have a standardized definition, which is why many different definitions of resilience can be found in the literature. Robustness, agility, or flexibility are often synonyms for resilience. The content, as follows, defines these terms.
Robustness refers to the ability to be stable in the face of external forces or changes and to service them without significant impairment. In the value stream context, this can mean that the system is little affected by risks that have occurred. Agility describes the ability of organizations or systems to adapt quickly to fluctuations or react flexibly to changes. About a manufacturing company, this would be conceivable with modular production facilities that can react quickly to changes in orders received, delivery times, or other situations. In normal language, resilience is usually used from a psychological perspective and is understood as resistance to external influences. According to [Len+23], resilience concerning production systems is the ability to maintain or quickly restore a stable state under the influence of stress or disturbances. The German Academy of Science and Engineering provides another definition with the "ability to anticipate and take into account actual or potential adverse events, to recognize their occurrence at an early stage and to prevent their occurrence, to mitigate their intensity, to reduce their damage, to react quickly and to recover from them and to adapt to them successfully and learn from them." [WS22]. Using this definition, [SM23] use the value stream risks mentioned above to draw a connection to a definition of resilience from the value stream perspective, which is:
Value stream resilience is the ability of a value stream to satisfy customer demand even of disruptions while meeting the set goals in the dimensions of flexibility, quality, productivity, and lead time by anticipating and taking disruptions into account, identifying them early, preventing them, and mitigating them. The more resilient a value stream is, the faster it will meet the set goals again after a disruption occurs and adapt to the new conditions. This ability can generally be referred to as adaptivity. In addition, [SM23] focuses on predicting disturbances intended to determine a system's resilience. This exceeds the scope of widely used definitions of resilience, which will be discussed in more detail in this work, as these mainly focus on the target variables of agility and robustness, which in some cases is used as a synonym for resilience. Rather than, the terms agility and robustness can be undestood as components of resilience [Muk+22]. The terms are, therefore, not synonymous with that; the terms agility and robustness can be understood as increasing the resilience of the value stream. The target variables of resilience can thus be divided into robustness, agility, and adaptability.
Since the production environment is part of the value stream, applying the definition already given is appropriate. Therefore, it corresponds to the definition and the objectives of resilience to which the contents of this work refer.

2.2 Production Planning and Control

The term Production Planning and Control refers to the area of direct production, which concerns, for example, machine times and maintenance, as well as surrounding areas such as incoming and outgoing orders, material flow, and quality control. Production Planning and Control is, therefore, a central means of ensuring the efficiency and functionality of at least part of the value stream to which this work will be devoted. Increasing efficiency in production is usually based on lean principles, which have their origins in a Toyota production system form the last century. Lean Production describes a continuously improved

system that aims to reduce waste and ensure a stable flow in production. Research in current literature often concerns the integration of lean principles of resilience in so-called job shoos, flow shops, or cell production. The structure of these production systems, approaches to control, and the autonomy of this process. The integration of data streams from the production process into digital models is discussed in the following subsection.[Dec24]

2.2.1 Production System

The structure of a production system is defined by the way in which the production lines are organized and how are controlled. There are various approaches, such as job shops, flow shops, or cell-based production, as well as distributed and centralized production control systems. These are briefly explained below.
A job shop is a type of production environment that resembles a workshop or station production. In contrast to classic mass production on an assembly line, as is often used in the automotive industry, there are many stations here that can operate flexibly and independently of one another. Products can pass through the job shop in any order [PU11]. This allows for a high degree of variance and a high degree of individualization of the manufactured products. Due to the high flexibility in the job shop, even small batch sizes can be produced economically. However, this production environment entails complex control and planning and is not well suited to large-scale production. The key here is the coordination of the production flow between the individual machines and the utilization of these.
A flow shop, linear sequence of processing stations [PU11]. All parts, therefore, follow the same path in the production facility, which makes the production environment less flexible but easier to coordinate and plan than a job shop. This enables larger series production and an optimized material flow for standardized products. The focus during planning is an even utilization of all stations is important so that no bottlenecks in the flow arise, which would cause production to back up at one station.
Cell production is characterized by the fact that a production facility is divided into autonomous cells. These cells are suitable for producing specific product families [Sin93] and can be quickly converted in order to switch agilely between similar products. The cells work in batch sizes economically. There are various organizational forms in which the flow between the cells, in particular, is an important metric for control.
When it comes to approaches to controlling production systems, a distinction is generally made between models in which data is evaluated centrally (Centralized Production Control) that pursue decentralized decision-making in many local production plants (Distributed or Decentralized Production Control). The advantages of Distributed Production Control compared to Centralized Production Control are the distribution of the failure risk as well as faster computing times, and the more individual design of local systems [DBW91]. These properties ensure that Distributed Production Control approaches to the Integration of ML into highly individualized, autonomous, and local production controls. However, centralized production control has the advantage of simplifying the making of globally optimized decisions [BN01]. In addition to the two approaches mentioned, there are other concepts that attempt to combine the advantages of both models. These will be discussed in more detail in the course of this paper.

2.2.2 Data Integration and digital models

The basis for the design of resilient and autonomous production systems is the integration of data, which can be analyzed classically using the value stream method explained in 2.1. The introduction of technologies from the fourth industrial revolution enables automated handling of data streams, which can be recorded by integrating a large number of sensors. The interweaving of the physical and virtual systems creates a cyber-physical system into which data streams can be integrated in real time. The data is often processed

in digital models such as the digital twin or the digital shadow. The digital twin is one of the most common models and is probably based on an idea that emerged at the beginning of the 21st century [Gri]. AI and cloud-based technologies are core components of the automated integration and processing of data in digital models for modern production systems.
Both the digital twin and the digital shadow are digital models of a physical process. Data from the process is imported in real-time into specially developed software. This results in a permanently up-to-date picture of the production process. By integrating many sensors in machines and other production facilities, a model of the real process can be created that is as accurate as possible. This means that machines, production sections, or, theoretically, entire factories and production networks are a digital image. They differ in that a digital shadow is only used as a form of representation, and only the digital twin enables automated data traffic back to the physical object [NJ22]. The data from a digital model derived from shadow or digital twin allows simulations to be carried out during the production process without touching the physical process, thus avoiding high effort and cost. For example, the resilience of a production process can be tested by simulating it in environments with induced disturbances. This simulation can also be used to generate data that allows the training of a model based on ML.

2.2.3 Autonomized Production Planning and Control

In addition to increasing efficiency, Production Planning and Control also aims to avert risks and dangers to production in order to ensure uninterrupted production. The control of the process must be designed to be robust against the value stream risks mentioned in 2.1 and must react agilely and adaptively to changing conditions, thus being resilient to its environment. There are various options for designing a resilient production facility, including the autonomization of production control. This includes, for example, the autonomous adjustment of maintenance intervals and machine times or autonomous decision support as an interface to the human operator. Autonomization itself helps to make the process more resilient by minimizing the human decision-making component and thus defusing possible sources of error. However, autonomization also presents risks and does not make the process ultimately resilient. Rather, it is important to find methods with which resilience can be integrated into autonomization.
The integration of maintenance measures is a central aspect of manufacturing companies. Machines and tools degrade over their service life and must be serviced or replaced. When it comes to preventive maintenance of machines, a distinction is made between time-based and condition-based maintenance, which is based on time intervals specified by the manufacturer or the checked condition of the system. Condition-based maintenance can maximize the maintenance intervals and thus reduce the number of maintenance operations measures that are a central aspect required. The aim is to carry out maintenance measures before there is a decrease in product quality or a tool failure. Every maintenance action carried out requires a stop in the production process. The planning of the measures can be supported by ML in order to enable the most cost-effective and automated integration into the production process. The assessment of the health status of a machine can also be automated using algorithms and sensors. This method is called predictive maintenance and has the prediction of the remaining useful life [SAS21], which is made by monitoring process parameters on the affected machine as a basis. The integration of such methods will be discussed in more detail in the course of this work.
In addition to the integration of maintenance, the actual planning of production plans plays a central role in the autonomization of the production process. The terms scheduling and dispatching describe the allocation of machines or workers in a production plan or the control and adjustment of the plan drawn up [And20]. Scheduling describes a rather longer term and dispatching, a rather short-term process. Heuristics, which are often in centralized production control approaches, are very suitable for the global optimization of production plans [QL21]. ML, absed approaches are also conceivable here, as they offer

greater flexbility and adaptability as well as the possibility of self-adaptation and could therefore be used especially in dynamic job shop environments[You+23].
ML-based approaches are particularly suitable for order dispatching, as in this case decisions can be made within a few seconds [Rol06]. The application takes place in the local production facilities, and it is therefore tied to a highly decentralized control. The technical implementation will be discussed in more detail in the course of the work.

2.3 Machine Learning

ML is an important branch in the field of AI. ML uses algorithms to develop various models that can be trained with data. In contrast to traditionally programmed software, ML-based models can permanently learn from current data and adapt without having to be explicitly reprogrammed. This property fits into the previously written components of the resilience of a system. For this to happen, a training process must first be carried out in which the model acquires its behavior. Data obtained from the production process can be used to train the model. Furthermore, data for training can be generated in simulations, which has the advantage that it is more freely accessible and is generated in adaptable environments.[Dec24]
ML algorithms include various approaches such as supervised, unsupervised, and reinforcement learning [And20]; most algorithms can be classified according to these groups. In general, ML-based models usually involve an agent and an environment. These can be classified as follows: "The term agent describes a computer system that is embedded in an environment and makes decisions" [And20]. In addition to the use of one agent in a system, there are also so-called multi-agent systems. Here, several cooperating or competing agents are used, which are particularly useful in complex systems.
In the case of supervised learning, a labeled dataset is used for training. Models based on this dataset are usually used for classification tasks in which the algorithm has to assign an input to a specific group [Vla17]. However, these datasets are complex to create and, therefore, rather costly.
Meanwhile, unsupervised learning uses datasets that do not have any information assigned. These are particularly well suited for grouping data and finding patterns in them. Among other things, these algorithms can be used to prepare datasets for use in unsupervised learning algorithms [MAY20].
RL is probably the most important type of learning for the use cases presented in this work. The focus is on the behavioral biology field of conditioning. The agent, which acts on a trial-and-error basis in its defined environment, is given positive or negative feedback for its behavior, which ultimately determines the behavior of the system [And20].
One of the best-known functions of ML-based programs is pattern recognition. Large amounts of data can be analyzed in a short time by MAS, and decisions can be described in 2.2. In the context of PPS, these decisions can be used to autonomously adapt the production process or serve as a decision-making aid for people in the production.
In the following section, further discussion will be presented about the application of ML using a RL algorithm in the production stream. Further in this chapter, it will be better understand the concepts of reinforcement learning and the Q-Learning, fundamentals concepts for Chapter 4. In Chapter 3 a literature review was presented, where some papers were reviewed due to select the best algorithm for the application.

2.4 Reinforcement Learning

The term RL, is given because there is a reinforcement of the agent's actions. By way of analogy, imagine a dog being trained in a park. Every time it correctly performs a trick, such as sitting, lying down, or rolling

on the ground, it gets a cookie. This reward reinforces the action, and the animal learns to repeat the act based on the order in which it is given. In this analogy, the dog is the agent, who, by performing the trick, a correct action, receives a cookie, which is the reward. In other words, the correct action is reciprocated and reinforced, employing a reward; hence, it is considered RL. Similarly, a hostile or incorrect action can be negatively rewarded as a punishment by the trainer [Ris23].
Important definitions must be presented as follows as defined:

- Agent: the entity that will interact with the environment; it is the decision maker. It can be a robot, an autonomous car, etc.
- Environment: the universe in which the agent will interact. The environment is the outside world; it comprises everything outside the agent. For example, it can be a maze, a house, a park, etc. It does not necessarily have to be a physical space.
- Action (a): behavior that the agent can perform. Most often, it is associated with a movement, for example, toward the north, toward the south, or picking up or dropping an object, and so on.
- State (s): conjuncture of the agent with the environment. For example, consider the case of a robot in a position in a given maze.
- Reward (r): retribution (positive or negative) that the agent gets when reaching the next state (s). For example, winning 10 points.
- Policy (Pi -): strategy of action that promotes state change in the expectation of obtaining the best result. In other words, it is the objective that you have for the training, what the agent will learn. It can be, for example, which trajectory should be followed most efficiently.
- Episode: a complete set of actions that ends when reaching a goal. For example, going from one point to another, going from the house to the castle, etc. Thinking that each movement is a single step, the set of steps to the final state is what is considered an episode.

It can be said that an agent learns through experience, trial, and error, or by receiving rewards in an environment. These rewards can be positive or negative. For example, a cookie would be a positive reward for a dog, while getting its attention or using a whistle with an annoying noise would be a negative reward.

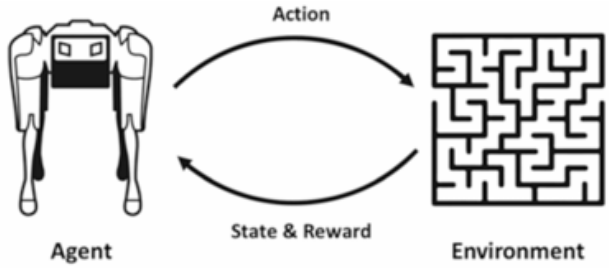

Figure 2.1: Framework of an agent's interaction with the environment [Ris23]

.

The goal, therefore, is to learn a policy that maximizes the accumulation of rewards. Here, it is not about getting immediate rewards but aiming for an accumulation of rewards at the end. The RL framework is composed of states, actions, and rewards [RA20], as shown in Figure 2.1.

In the framework, there is an agent that is in a certain initial state. In this state (s), the agent acts (a) on the environment, which promotes the change to the next state *(s')*, and it the receives a reward *(r)*.
In RL, the agent records the data of its interaction and learns from the consequences of its actions. The system increments the agent with positive or negative rewards. The machine learns the consequences of its actions and what to do to achieve the goals. The idea is that the machine can design its own strategies to achieve its ultimate goal, even if circumstances change. In RL, the agent learns to perform a task through interaction, or rather through experience, trial and error, cause and effect, or obtaining rewards from the environment.
When an agent initially enters an environment, it lacks knowledge of how the world operates. By interacting with the environment, the agent gathers data on the state and the actions it takes, and receives rewards, which can be considered as labels. With this data, the agent can make better-informed decisions and take improved actions.
The agent's goal is to maximize the total reward it receives. It looks for a series of actions that will result in the highest cumulative reward. By interacting with the environment, the agent learns its rules and, after training, can select a set of actions (policy) to achieve its objective.
The agent is not told which actions to take but instead must discover which actions bring the most benefits by trying them out. Thus, the agent learns from its own experience.
The agent's actions can influence not only the immediate reward but also the subsequent situation and, consequently, all subsequent rewards. RL is characterized by two factors: the search through trial and error and the presence of a delayed reward.
The world is constantly changing, and RL can excel in these dynamic environments. An embedded RL agent encounters and performs well in these new situations [Ris23].

2.5 Markov Decision Process: Concept Explanation

The first step in developing systems based on RL will be to model your problem as a MDP [Bel54]. To do this, it is will be understood the fundamentals of formulating and solving such problems. This section used as main literature [Ris23]

2.5.1 Markov Chain

The origins of RL are founded in Markov Chains. Markov Chains refer to probabilistic systems in which the future state depends on the present state and not on past states. This means that it does not matter which states the agent passed through until the present state; what matters are the relationships and probabilities of the next actions. Markov Chains allow the study of state transitions from the present to the future.
To obtain an example, let us say that in a given situation, there are only two possible states: "sunny" or "rainy". It is also assumed that some observations are performed and noticed that the sunny days were followed by sunshine 70'%' of the time, and therefore, 30'%' of them were rainy. It is also observed that on a rainy day, the next day was followed by sunshine 60'%' of the time and by rain 40'%' of the time. This change of states can be understood as the probability of transition from one state to another. Thus, it is possible to model the states and their transitions, as presented in Figure 2.2.
Notice that this problem in Figure 2.2 can be modeled as a graph (a representation of objects and their relationships). The graph can be read as follows: there is a 70'%' chance that a sunny day will be followed by another sunny day and a 30'%' chance that it will be followed by a rainy day; and from a rainy day, there is a 40'%' chance that it will be followed by a rainy day and a 60'%' chance that it will be followed by a sunny day

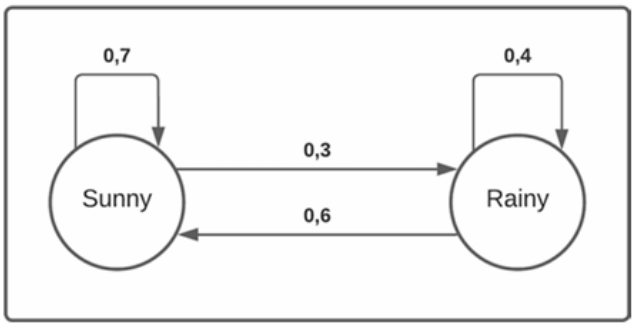

Figure 2.2: Graph of the transitions between sunny and rainy states [Ris23]

Table 2.1: Matrix of current and future states

		Future states	
		Sunny	Rainy
Present States	Sunny	0.7	0.3
	Rainy	0.6	0.4

The graph can be represented as a matrix, which can be organized by considering the rows as present states and the columns as future states, thus, the state transition matrix can be assembled in Matrix 2.1.

$$M = \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix} \tag{2.1}$$

This is a simplified demonstration. However, numerous states can be imagined in a complex system. For these more complex cases, matrix calculus can be used (which comes from linear algebra) to make the operation easier.
Therefore, this technique has its importance. Imagine that a certain day is a rainy Monday, and the probability of rain on Wednesday is wanted to know. Note that it can only have two possible paths, according to Table 2.2.
Therefore, for this case, the following calculus can be performed:

- 1°: Rainy-Sunny x Sunny-Rainy = 0.6 x 0.3 = 0.18
- 2°: Rainy-Rainy x Rainy-Rainy = 0.4 x 0.4 = 0.16

Table 2.2: Possible paths from the current state.

Paths	Monday	Tuesday	Wednesday
1°	Rainy	Sunny	Rainy
2°	Rainy	Rainy	Rainy

Figure 2.3: Resulting Matrix. [Ris23]

In other words, these two possible paths are the only achievables ones. Therefore, since it is raining on Monday, the probability of rain on Wednesday is the sum of these two probabilities, which is 0.18 + 0.16 = 0.34.
Note that if one transition is the matrix by itself, two transitions equal the matrix multiplied by the matrix itself, which results in another 2 × 2 matrix with the following results:

$$MxM = \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix} * \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.67 & 0.33 \\ 0.66 & 0.34 \end{bmatrix} \tag{2.2}$$

Additionally, if the initial state was a rainy Monday and the claim for the final state will be a rainy Wednesday (Figure 2.3), the result is obtained in the matrix.
Note also that if Monday is rainy and the probability of getting a sunny Wednesday must be known, the result is also calculated in the matrix, which is a probability of 0.66. Similarly, if Monday is sunny, the probability of having a sunny Wednesday is 0.67.
The matrix calculation comprises the operation of all possible paths, in this case, two, obtaining the resulting matrix.
Therefore, in this probabilistic system, the future state depends only on the present state and not on the past states, culminating in a Markov Chain.

2.5.2 Markov Decision Process

The MDP is an extension of the Markov Chain with the inclusion of a reward function. It provides a logic for modeling decision-making in situations where outcomes are partially random and partially controlled by a decision-marker.[Ris23]
RL can be expressed with the MDP as shown in Figure 2.4. Each environment is represented with a state that reflects what is happening in the environment. The RL agent takes actions in the environment that cause a change in the environment's current state, generating a new state, and receives a reward based on the results. The agent receives a positive reward for good actions and a negative reward for bad ones, which helps the agent evaluate the action performed in a given state and learn from experiences. [FK24]

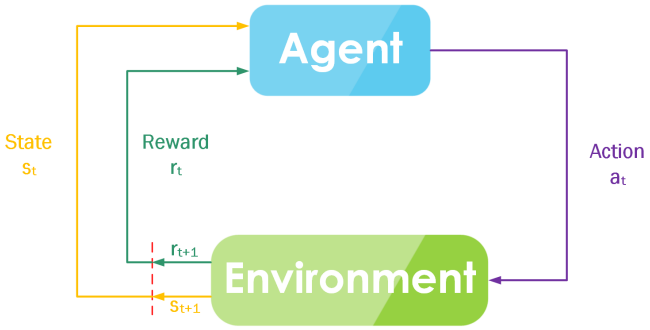

Figure 2.4: MDP [FK24]

As discussed earlier, most RL problems can be modeled as an MDP [RA20]. This is because MDP is a way of formulating processes in which the transitions between states are probabilities; observing what state the process is in is possible and intervening at decision points through actions. In turn, each action has a positive or negative reward, depending on the state of the process.
At each step of the process, the agent tries to choose the optimal actions from the many possibilities. An agent can try many different policies, although we are looking for the one that gives us the most utility.
As seen in Figure 2.4, four fundamental elements characterize an MDP: state, action, reward function, and transition function. A conventional description of an MDP is the tuple (S, A, R, P). These elements are detailed below.

1. *States (s)*: The agent can be in a set of states. The state contains the data to make decisions, determine rewards, and guide transitions.

2. *Action (a)*: A set of actions the agent can take to move from one state to another. The set contains all the possible actions that can be performed in the environment.

3. *Reward Function (r)*: The probability of obtaining a reward when an agent moves from one state to another. Indicates the direct reward for acting “a” in the state “s”. In some problems, the reward is not explicit, requiring the modeling of the reward to achieve the goal of solving the problem. Poor formulation of the reward can result in inappropriate

4. *Transition Function (t)*: The probability of an agent moving from one state to another by acting. It refers to the function that drives the dynamics of the system over time, moving the agent from state “s” to a new state, “s”. Usually, a transition involves both a stochastic and deterministic component. Transitions can be viewed from an external environment without being explicitly modeled behavior.

The goal of an MDP or RL problem is the expected cumulative total return. This is achieved by properly choosing a decision policy that will dictate how agents should act in every possible state. Since future states cannot be known in advance, we must now decide how to act in all possible situations to maximize the expected cumulative return on average. [Ris23]
According to [FK24], the RL agent learns from taking actions in the environment, which causes a change in the environment’s current state and generates a reward based on the action taken as expressed in the MDP. The probability of the transition to state "s"' with reward r is defined from taking action "a" in state "s" at time "t", for all $s' \in S, s \in S, r \in R, a \in A(s)$, as:

$$P\left(s', r \mid s, a\right) = \Pr\left\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right\} \tag{2.3}$$

The agent receives rewards for performing actions and uses them to measure the action’s success or failure. The Reward R can be expressed in different forms, as a function of the action R(a), or as a function of action State pairs R(a,s).
The agent’s objective is to maximize the expected summation of the discounted rewards that drive the agent to take the selected actions. The rewards are granted by adding all the rewards generated from executing an episode. The episode (trajectory) represents a finite number of actions and ends when the agent achieves a final state, for example, when a collision occurs in a simulated navigation environment. However, in some cases, the actions can be continuous and cannot be broken into episodes. The discounted reward, as shown in 2.4, uses a multiplier to γ the power of *k*, where $\gamma \in [0, 1]$. The value of *k* increases by one at each time step to emphasize the current reward and to reduce the impact of the future rewards, hence the term discounted reward.

$$G_t = E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right] \tag{2.4}$$

Emphasizing the current action's immediate reward and reducing the impact of future actions' rewards help the expected summation of discounted rewards to converge.

2.6 Reinforcement Learning Algorithms

For the purpose of this paper, a simple overview of the different RL Algorithms will be presented. However, the main focus of this paper is on the Q-Learning Algorithm, which was selected as the suitable algorithm for the purpose of the Chapter 4. A better understanding of how this algorithm was selected based on literature review, will be presented in the Chapter 3.
According to [FK24], while most RL algorithms use deep neural networks, different algorithms are suited for different environment types. RL algorithms are classified according to the number of the states and action types available in the environment into three main categories: 1) a limited number of states and discrete actions, 2) an unlimited number of states and discrete actions, and 3) an unlimited number of states and continuous actions. The three categories, together with algorithms belonging to those categories, are shown in Figure 2.5 and will be presented below.

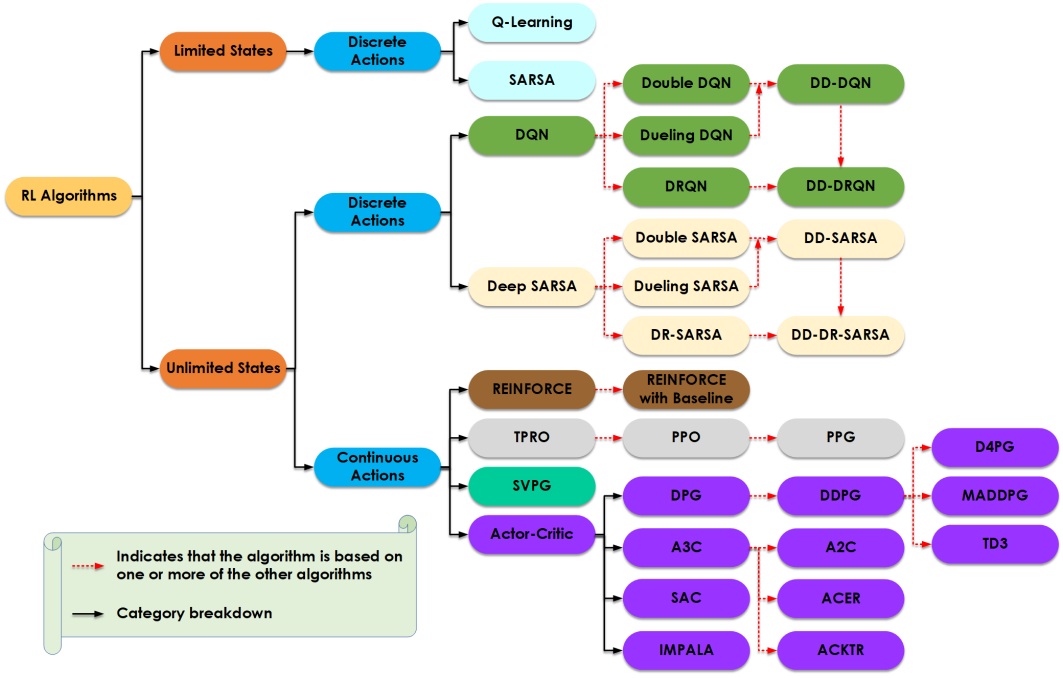

Figure 2.5: Reinforcement Algorithms classification based on the environment type [FK24]

The first classification concerns environments with limited states and discrete actions. These environments are relatively simple, where the agent can select from pre-defined actions and be in pre-defined known states. For example, when an agent is playing a tic-tac-toe game, the nine boxes represent the states,

and the agent can choose from two actions: X or O, and update the available states. The two commons algorithms to represent this classification are the Q-learning and State-Action-Reward-State-Action (SARSA) Q-learning algorithm is commonly used to solve problems in such environments. This algorithm finds the optimal policy in a MDP by maintaining a Q-Table table with all possible states and actions and iteratively updating the Q-values for each state-action pair using the Bellman equation until the Q-function converges to the optimal Q-Value, [JP92]. This algorithm will be presented in more detail in the next subsection.
The next classification is environments with unlimited States and Discrete Actions. In some environments, such as playing a complex game, the dates can be limitless. However, the agent's choice is limited to a finite set of actions. In such environments, the agent mainly consists of a Deep Neural Network, usually a Convolution Neural Network, responsible for processing and extracting features from the state of the environment and outputting the available actions. Different algorithms can be used with this environment type, such as Deep Q-Networks (DQN), Deep SARSA, and their variants.[JP92]
The last Classification is the environments with unlimited states and continuous actions. Although discrete actions are sufficient to move a car or UAV (unmanned aerial vehicle) in a virtual environment, such actions do not provide a realistic object movement in real-life scenarios. Continuous actions describe the quantity of movement in different directions where the agent does not choose from a list of predefined actions. For example, a realistic UAV movement specifies the quantity of required change in roll, pitch, yaw, and throttle values to navigate the environment while avoiding obstacles, rather than moving UAV using one step in predefined directions: up, down, left, right, and forward.[JP92]

2.6.1 Q-Learning

Q-learning is a form of model-free RL. It can also be viewed as a method of asynchronous dynamic programming (DP). It provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains. [Wat89] Learning proceeds similarly to [Sut84] method of temporal differences: an agent tries an action at a particular state, and evaluates its consequences in terms of the immediate reward or penalty it receives and its estimate of the value of the state to which it is taken. By trying all actions in all states repeatedly, it learns which are best overall, judged by long-term discounted reward. Q-learning is a primitive [Wat89] form of learning, but, as such, it can operate as the basis of far more sophisticated devices.
The following algorithm's explanation used the [JP92] as main source of literature.

2.6.1.1 The task for Q-learning

Consider a computational agent moving around some discrete, finite world, choosing one from a finite collection of actions at every time step. The world constitutes a controlled Markov process with the agent as a controller. At step n, the agent is equipped to register the state $x_n(\in X)$ of the world and can choose its action $a_n(\in Q)^1$ accordingly. The agent receives a probabilistic reward r_n, whose mean value $R_{x_n}(a_n)$ depends only on the state and action, and the state of the world changes probabilistically to y_n according to the law:

$$\text{Prob}\left[y_n = y \mid x_n, a_n\right] = P_{x_n y}\left[a_n\right] \tag{2.5}$$

The task facing the agent is that of determining an optimal policy, one that maximizes total discounted expected reward. By discounted reward, we mean that rewards received *s* steps hence are worth less than rewards received now, by a factor of $\gamma^s (0 < \gamma < 1)$. Under a policy π, the value of state *x* is

$$V^{\pi}(x) \equiv \mathcal{R}_x(\pi(x)) + \gamma \sum_y P_{xy}[\pi(x)]V^{\pi}(y) \tag{2.6}$$

Because the agent expects to receive $R_x(\pi(x))$ immediately for acting π recommends, and then moves to a state that is 'worth' $V^{\pi}(y)$ to it, with probability $P_{xy}[\pi(x)]$. The theory of DP [RS62] assures us that there is at least one optimal stationary policy π^* which is such that

$$V^*(x) \equiv V^{\pi^*}(x) = \max_a \left\{ R_x(a) + \gamma \sum_y P_{xy}[a]V^{\pi^*}(y) \right\} \tag{2.7}$$

is as well as an agent can do from state *x*. Although this might look circular, it is actually well-defined and DP provides several methods for calculating V^* and one π^*, assuming that $R_x(a)$ and $P_{xy}[a]$ are known. The task facing a Q learner is that of determining a pi^* without initially knowing these values. There are traditional methods (e.g. [MKH88]) for learning $R_x(a)$ and $P_{xy}[a]$ while concurrently performing DP, but any assumption of certainty equivalence, i.e., calculating actions as if the current model were accurate, costs dearly in the early stages of learning [AS90]. According to [Wat89], classes Q-learning as incremental DP because of the step-by-step manner in which it determines the optimal policy.
For a policy π, define Q values (or action-values) as:

$$\mathcal{Q}^{\pi}(x,a) = R_x(a) + \gamma \sum_y P_{xy}[\pi(x)]V^{\pi}(y). \tag{2.8}$$

In other words, the Q value is the expected discounted reward for executing action a at state x and following policy π thereafter. The object in Q-learning is to estimate the Q values for an optimal policy. For convenience, define these as $\mathcal{Q}^*(x,a) \equiv \mathcal{Q}^{\pi^*}(x,a), \forall x,a$. It is straightforward to show that $V^*(x) = max_a Q^*(x,a)$ and that if a^* is an action at which the maximum is attained, then an optimal policy can be formed as $\pi^*(x) \equiv a^*$. Herein lies the utility of the Q values if an agent can learn them, it can easily decide what it is optimal to do. Although there may be more than one optimal policy or a^*, the Q^* values are unique.
In Q-learning, the agent's experience consists of a sequence of distinct stages or *episodes*. In the n^{th} episode, the agent:

- observes its current state x_n,
- selects and acts a_n,
- observes the subsequent state y_n,
- receives an immediate payoff r_n, and
- adjusts its Q_{n-1} values using a learning factor α_n, according to:

$$Q_n(x,a) = \begin{cases} (1-\alpha_n)\,\mathcal{Q}_{n-1}(x,a) + \alpha_n\left[r_n + \gamma V_{n-1}(y_n)\right] & \text{if } x = x_n \text{ and } a = a_n \\ Q_{n-1}(x,a) \end{cases} \tag{2.9}$$

where

$$V_{n-1}(y) \equiv \max_b \{Q_{n-1}(y,b)\} \tag{2.10}$$

is the best the agent thinks it can do from state y. Of course, in the early stages of learning, the Q values may not accurately reflect the policy they implicitly define (the maximizing actions in Equation 2.10). The initial Q values, $Q_0(x,a)$, for all states and actions are assumed given.
Note that this description assumes a look-up table representation for the $Q_n(x,a)$. According to [Wat89] shows that Q-learning may not converge correctly for other representations.
The most important condition implicit in the convergence theorem given below is that the sequence of episodes that forms the basis of learning must include an infinite number of episodes for each starting state and action. This may be considered a strong condition on the way states and actions are selected - however, under the stochastic conditions of the theorem, no method could be guaranteed to find an optimal policy under weaker conditions. Note, however, that the episodes need not form a continuous sequence - that is, the y of one episode need not be the x of the next episode.
The following theorem defines a set of conditions under which $Q_n(x,a) \to Q^*(x,a)$ as $n \to \infty$. Define $n^i(x,a)$ as the index of the i^{th} time that action a is tried in state x.

2.6.1.2 Theorem

Given bounded reward $|r_n| \leq R$, learning rates $0 \leq \alpha_n < 1$, and

$$\sum_{i=1}^{\infty} \alpha_{n^i(x,a)} = \infty, \sum_{i=1}^{\infty} \left[\alpha_{n^i(x,a)}\right]^2 < \infty, \forall x, a \tag{2.11}$$

then $Q_n(x,a) \to Q^*(x,a)$ as $n \to \infty, , a$, with probability 1.

2.6.1.3 The convergence proof

The key to the convergence is an artificially controlled Markov process called the *action-replay process* (ARP), which is constructed from the episode sequence and the learning sequence α_n.
A formal description of the ARP is given in the appendix, but the easiest way to think of it is in terms of a card game. Imagine each episode $\langle x_t, a_t, y_t, r_t, \alpha_t \rangle$ written on a card. All the cards together form an infinite deck, with the first episode card next to the bottom and stretching infinitely upwards in order. The bottom card (numbered 0) has the agent's initial values $Q_0(x,a)$ written on it for all pairs of x and a. A state of the ARP, $\langle x, n \rangle$, consists of a card number (or *level*) n, together with a state x from the real process. The actions permitted in the ARP are the same as those in the real process.
The next stage of the ARP, given current state $\langle x, n \rangle$ and action a, is determined as follows. First, all the cards for episodes later than n are eliminated, leaving just a finite deck. Cards are then removed one at a time from the top of this deck and examined until one is found whose starting state and action match x and a, say at episode t. Then a biased coin is flipped, with probability α_t of coming out heads and $1 - \alpha_t$ of tails. If the coin turns up heads, the episode recorded on this card is replayed, a process described below; if the coin turns up tails, this card is thrown away, and the search continues for another card matching x and a. If the bottom card is reached, the game stops in a special, absorbing state and provides the reward written on this card for x, a, namely $Q_n(x,a)$.
Replaying the episode on card t consists of emitting the reward, r_i, written on the card, and then moving to the next state $\langle y_t, t-1 \rangle$ in the ARP, where y_t, is the state to which the real process went on that episode.

Card t itself is thrown away. The next state transition of the ARP will be taken based on just the remaining deck.
The above completely specifies how state transitions and rewards are determined in the ARP. Define $P^{\text{ARP}}_{\langle x,n\rangle,(y,m\rangle}[a]$ and $R^{(n)}_x(a)$ as the transition-probability matrices and expected rewards of the ARP. Also define:

$$P^{(n)}_{xy}[a] = \sum_{m=1}^{n-1} P^{\text{ARP}}_{\langle x,n\rangle,\langle y,m\rangle}[a] \tag{2.12}$$

As the probabilities that, for each x, n and a, executing action a at state $\langle x, n\rangle$ in the ARP leads to state y of the real process at some lower level in the deck.
As defined above, the ARP is as much a controlled Markov Process as is the real process. Therefore, one can consider sequences of states and controls and optimal discounted Q^* values for the ARP^2. Note that episode cards are only removed from the deck during such a sequence and are never replaced. Therefore, the bottom card will always be reached after a finite number of actions.

2.7 Petri Nets

Petri nets are a graphical and mathematical modeling tool applicable to many systems. They are a promising tool for describing and studying information processing systems characterized as concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic. As a graphical tool, Petri nets can be used as a visual communication aid similar to flow charts, block diagrams, and networks. In addition, tokens are used in these nets to simulate systems' dynamic and concurrent activities. As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems. Both practitioners and theoreticians can use Petri nets. Thus, they provide a powerful medium of communication between them; their models are more methodical, and theoreticians can learn from practitioners how to make their models more realistic. [Mur89]
According to [Wan12],a Petri net is a particular kind of bipartite-directed graph populated by four types of objects. These objects are *places, transitions, directed arcs* and *tokens*. Directed arcs connect places to transitions or transitions to places. In its simplest form, a Petri net can be represented by a transition together with an input and output place. This elementary net may be used to represent a data processing event, its input data, and output data, respectively, in a data processing system. To study the dynamic behavior of a Petri net-modeled system regarding its states and state changes, each place may hold either no positive number of tokens or no positive number of tokens. Tokens are a primitive concept for Petri nets, places, and transitions. For instance, the presence or absence of a token in a place may indicate whether a condition associated with this place is true or false.

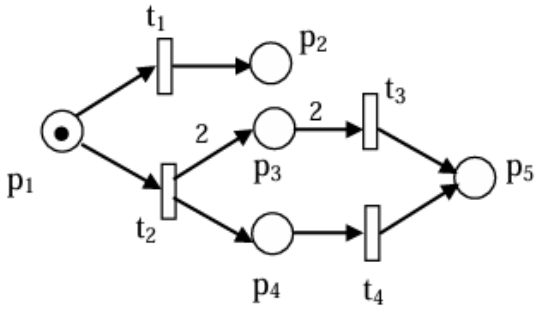

Figure 2.6: A Petri net with 5 places and 4 transitions. [Wan12]

Most theoretical work on Petri nets is based on the formal definition. However, a graphical representation of a Petri net is much more useful for illustrating the concepts of Petri net theory. A Petri net graph is depicted as a bipartite-directed multigraph. Corresponding to the definition of Petri nets, a Petri net graph has two types of nodes: a circle represents a place, and a bar or box represents a transition. Directed arcs (arrows) connect places and transitions, with some arcs directed from places to transitions and others directed from transitions to places.
The Petri net shown in Figure 2.6 is composed of five places, namely p_1, p_2, p_3, p_4 and p_5, and four transitions, namely t_1, t_2, t_3 and t_4. p_1 is an input place to t_1 and t_2 because one directed arc is contacting p_1 to t_1 and another is contacting p_1 to t_2. p_3 and p_4 are both output places to t_2 because there is an arc from t_2 to p_3 and an arc from t_2 to p_4. p_1 has one token, while all other places have 0 tokens. A directed arc can have a weight labeled to the arc in the graphic representation. The weight of the arc from t_2 to p_3 and arc from p_3 to t_3 is 2. The weight of all other arcs is 1, by default.
The execution of a Petri net is controlled by the number and distribution of tokens in the petri net. By changing the distribution of tokens in places that may reflect the occurrence of events or execution of operations, for instance, one can study the dynamic behavior of the modeled system. A petri net is executed by firing transitions. However, only an enabled transition can fire.
A transition t is said to be enabled if each input place p of t contains at least the number of tokens equal to the weight of the directed arc connecting p to t. For example, both transitions t_1 and t_2 are enabled in the Petri net of Figure 2.6. The firing of an enabled transition t removes from each input place p the number of tokens equal to the weight of the arc from p to t, and deposits in each output place p the number of tokens equal to the weight of the arc from t to p. Figure 2.7 shows the new token distribution of the Petri net of Figure 2.6 after firing t_2.

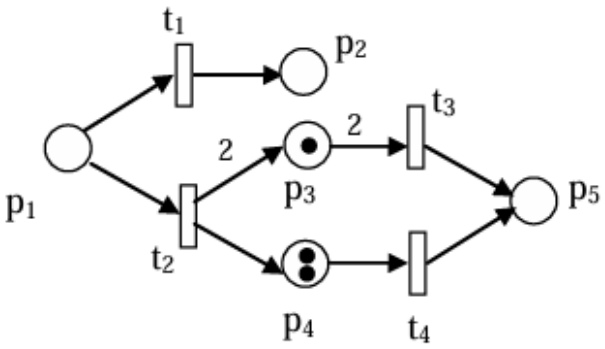

Figure 2.7: Firing of transition t_2.[Wan12]

Places are passive elements, while transitions are active elements. In a practical system model, places represent buffers, channels, geographical locations, conditions, or states, while transitions represent events, computation, transformation, or transportation. Tokens represent objects (e.g., humans, goods, and machines), information, conditions, or states of objects. A system state is specified by token distribution in places, called a *marking* in Petri net language. State transitions leading from one state to another are realized through transition firing.
Notice that since only enabled transitions can fire, the number of tokens in each place remains non-negative when a transition is fired. Firing a transition can never try to remove a token that is not there.
A transition without any input place is called a *source transition*, and one without any output place is called a *sink transition*. Note that a source transition is unconditionally enabled but doesn't produce tokens.
A pair of a place p and a transition t is called a self-loop if p is both an input and output place of t. A Petri net is said to be *pure* if it has no self-loops.
The Petri net was used as the primary model for the traditional simulation method, serving as a benchmark algorithm for simulation to determine if the RL Algorithm performs better than a simple method in which each order is fired in a sequential production line. This was crucial for identifying areas where a decision algorithm could be applied to make a difference in the production chain.

2.8 Errors

Before the conclusion of this chapter, the concept definitions of the Mean Squared Error (MSE) and Mean Absolute Error (MAE) will be presented. These important errors were used based on the output analysis of the method comparing the traditional method (benchmark method developed using the Petri Nets concept, deeply profound in Chapter 4) and the AI Method, which used the Q-learning algorithm, to automate some self-decision-making parts. The MSE and MAE were considered as parameters after a literature review, presented in Chapter 3.

2.8.1 Mean Square error

According to [Naj+23], the MSE is calculated to ensure that the original and the predicted methods are in variations or not. The lesser the MSE between two methods, the better the predicted method; it is defined as

$$\mathrm{MSE}\,(q_1, q_2) = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \left(q_1(i,j) - q_2(i,j)\right)^2 \tag{2.13}$$

where $q_1(i,j)$ and $q_2(i,j)$ indicate the values of method 1 (original) and method 2 (predicted), respectively.

2.8.2 Mean Absolute error

According to [ARA22], the MAE characterizes the alteration among the original and predictable values and is mined as the dataset's total alteration mean.

$$\mathrm{MAE} = \frac{1}{n} \sum_{i=1}^{n} \left| Y_i - \hat{Y}_i \right| \tag{2.14}$$

2.9 Cost Function

According to [Col12], cost fuctions are closely related to production functions. In fact the cost function is the solution to the following cost minimization problem:

$$[H] \qquad \begin{aligned} C(Q,w,r) &= \min_{K,L} rK + wL \\ &\text{s.t.} \\ Q &= \exp(\omega) K^{\beta} L^{\alpha} \end{aligned} \tag{2.15}$$

Under this assumption of a Cobb-Douglas production function, the Cost function has the following form:

$$C(Q,w,r) = \bar{\omega} + \frac{\alpha}{\alpha+\beta} w_{it} + \frac{\beta}{\alpha+\beta} r_{it} + \frac{1}{\alpha+\beta} q_{it} - \frac{1}{\alpha+\beta} \left(\bar{\omega} - \omega_{it}\right) \tag{2.16}$$

So notice that the original unobserved productivity term ω_{it} is still in the cost equation. Moreover, more productive firms are likely to produce more output; thus, q_{it} and ω_{it} are negatively correlated! This would lead us to conclude that firms have decreasing returns to scale since more prominent firms have lower unit costs. So some instructions are required.
This function was adapted to the variables available to develop the AI algorithm for this work. Therefore, the cost function could be modeled as follows:

$$C = C_{production} + C_{transport} + C_{storage} \tag{2.17}$$

Where the $C_{production}$ could be defined as:

$$C_{produtction} = \frac{\sum T_{P,i}}{T_{P,Total}} \tag{2.18}$$

In the provided context, P_i denotes the production time associated with a machine, where the subscript i refers to the specific machine in question. The notation P_{Total} is used to express the aggregate of production times for all machines considered within a given calculation. To illustrate, consider an instance involving three machines, each with distinct production times. If the objective is to compute the cost function for two selected machines, the term i-element would encompass the combined production times of these two machines, whereas P_{Total} would represent the cumulative production time across all three machines. Similarly, the concept of $C_{transport}$ is introduced in the Equation 2.19, adhering to the same logic established for production time calculation.

$$C_{transport} = \frac{\sum T_{T,i}}{T_{T,Total}} \tag{2.19}$$

For calculating $C_{storage}$, a distinct methodology is applied due to the imposition of a storage capacity limit, which defines the maximum number of components that can be stored. This is contrasted against the presently available quantity of components. The Equation 2.20 illustrates this case.

$$C_{storages} = \frac{S_{status}}{S_{Maximum}} \tag{2.20}$$

The term S_{status} indicates the currently available components in the inventory, and S_{Maximum} indicates its maximum capacity. The revised equations are essential for elucidating the reward function employed in the Q-Learning algorithm within the RL components of the project, with a more comprehensive explanation provided in Chapter 4.

3 Literature Review

This chapter will discuss the main methods for selecting the best RL algorithm and KPIs for the methods comparison. It begins with a brief overview of the papers used as references for this study and a discussion of why this paper was deemed necessary for the project. 15 papers have been chosen to analyze, which was considered a sufficient number for this work. The study influenced this decision [Dec24], which conducted a systematic literature review analyzing 26 papers.
The database used for this survey was ScienceDirect, and the main keywords used to select the papers were "RL Algorithm" and "Manufacturing schedule."
The project's initial phase will involve a detailed exposition of the V-Model methodology employed in its development process. This methodology delineates the systematic steps undertaken to create a robust RL algorithm.

3.1 Methodology: V-Model

The V-model lifecycle was introduced as a guideline for software development processes. The V-Model aims to improve both the efficiency of software development and the software's reliability. The V-Model offers a systematic roadmap from project initiation to product phase-out. The V-Model also defines the relationship between the development and test activities; it implements verification of each phase of the development process rather than testing at the end of the project. The V-model methodology is illustrated in Figure 3.1. [Mus+17]

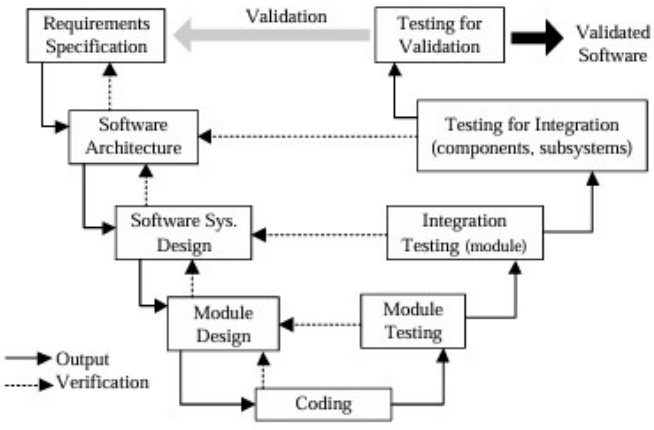

Figure 3.1: V-model software safety integrity and development lifecycle. [Mus+17]

Before initializing a software development process according to the V-model, a software planning phase has to be realized, wherein a software quality assurance plan, software verification, and software validation plans, and a software maintenance plan are fully defined. Later, the software requirements should be

determined by the customer and the stakeholders. Using the selected software architectures (including modeling methods), the designers develop software modules. Each phase is verified immediately after completion. Note that the left side of the V-model in Figure 3.1 represents the decomposition of the problem from the business world to the technical world [Rat11]. After the coding phase, the V-model's right side denotes the developed software's testing phase.
The advantages of using the V-Model are that it facilitates greater control due to the standardization of products in the process, cost estimation is relatively easy due to the repeatability of the process, each phase has specific products, greater likelihood of success because of the early development of test plans and documentation before coding, and, finally, provides a simple roadmap for the software development process.[Mus+17]
In the context of this project, the chosen methodology was implemented from the outset. Precisely, Chapters 2 and 3.2 delineate the phase of requirements specification. This chapter expounds on three critical phases: software architecture, software system design, and module design. Concurrently, coding and testing activities were pursued throughout the project's development, with the comprehensive codebase being documented in the appendix. Notably, integration was deemed unnecessary for the objectives of this project. Consequently, Chapter 5.1 is devoted to a comparative analysis of performance between the traditional methodology, employing Petri Net, and the RL algorithm, as gleaned from the testing phase.

3.2 Literature Overview

The section outlines the research papers that were instrumental in selecting the suitable algorithm for Chapter 4. It provides a concise summary of each paper, accompanied by a discussion on its relevance and the rationale behind its selection.
The paper [Sum+24] focuses on production scheduling using RL, explicitly addressing the problem of schedule nervousness in the manufacturing process. This research uses RL for production rescheduling, considering "schedule nervousness"-the disruptions caused by frequent changes in a manufacturing schedule. It aims to develop an online rescheduling agent using RL to minimize nervousness and maximize the stability of obtained schedules while guaranteeing flexibility and scalability. It proposes an explorative RL model using the Proximal Policy Optimization algorithm. This data-driven approach enables near-optimal scheduling decisions in real-time, hence finding value in large-scale industrial applications where traditional mathematical programming methods fall short due to computation costs.
The RL model was then compared to conventional approaches to mathematical programming regarding computational time and cost efficiency. Other RL methods that were considered include A2C and IMPALA, but it has been shown that Proximal Policy Optimization is far more stable and efficient. The problem Statement is a production-scheduling problem where different tasks must be assigned to various parallel machines, given task timing, machine assignments, and process costs. The generation challenge here is to generate alternative schedules that minimize schedule nervousness- disruptions- and adjust dynamically to real-world changes in real-time. The comparison parameters were defined as computational efficiency (How fast can a revised schedule be generated in response to changes?) Solution quality (Closeness of the generated schedules to the optimum solution), Schedule nervousness (this stands for the impact of refreshes within the schedule on the entire stability of the process), and Scalability (this is viewed as an ability to handle scheduling problems that are larger and more complex). The paper concludes that the RL model performed better in computational time for generating near-optimal solutions in practical applications. Unlike traditional optimization techniques, it was proper and efficient for handling huge scheduling problems with no significant computational burdens.

This document provides critical insights pertinent to the domain of manufacturing scheduling, emphasizing the project's objectives concerning optimizing and augmentation production flexibility. It conducts a comparative analysis of several algorithms, advocating for this comparison as a benchmark against alternative methodologies. A potential limitation of this document is its focus on a real-time algorithm, which diverges from the solution proposed in Chapter 4.

The paper [XJR12] studies the application of RL to the joint pricing-lead-time-scheduling decisions in the make-to-order system. The study focuses on challenges such as deciding whether to accept or reject orders, setting appropriate prices, and determining lead times to influence demand. The objective is to maximize the firm's long-run expected profits under stochastic demand conditions while considering production capacity constraints and the trade-offs between price, lead time, and capacity. The problem is formulated as a Semi-Markov Decision Problem, and the proposed method applies RL to a particular Q-learning algorithm. This approach lays down an RL strategy that deals with dynamic pricing optimization in lead-time decisions and scheduling. The performance of Q-learning is benchmarked against two policies: the first-come-first-serve policy and the threshold heuristic policy (a decision rule that evokes the concept of priority versus orders that have surpassed the threshold amount of profitability).

The main challenge is to simultaneously arrive at decisions on pricing, lead time, and scheduling in a resource-constrained environment. Under the conditions of stochastic order arrivals, where customer preference for price versus lead time can be heterogeneous, the manufacturer has to determine which orders to accept, at what cost, and at what production schedule will meet demands. The main parameters used for comparison were the Order acceptance rate (the share of the accepted order within the total orders), quantity acceptance rate (the percent of the total amount ordered accepted compared with the amount ordered), average reward (The average reward that, in the long run, would be generated by adopting the selected policy) and performance leading (The performance is evaluated, by simulation experiments, for three different scenarios: one related to a baseline demand, one regarding a variation in demand, and one related to capacity constraints).

This paper concludes that Q-learning is superior the heuristic algorithm since it maximizes long-term rewards, even though Q-learning always has lower order and quantity acceptance rates. It is shown that RL-based solutions can adaptively respond to temporal changes in environmental conditions such as demand and capacity variation and learn practical strategies of order acceptance and scheduling over time. This paper has shown its importance because it presents an application of Q-learning, the selected algorithm, after the literature revision. It is essential to mention how the paper compared the Q-learning performance and the benchmark algorithm presented to inspire what was developed in Chapter 4.

According to [JBD21], which explores the permutation flow shop scheduling problem in a manufacturing setting involving multiple production lines and demand plants. The study aims to present a novel RL approach to optimize scheduling in a flow shop scheduling problem with multiple production lines. The goal is to minimize the makespan (total completion time) while managing demand plants across multiple production lines. The RL method used in the study is based on Proximal Policy Optimization. This approach generates job sequences iteratively by simulating production environments and assigning rewards to minimize idle times and machine synchronization delays.

The problem focuses on the flow shop scheduling problem, where jobs must be processed in a fixed sequence on multiple machines, and multiple production lines operate in parallel. Each line produces sub-parts of the final products assembled at a synchronization machine. Varying demand plans increase the complexity of the problem. The RL approach is compared with other methods, such as Mixed-Integer Linear Programming, heuristics, and iterated greed. The parameters for comparison used were the makespan (total completion time for all jobs), computational time (time required to generate solutions), and performance under different demand plans (the algorithms are tested on balanced and imbalanced demand distribution). The RL-based method performs better than conventional methods, especially under short cutoff times. The RL

approach performs comparably to existing methods for medium and long cutoff times but outperforms them in terms of problems with imbalanced demand plans.
This paper illustrated the conceptualization of a RL algorithm employing a multi-agent system to optimize several production lines. The significance of this work is underscored by its potential to pioneer a solution that harnesses multiple agents. Although the development of such a solution was not actualized due to its inherent complexity, its application's feasibility was deliberated within this study's scope.
In [AFI17], it investigates a novel approach to task scheduling in heterogeneous distributed systems using RL. The study aims to solve the task scheduling problem in distributed systems by employing an RL algorithm. The focus is on improving execution time in systems characterized by distributed and heterogeneous nodes, considering the dependencies between tasks organized in a directed acyclic graph. The primary algorithm used in the study is Q-learning, a well-known RL method. Additionally, SARSA is introduced as a comparison technique. These algorithms aim to optimize the scheduling of tasks in distributed systems by learning from the environment and maximizing long-term rewards.
The key problem is scheduling concurrent tasks with dependencies across multiple heterogeneous nodes in a distributed system. The challenge lies in minimizing the overall execution time of the tasks by efficiently allocating tasks to nodes while considering the communication delays and resource limitations inherent in such systems. In addition to Q-learning, the study compares the performance of SARSA and the HEFT (Heterogeneous Earliest Finish Time) heuristic algorithm. These algorithms serve as benchmarks to evaluate the efficiency and adaptability of the RL approach. The parameters used for comparison were the execution time (total time required to complete all tasks), cumulative reward (a measure used in RL to evaluate the effectiveness of scheduling policy), number of nodes (the performance is evaluated with clusters of 2, 4 and 8 nodes to test scalability) and task dependencies (the impact of inter-task dependencies on the scheduling efficiency is also analyzed.
The work concluded that the results demonstrate that RL-based methods, specifically Q-learning and SARSA, can outperform traditional heuristics like HEFT in finding better task-scheduling solutions over time. However, the study also highlights challenges in scaling these methods to larger clusters due to the increased complexity of the state space.
This paper proposes an additional application for the Q-learning algorithm, which is vital for this study's objectives. Furthermore, an essential facet of this research is assessing the algorithm's performance in complex scenarios. It reveals that the Q-learning algorithm demonstrates commendable efficiency in environments characterized by non-complex MDP.
In [SGI24], the paper addresses the problem of airline dynamic pricing with patient customers using deep exploration-based RL. The study examines dynamic pricing in the airline industry, particularly focusing on non-myopic or patient customers who do not immediately purchase tickets but wait for favorable prices. The objective is to develop a dynamic pricing strategy using RL to maximize airline revenues by better understanding and accounting for this customer behavior. The primary algorithm used is a DQN with RL techniques. Additionally, a bootstrapped DQN is explored for its ability to improve exploration in environments with sparse rewards. These algorithms help learn pricing policies dynamically, responding to changes in customer behavior over time. The key problem addressed is the challenge of maximizing airline revenue in the presence of patient customers. These customers tend to wait for lower prices and use tools like price tracking and flight search engines, making it difficult for airlines to predict when they will make a purchase. This study introduces a novel MDP incorporating the history of offered prices as part of the state, allowing for more realistic customer behavior modeling.
The DQN and bootstrapped DQN algorithms were compared to traditional dynamic pricing methods. The paper mentions the HEFT heuristic as a baseline, although this is mainly a heuristic for task scheduling and not directly related to pricing. The study explores the performance of different RL techniques and compares them with established pricing models used in the airline industry. The parameters for comparison used

were the total revenue (the primary measure of success is the airline's total revenue over a fixed selling period), action history (the importance of considering past pricing actions in state variables is emphasized, as it allows for better customer modeling), exploration efficiency (RL algorithms are evaluated based on how effectively they explore the pricing policy space to find near-optimal solution) and performance across different demand scenarios (the methods are tested under both stationary and non-stationary demand to reflect real-world variations in airline ticket sales). The study concludes that RL-based methods, mainly bootstrapped DQN, effectively increase airline revenue by dynamically adjusting pricing based on customer behavior. The paper highlights the importance of including historical pricing information in dynamic pricing models. It suggests that alternating between high and low prices early in the sales period can lead to better outcomes in environments with patient customers.

Although this document does not directly address the project's primary objective, it identifies several crucial attributes that merit consideration. The project aims to engineer a resilient algorithm capable of adapting to variations within the training dataset, similar to the scenario outlined in this document. Moreover, it is essential to recognize the role of an external factor, namely the customer, in influencing the dynamics of price simulation. This factor is critical for the development of algorithms designed to exhibit resilience.

In [Sha+23] investigates a flexible job shop scheduling problem with dynamic job arrivals and urgent job insertions in semiconductor manufacturing. The study utilizes a Double DQN algorithm to solve this problem by optimizing real-time job assignments and machine allocation. The primary objective is to minimize completion time and average tardiness while considering the dynamic nature of semiconductor production, where urgent jobs like test chips must be prioritized due to their high impact on research and development timelines. The state space includes machine utilization, operation completion rate, and real-time shop floor data.

The Double DQN algorithm is compared against Deep Q-Network, Q-learning, SARSA frameworks, and traditional heuristic methods. The study evaluates the algorithms using parameters such as job completion time, machine workload, and tardiness. The results demonstrate that the Double DQN outperforms other algorithms, particularly in scenarios with high job arrival variability and urgency. This algorithm's real-time adaptability reduces completion time and increases scheduling efficiency for regular and urgent jobs.

This paper is deemed relevant for exploring the abrupt integration of new jobs into a series of pre-existing ones. A pivotal contribution made in Chapter 4 involves the development of an orders list. This list comprises a predetermined array of potential orders to be simulated by the algorithm designed for this project. The capability to insert or modify new orders has been identified as a potential scenario for resilience.

In [Yon+24], the paper investigates the dynamic distributed job shop scheduling problem with transfers and random job arrivals, using deep reinforcement learning. The goal is to minimize mean tardiness in distributed job shops where multiple factories and machines operate simultaneously. The study models the problem as a MDP and uses state features extracted from the dynamic distributed job shop. Five DRL algorithms, including DQN, Double DQN, Dueling DQN, Trust Region Policy Optimization, and Proximal Policy Optimization, are employed to address the scheduling problem.

The algorithms are compared based on their ability to minimize tardiness while handling real-time scheduling with random job arrivals and operation transfers. The deep RL approaches were tested under 243 production configurations to evaluate their effectiveness and generalization capabilities. The results indicate that the scheduling methods can significantly improve scheduling efficiency by reducing mean tardiness in complex and dynamic environments.

This study has elucidated its importance through a comparative analysis of diverse algorithms, presenting a comprehensive overview of each algorithm's application and advantages. It has systematically demonstrated enhanced performance in every evaluated scenario.

In [Zhi+23], the paper addresses the problem of scheduling virtual network functions in a service function chain for low-latency networks. The study proposes a scheduling algorithm using Dueling Double DQN,

which integrates RL to optimize the placement, scheduling, and routing of virtual network functions. The objective is to minimize the rejection rate while ensuring that delay-sensitive virtual network functions are scheduled within strict service deadlines. The paper models the scheduling as a MDP and utilizes deep RL to maximize long-term rewards in dynamic network environments.
The Dueling Double DQN algorithm is compared against traditional methods like Genetic Algorithms, Single-rule-based heuristics, and DQN. The comparison parameters include SFC rejection rate, average virtual network functions processing time, and network bandwidth utilization. Results show that the algorithm outperforms these methods, significantly reducing the rejection rate by approximately 8% compared to genetic algorithms, especially in high-traffic scenarios. The study concludes that integrating virtual network functions placement and scheduling with routing optimization in Dueling Double DQN leads to improved network efficiency for low-latency services.
This research presented a case for employing the Dueling Double DQN algorithm as a potential solution for this project. Despite the application being initially regarded as beyond the project's scope, it was nonetheless considered for methodological comparison in Section 3.3.1.
In [Luo20], the paper addresses the dynamic flexible job shop scheduling problem with new job insertions. The study applies a DQN algorithm to optimize scheduling decisions dynamically. The primary goal is to minimize total tardiness in the system. The flexible job shop problem is modeled as a MDP, where decisions are made based on real-time production conditions. The algorithm is designed to handle continuous production states, selecting the most suitable operation and machine combination at each scheduling point. The authors propose six composite dispatching rules, which are evaluated using state features such as machine utilization and job completion rates.
In terms of algorithm comparison, the DQN approach is compared against traditional dispatching rules and standard Q-learning algorithms. The evaluation is based on minimizing total tardiness and ensuring timely job completion. Parameters such as machine utilization, job completion rates, and tardiness are used for comparison. The results show that the DQN method outperforms traditional dispatching rules and the Q-learning agent, demonstrating its ability to generalize and handle real-time dynamic scheduling situations more effectively.
This research delves into the real-time production dynamics of job shops, introducing the DQN algorithm as a proposed solution. The critical evaluation of this proposition is detailed in Section 3.3.1. Additionally, the study delineates key output parameters, notably tardiness, which quantifies the temporal discrepancy between the outcomes of the AI-driven approach and traditional methodologies, as further elaborated in Section 3.3.2.
In [WT96], the paper explores the application of RL to job shop scheduling, specifically within the context of space shuttle payload processing for NASA. The goal of the study is to minimize the makespan and to repair schedules when unforeseen issues arise dynamically. The approach uses $TD(\lambda)$, a temporal difference RL algorithm, to train a neural network to learn a heuristic evaluation function. This evaluation function is employed in a one-step lookahead search procedure to solve new scheduling problems efficiently. The study demonstrates that RL can automatically generate domain-specific heuristics that outperform traditional scheduling algorithms such as Zweben's iterative repair method based on simulated annealing. In the comparison, the $TD(\lambda)$ scheduler was tested on synthetic problems and real-world NASA space shuttle scheduling tasks, outperforming the iterative repair method. Parameters such as schedule duration (makespan) and the number of constraint violations were used to evaluate the scheduling efficiency. The $TD(\lambda)$ approach proved to be adaptable, successfully generalizing its learned knowledge to handle larger and more complex scheduling problems than those used during training, thus highlighting the potential of RL for high-performance AI scheduling systems.
This paper exerts a minimal influence on this project, yet it remains pertinent. The objective to minimize the makespan, coupled with the methodology employing a neural network to approximate temporal differences

in the RL algorithm, are salient features that contribute to the project's development. These aspects notably impact the time simulation as discussed in Chapter 4.

In [SYL23], the paper addresses the challenge of multi-objective order scheduling in communication industries, which involves managing complex and high-volume work orders. The study introduces a solution using a Multi-Objective Reinforcement Learning approach, specifically focusing on optimizing Priority Dispatching Rules. The algorithm leverages a Graph Neural Network to represent the scheduling tasks and uses Proximal Policy Optimization to train the policy network. The key objectives in this scheduling problem include minimizing time, cost, and distance for work orders, and the study integrates these factors using a convex hull algorithm to find the optimal weight vector for the reward function.

Compared with traditional scheduling methods, such as the Shortest Processing Time and Most Work Remaining, the proposed method achieves significant improvements, particularly in multi-objective optimization. The parameters used for comparison include processing time, operation costs, and distance traveled for work orders. The results demonstrate that the approach outperforms single-objective methods, providing a more balanced optimization of all objectives and enhancing scheduling efficiency, especially for larger instances.

This manuscript presents the application of a sophisticated algorithm, which combines the Shortest Processing Time and Most Work Remaining, which warranted attention in section 3.3.1 due to the imperative of evaluating complex algorithms, notwithstanding the potential for their exclusion. The subject of investigation aligned with the project's scope, thus affirming its pertinence to the assessment process.

In [BAB21], the paper explores the application of RL for solving Job Shop Scheduling Problems to minimize the makespan and optimize production efficiency. The proposed approach integrates Proximal Policy Optimization as the core RL algorithm, aiming to train an intelligent agent capable of dynamically optimizing the allocation of jobs to machines. The objective is to achieve near-optimal solutions in real-time, making the scheduling system both efficient and adaptive to changes in production environments.

The study compares the performance of the RL-based system with traditional heuristic methods like Shortest Processing Time and Longest Processing Time. The comparison parameters include execution time and the quality of the solution, measured by the makespan. Results show that the RL-based method provides a competitive trade-off between time and quality, significantly outperforming traditional heuristics and offering solutions in a fraction of the time compared to more computationally intensive algorithms like Peng's and Zhang's approaches.

This paper has demonstrated its significance by discussing strategies to minimize the makespan, an essential output parameter for evaluating the performance of RL algorithms. This study aims to develop a solution that operates in real-time, thereby enhancing the scheduling system's efficiency in response to changes within the production environment. Additionally, the algorithm's resilience was identified as a pertinent factor, underscoring its importance in the broader adaptive and robust system design context.

In [BG20], the paper provides a comprehensive literature review on applying RL in solving machine scheduling problems. The study analyzed 80 papers published between 1995 and 2020, focusing on the RL algorithms used, the machine environments applied, job characteristics, objectives, and benchmark methods. The paper highlights that job shop and unrelated parallel machine scheduling are the most frequently studied problem types. The primary objective of the review is to identify trends and gaps in the literature, providing insights for future research.

Various RL algorithms, such as Q-learning, SARSA, and DQN, are compared based on their effectiveness in dynamic environments. The study emphasizes that RL methods have performed better than conventional approaches, particularly in dynamic environments with large datasets. However, there are still unexplored areas, such as multi-objective problems, where RL could be further developed and applied. The paper concludes that while RL offers significant potential in scheduling problems, more research is still needed in specific types of scheduling environments and constraints.

This paper has conducted a comparative analysis of diverse methodologies, incorporating an exhaustive review of the pertinent literature. It constitutes a significant contribution to the field as it elucidates the distinctions among several RL algorithms, aligning precisely with this chapter's objectives.
In [Xia+24], the paper addresses the flexible double shop scheduling problem, which integrates job and assembly shop scheduling. The study proposes a novel RL approach using a deep temporal difference network to minimize the makespan, focusing on the scheduling association between related tasks in both shops. The problem is modeled as a MDP where the state and action spaces are optimized using historical machining data. Ten state features are defined to represent the job shop environment, and eight simple constructive heuristics are used as candidate actions for scheduling decisions. The RL-based deep temporal difference network algorithm outperforms traditional methods, showing better efficiency in solving practical scheduling problems in manufacturing.
The study compares the deep temporal difference network algorithm with other methods such as genetic algorithms and particle swarm optimization. The parameters of comparison include the makespan, processing time, and machine utilization. Results indicate that deep temporal difference network achieves better scheduling performance and flexibility in handling dynamic and complex scheduling environments, reducing waiting times and improving overall production efficiency.
This paper introduces an innovative logistic regression algorithm, which offers significant contemplation for our current project. Despite its appeal, a pragmatic approach suggests favoring a simpler algorithm to enhance performance in simulating the production chain. The potential for employing more sophisticated algorithms exists for future endeavors, making this paper a critical reference for forthcoming research activities

3.3 Intensive Literature Comparison

In the subsequent section, following the analysis provided in Section 3.3, a synthesis is formulated, encapsulating the salient information from each referenced study. The objective of this synthesis is to methodically determine i) the optimal algorithm for the implementation within the scope of this research and ii) the essential KPIs required for the comparative assessment of the RL algorithm against the established benchmark algorithm. Although the benchmark algorithm was identified as pertinent to this research endeavor, traditional methodologies employing Petri nets for simulation processes have been previously acknowledged as appropriate for executing the production simulation tasks.

3.3.1 Methods Comparison

In this section, we aimed to determine the most suitable algorithm for our project. We selected 15 papers from the ScienceDirect database, which we briefly discussed in Section 3.3. We summarized the paper's objectives, the algorithms used, additional characteristics of the algorithms, and the benchmark method in Tables 3.1 and 3.2.

Table 3.1: Literature Revision: Algorithm Comparison (part 1).

Name of paper	Goal of the paper	Algorithm used	Additional information about the algorithm	Benchmark method
[Sum+24]	Production rescheduling	Explorative RL	MDP Policy Proximal Optimization (PPO)	General Algebraic Modeling System with CPLEX solver for mathematical optimization
[SGI24]	Dynamic pricing problem	DQN, bootstrapped DQN	MDP Reward Function State Variables	traditional dynamic pricing algorithm
[Sha+23]	Dynamic Flexible Job Shop	Double DQN	MDP Reward Function Scheduling rules state features	DQN Q-Learning State-Action-Reward-State-Action
[Yon+24]	Deep Reinforcement Learning	DQN	MDP Reward Function	Numerical experiments
[Zhi+23]	Virtual Network Function scheduling	Dueling Double Deep Q-Network	MDP	standard Deep Q-Network
[Xia+24]	Deep Temporal Difference Network	MDP Reward Function	Q-Learning, Deep Deterministic Policy Gradient and standard heuristics	
[XJR12]	Scheduling Decision in Make-to-Order System	Q-Learning Algorithm	Semi-MDP Discrete-Event Simulation Model	heuristic policy
[WT96]	Job-shop scheduling	Temporal Difference Learning	MDP Reinforcement Function Multiple-Objectives	tradicional scheduler performance
[SYL23]	Multi-objective Order Scheduling	Deep Multi-Objective Reinforcement Learning: optimize the Priority Dispatching Rule	MDP Reward Function Graph Embedding	Tradicional PDRs such as Shortest Processing Time and Most Work Remaining
[BG20]	Machine Scheduling Problem	Q-Learning DQN	MDP State representation Objectives and Contraints	Tradicional dispatching rules Metaheuristic Method (Genetic Algorithms, Simulated Annealing)
[Ton+]	Online scheduling in smart factories	Multi-Agent Reinforcement Learning	Cyber-Physical System Architecture State Representation Reward Functions Neural Network	Genetic Algorithms, contract net protocol, centralized reinforcement learning

Table 3.2: Literature Revision: Algorithm Comparison (part 2).

Name of paper	Goal of the paper	Algorithm used	Additional information about the algorithm	Benchmark method
[JBD21]	Permutation Flow Shop Scheduling	RL	MDP State Representation Reward Functions	Mixed-Integer Linear Programming and Constraint Programming
[Luo20]	Dynamic Flexible Job Shop	DQN	MDP Reward Function state features	Tradicional dispatching rules and Q-learning agent
[AFI17]	Scheduling in Heterogeneous Distributed Systems	Q-Learning SARSA	MDP Three Layers of complexity	HEFT
[BAB21]	intelligence scheduling	Proximal Policy Optimization	MDP Reward Function State and Action Spaces	Longest Processing Time and shortest Processing Time

Upon examining the data presented in Tables 3.1 and 3.2, it was observed that the DQN and Q-learning algorithms were the most frequently utilized methodologies in the reviewed literature. The Q-learning algorithm, as delineated by [Wat89], offers a straightforward mechanism for agents to achieve optimal decision-making in environments characterized by Markovian dynamics. This algorithm represents a foundational approach within the realm of RL, which is advantageous for the objectives of this study. Conversely, the DQN algorithm amalgamates Q-learning principles with *deep convolutional* artificial neural networks, which are adept at processing spatially structured data arrays, such as images [RA20]. Given the foundational role of Q-learning in developing DQN and its prominence in the literature, it was deemed an appropriate choice for the methodologies employed in this research.
The papers also examined the use of a MDP and the implementation of a Reward function. This function is utilized during the algorithm to maximize the decision-making process based on the highest score from the reward function. For instance, when choosing between two paths, where one is two kilometers longer than the other, the reward function should assign a higher score to the shorter path, prompting the algorithm to select the shortest route.
In the analyses presented within the papers, the benchmark algorithm was addressed lastly, as noted in Tables 3.1 and 3.2. Predominantly, two methodologies were utilized: the conventional approach leveraging the general algebraic solution and a straightforward RL technique, specifically Q-learning. Given this context, it was deemed appropriate to juxtapose the RL solution with the conventional algebraic method. The latter was conceptualized through the framework of Petri Nets for simulating a Production Plan.

3.3.2 Key Performance Indexes selection

A critical objective pursued through the literature review was the identification of KPIs, which are essential parameters that can be utilized to attest to and validate the performance of the algorithm developed for this project. This was achieved by extracting key parameters from the literature used as performance criteria, facilitating the development of a discussion on the results. Table 3.3 presents the principal KPIs obtained from the analyzed papers. Based on this compilation, three KPIs were specifically chosen for the performance analysis of the developed algorithm.

Table 3.3: Literature Revision: KPIs utilized in the papers to measure performance.

Name of paper	Criteria 1	Criteria 2	Criteria 3	Criteria 4	Criteria 5
[Sum+24]	Total Assignment Cost	Schedule Nervousness	Computational Time	Feasibility and Efficiency	
[SGI24]	Total Revenue	Exploration Efficiency	Consistency and Stability		
[Sha+23]	Makespan: Maximum Production time	Mean Tardiness	Time of usage of machine Index	Robustness	Ressources efficiency
[Yon+24]	Ability to response for dynamic events	Mean Tardiness	Time of usage of machine Index	Transfer Efficiency	
[Zhi+23]	Rejection Index	Mean Cost to rotate between states	Mean Tardiness	Mean Time of usage of the nodes	
[Xia+24]	Makespan: Maximum Production time	Mean Tardiness	Time of usage of machine Index	Transfer Efficiency	
[XJR12]	Request Acceptance Index	Quantity Acceptance Index	Mean Reward		
[WT96]	Ressource Dilation Factor	Repair actions number	Comparative Perfomance		
[SYL23]	Makespan	Cost	Distance	Generate Data Perfomance	
[BG20]	Environment type	Goal of the problem	Algorithm Performance		
[Ton+]	Makespan: Maximum Production time	Balance of duty's load	Resilience of unexpected Events		
[JBD21]	Makespan: Maximum Production time	Idle Time	Solution quality		
[Luo20]	Total Tardiness	Mean and extimated Tardiness	Reward Rule Efficiency		
[AFI17]	Number of States	Cumulative reward	Execution time		
[BAB21]	Runtime: Time of execution	Makespan: Maximum Production time			

According to Table 3.3, the three main KPIs that can be used are the makespan (maximum production time), the mean tardiness, and the computational time. For the mean tardiness, it was understood to be the calculation of the MSE, explained in Chapter 2.13. For that, it was also considered relevant for the results discussion to analyze the MAE, explained in Chapter 2.14, because in terms of performance, usually an AI Application method can be considered effective only if it achieves significantly less error than the statistical baseline bearing with its disadvantages of more complexity (both in design and implementation) and resource-intensive (in terms of memory and computing cycles) as well as less interpretability. Normally, the MAE shows an approximate linear growth, and the MSE shows a quadratic growth. [Hao+22].
The computational effort required to solve a system of is compensated for by a relaxation in the restriction in the time step that can be used in the simulation [Zop00], what, in other words, can be measured as the time difference between the start of the program and its end, thus calculating its effort for a specific hardware specification in certain condition of usage of the machine.
The makespan can be defined as the minimization of the maximum completion time of the last operation performed [DSA23]; in other words, the makespan refers to the total time required to complete a set of tasks or processes in a system, particularly in scheduling problems. Specifically, makespan is the elapsed time from the start of the first task to the completion of the last one.

3.4 Resilience: Literature Review

According to [Bhu+20], resilience can be defined as one of the most straightforward definitions for resilience, which is the ability of a system to absorb any changes or disturbances. To lower the impact of disturbances on the system's performance, it should be able to absorb the disturbance efficiently. Higher absorption efficiency implies higher resilience.
The paper explores six resilience principles in the industrial field, which can be listed as follows.

- **Flexibility** is the ability of a process to keep its outputs variation within a specified range when the input inflexibility of a process makes it more resilient to disturbances
- **Controllability** a process is controllable if its output parameter can be controlled and set to target points within a certain time when there is an unexpected deviation in input parameters with a steady state. In contrast, controllability refers to the dynamic state of the process.
- **Early detection** Early detection of disturbances occurs when all the preventive measures fail to avoid the disturbance. This principle is important because an undetected disturbance can often lead to disasters.
- **Minimization of failure** Minimization of failure refers to implementing preventive measures to avoid disasters.
- **Limitation of effects** In case of an unavoidable disastrous event, it is important to ensure that the loss to the process system is minimum. This principle is used for setting up strategies to limit the consequences of a disastrous event.
- **Administrative controls/procedures** Some disturbances cannot be detected and prevented using the abovementioned principles. This principle helps in developing management systems such as training program for workers, updating operating procedures, etc. for dealing with such disturbances.

According to [Lih+11], four social-technical factors affect the resilience of a team of workers during a disaster, which are monitoring, learning, responding, and anticipating, as well as relation-oriented qualities like leadership and cooperation with another team. The study explores the way of proceeding with resilience are receiving feedback from the employees, performing a literature review, identifying possible sources of disturbances, and then suggesting measures to make the plant more resilient. Figure 3.2 illustrates the process of resilience in the industrial chain.

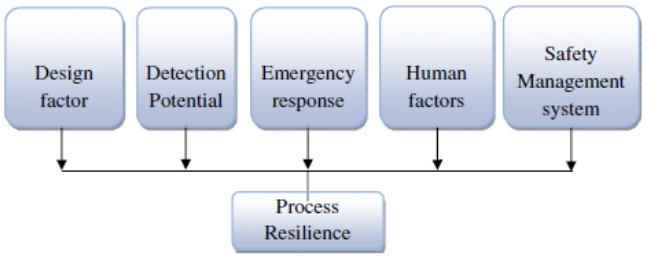

Figure 3.2: Contributing factors of process resilience.[Lih+11]

Based on the analysis of the papers' components, it was determined that in order to proceed with the resilience analysis during the algorithm simulation, it was imperative to develop a potential disturbance

scenario, such as machinery malfunction, in which the parameters could be manipulated to intentionally disrupt the algorithm. This would yield significant data for monitoring and subsequently enable conclusive insights into the algorithm's performance. Further elaboration on the implementation of these characteristics will be provided in Chapter 4.1.3.

4 Project Development

The following passage outlines the project's approach to the algorithm, considering all previously introduced elements. It commences by detailing the methodology employed in developing the software project, known as the V-Model. Subsequently, key project elements are introduced, including hypotheses, the simulation environment, and resilience scenarios as discussed in Section 3.4. In Chapter 5, we provide a modeling explanation, discussing the Petri nets models, the RL environments for the algorithm, and the KPIs used to evaluate the performance. This chapter also introduces the input variables used for the simulation and to train the RL algorithm. Throughout the project, we utilized Matlab R2024a® as the primary tool to generate all the results presented in Chapter 5. The Matlab® code was annexed to this project's documentation.

4.1 First Considerations

This section delineates the simulation's initial conditions and resilience scenarios, including the environmental modeling approach. The term "initial conditions," also called a hypothesis, specifies the environment's modeling conditions, the parameters under consideration, and the types of interference that can be preemptively excluded. Moreover, Section 4.1.2 elaborates on the modeling of the production plane, taking into account the initial conditions outlined in Section 4.1.1. Lastly, Section 4.1.3 examines three scenarios of disturbance within the production simulation, aiming to compare the responses of the RL algorithm with those of the traditional Petri Nets Simulation method.

4.1.1 Hypothesis

The initial conditions for modeling the industrial production environment are as follows:

1. All external factors related to machinery issues, including workforce disruptions and delays in delivery, significantly affect the overall transportation time.
2. All internal factors related to machinery issues, including malfunction, affects directly the overall production time.
3. In an optimal operational framework, all machinery work with the same efficiency for each respective component. Thus, it is feasible to parameterize production and transportation times as fixed constants within this model.
4. The machinery can function beyond standard labor hours; it is considered a 24-hour factory with no stop hours.
5. The storage units are designated exclusively for storing specific products and are subject to a predetermined maximum capacity.
6. The analysis will encompass six distinct products, designated as A, B, C, D, E, and F, in addition to one base and one electrical component.

7. To assemble the final product, four distinct or identical components, one base, and one electrical component are required.
8. The base and the electrical component assemble four distinct products and exhibit universal compatibility with each.
9. The 3D printer will manufacture the base component.
10. The electrical component comes ready to use from the supplier.
11. The simulation focuses exclusively on the flow of produced materials and the machinery involved, which require the incorporation of supplementary materials.

These conditions are designed to transform the system into a computationally modelable approximation. It is important to note that this model intentionally omits certain disturbances that may occur in real-life scenarios to simplify the computational process.

4.1.2 Flowfactory Environment

The initial phase in constructing a RL algorithm involves defining the operational environment. The Flow Factory Laboratory, affiliated with the PTW at the TUDa, was selected to conduct simulations. This environment conformed to all previously established hypotheses detailed in Section 4.1.1, ensuring its appropriateness for modeling.
The diagram in Figure 4.1 illustrates the operational model of a FlowFactory environment. Within this model, an order initiates the process by specifying the required quantities of products A, B, C, D, E, and F needed to assemble a final product. This final product invariably comprises four components. This data is conveyed to the main storage facility upon determining the requisite quantities for production. At this juncture, the facility can execute one of three actions: manufacturing the product in full, applying a laser stamp, or advancing the four components to the assembly stage. Notably, the main storage can accommodate two variations of the same product: one that has been manufactured but lacks the laser stamp and another ready to be dispatched for the commissioning process.

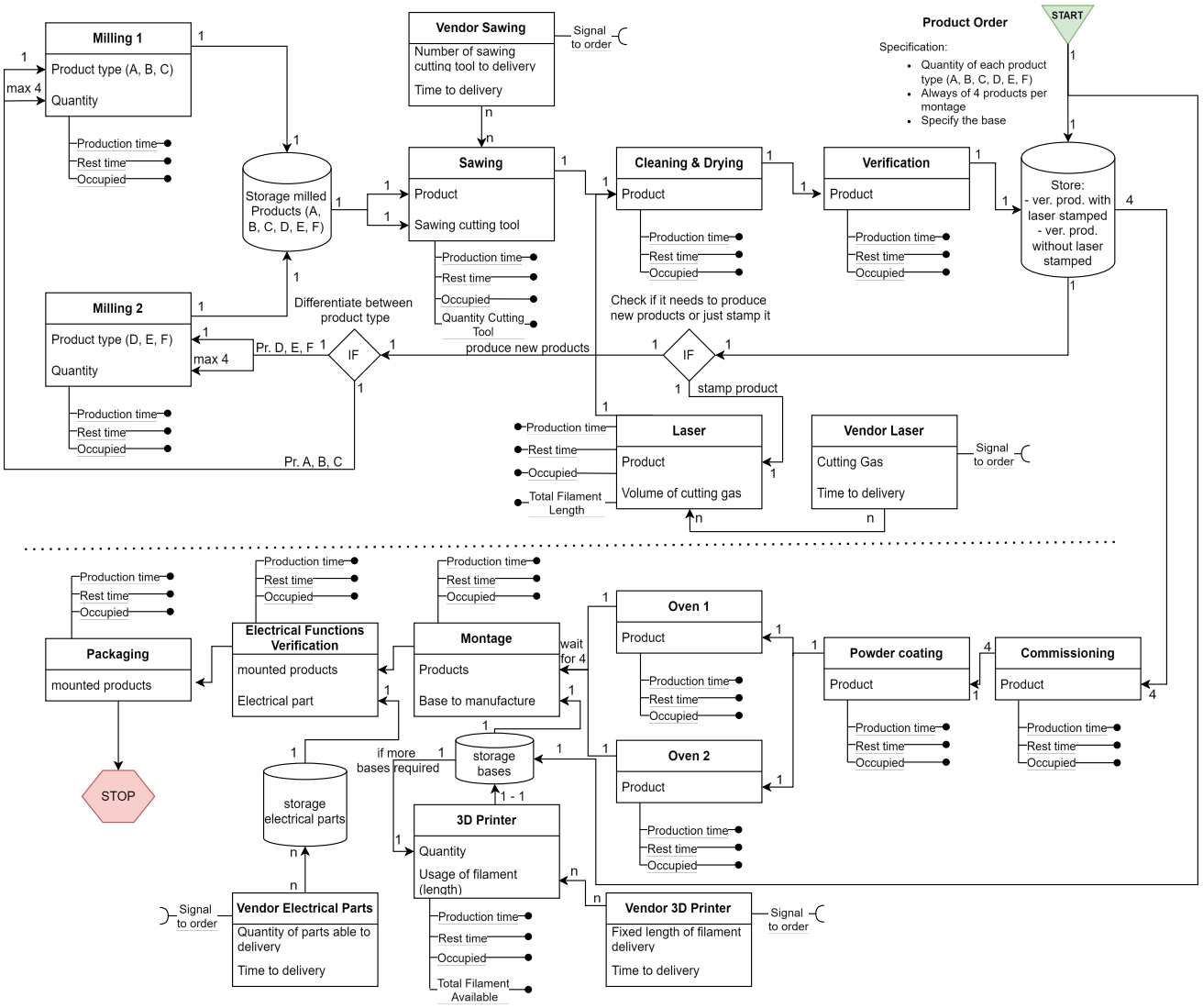

Figure 4.1: Flow Factory Simulation Environment

The objective of the simulation for the so-called traditional method is to assess inventory levels in the primary storage to ascertain the availability of products, which can be seen in Figure 4.2. The prioritization scheme dictates that products ready for dispatch are given precedence, followed by products lacking a laser stamp, and lastly, the fabrication of entirely new products is considered. A crucial aspect of this process involves determining whether to manufacture a base component. This decision is contingent upon inspecting "storage bases" to determine the availability of requisite base components. Without these components, a 3D printer fabricates a new base. Similarly, the protocol for electrical components mirrors this process; it entails verifying the availability of these components in storage. Should there be a deficiency, a requisition for a new shipment of electrical components from the supplier is initiated. This simulation is designed to optimize inventory management and ensure the efficient allocation of resources in the manufacturing process.

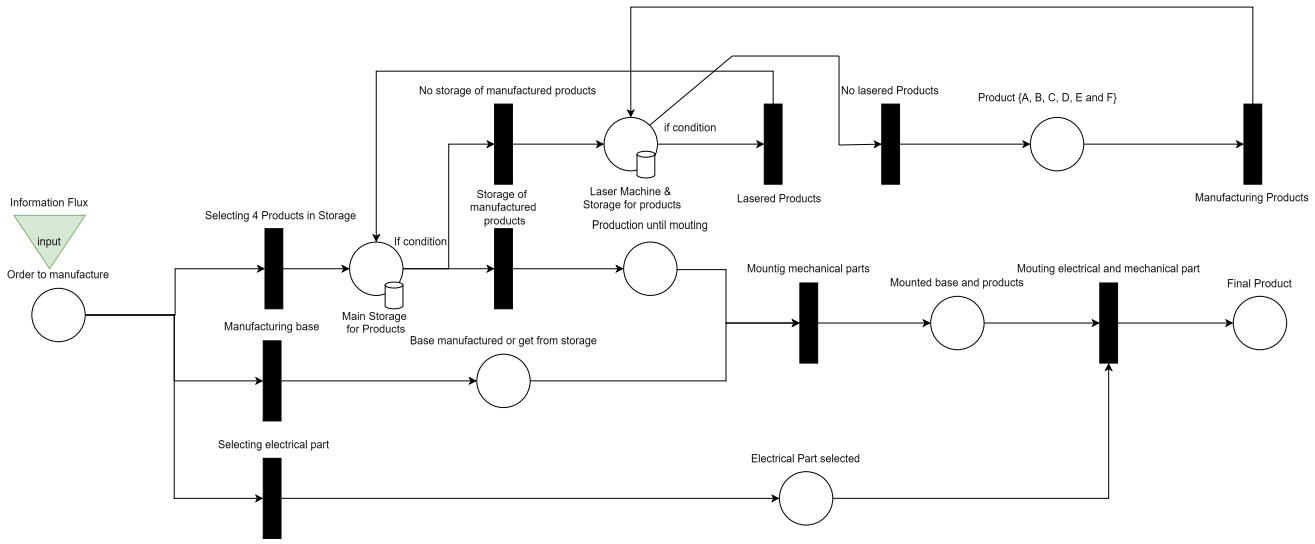

Figure 4.2: Informational flow in the simulated environment.

In equipment behavior concerning material replenishment, sawing machines, laser cutters, and 3D printers demonstrate analogous patterns to those observed in electrical functional verification processes. This similarity is particularly evident in scenarios where material depletion necessitates intervention, traditionally involving the engagement of an external supplier. Modern adaptations to this process have seen the integration of decision algorithms designed to autonomously initiate material reordering from suppliers, thereby streamlining the replenishment process. This mechanism and its underlying principles are slated for comprehensive exposition in Section 4.2.
Each piece of machinery—sawing machines, laser cutters, and 3D printers—utilizes specific consumables essential for its operation: cutting tools, cutting gas, and filament, respectively. These materials are integral to their respective operational phases, ensuring the continuation of the manufacturing process without interruption.
The principal objective of the RL-scenario is to develop an autonomous decision-making algorithm that minimizes environmental dependency by exclusively utilizing information flux. This approach entails executing decisions concurrently across disparate segments of the project, thereby facilitating parallel actions within the simulation framework. The ultimate aim of such parallelization is to enhance the efficiency and optimization of the simulation process.
Section 4.2 will further detail how the material flows over the environment.

4.1.3 Resilience Analysis

The approach involved simulating specific scenarios reflective of real-world conditions to accurately assess the software's resilience. Consequently, three distinct situations were identified and considered for this analysis.

1. Storage disruption: Simulates the scenario when a machine's inventory is compromised, causing it to operate at reduced capacity.

2. Machinery malfunctions: Simulates a potential issue with a machine, such as a worn tool, that reduces machine efficiency.

3. Workforce disruption: When a labor problem arises, such as a shortage of necessary workforce, it directly impacts production dynamics.

Clarifying that this project did not develop a real-time algorithm is critical. The primary objective of this study is to conduct a comparative analysis between a RL algorithm and a benchmark algorithm. Consequently, the three scenarios presented were not generated using heuristic random algorithms to emulate sudden changes in production. Instead, they were systematically pre-defined, spanning from one specific order to another, as elaborated upon in Section 4.3.
Disruptive scenarios can be analyzed using the input parameters defined in Section 4.1.1. In the case of storage disruption, the storage cost (storage quantity) defined in Equation 2.17 will be affected. For machinery malfunction, the production cost in Equation 2.17 is directly impacted, as stated in one of the hypotheses. In the scenario of workforce disruption, the transportation cost (transportation time) outlined in Equation 2.17 will be affected. This was also established as a hypothesis for the study in Section 4.1.1.

4.2 Modeling the solution

4.2.1 Petri Nets

This section delineates the environment outlined in Section 4.1.2, employing a Petri Nets model to elucidate the material flow dynamics. Given the foundational discussions in Section 4.1.2 and referencing Figure 4.2, three distinct models are delineated that operate autonomously yet are contingent upon decisions made during the information flow process. Each model initiates from variable starting points within the stage, predicated on the component availability within the main storage. Ultimately, these models converge at a singular phase, culminating in assembling the final product and integrating the base, electrical component, and packaging.
It is noticeable that Figure 4.5 is a part of Figure 4.4, and subsequently, Figure 4.4 is a part of Figure 4.3. Due to the image size for this documentation, the last element represented by a square in the Figure 4.3 can be understood as the Petri net flow of the 4.4, and consequently, the square state of this Figure 4.4 represents the Figure 4.5. Therefore, the explanation of the Petri net models will be presented. First, the parts specific to each model will be explained, followed by an explanation of the parts that are common to all models.
Referencing Figure 4.3, the flow of materials can be elucidated in the absence of inventory. The procedural blueprint for product fabrication initiates at the outset. The protocol mandates that all products begin manufacturing with the milling process. As delineated in Figure 4.1, Milling Machine 1 is designated for processing Products A, B, and C, whereas Milling Machine 2 is tasked with fabricating Products D, E, and F. Subsequent to the milling phase, the workflow dictates that each product undergoes sawing, followed by cleaning and drying processes, culminating in a verification stage. Upon successful verification, products are relegated to the main storage area, pending further processing with the laser stamping technique, details of which are expounded in the subsequent paragraph.

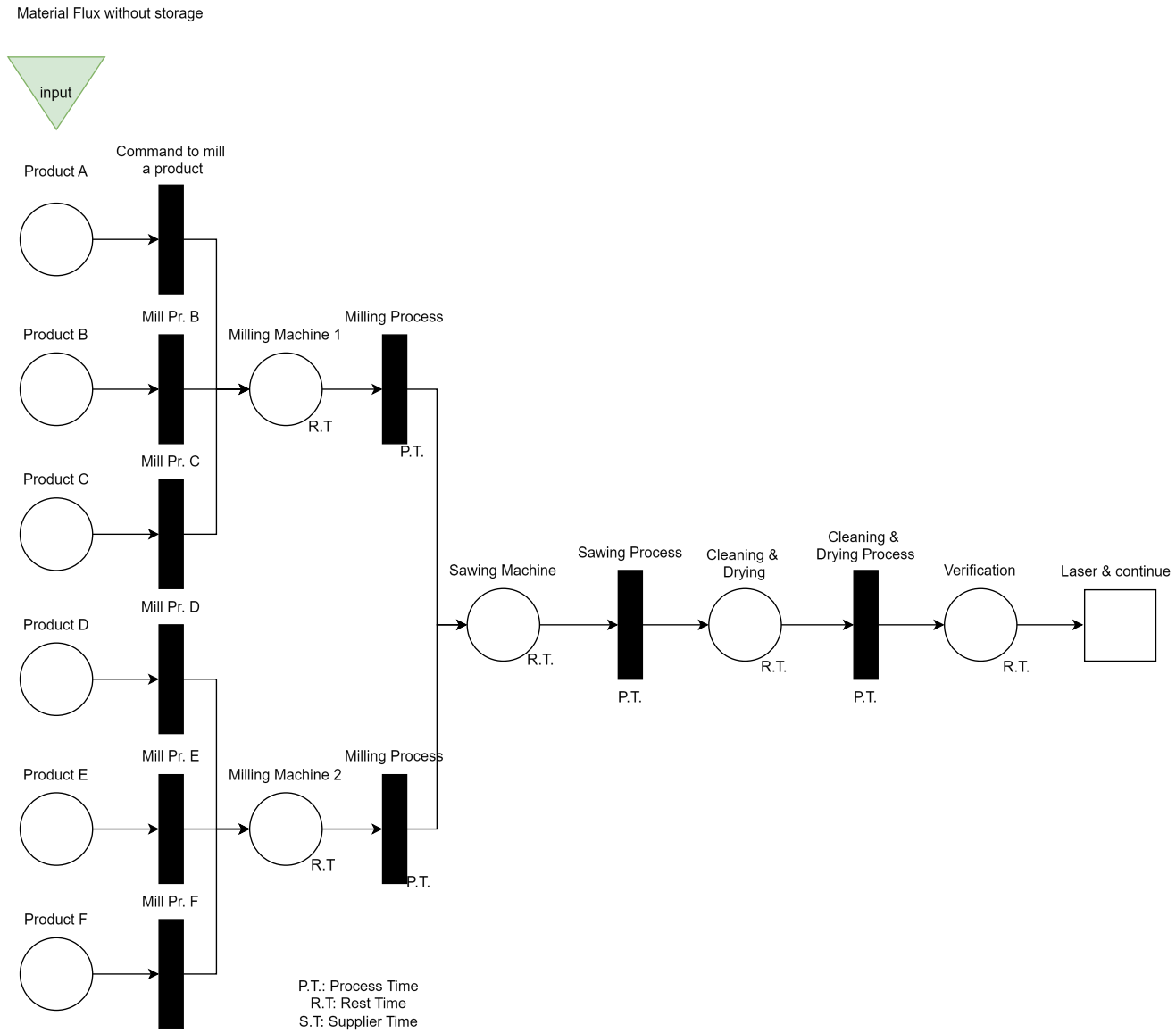

Figure 4.3: Petri Net Simulation of the environment, when the product must be all manufactured and passed throw all stages.

In Figure 4.4, similarities are observable with Figure 4.3, particularly in the context where products are available in the main storage but require the laser stamping phase. Initially, the product is retrieved from the storage and directed to the laser machinery for processing. After the laser stamping, the product undergoes a series of post-processing steps, including cleaning, drying, and verification, to ensure it meets the required specifications. Following these procedures, the product is returned to the main storage and prepared for the assembly phase. This assembly process is delineated in the subsequent paragraph, providing a comprehensive overview of the product's preparation stages.

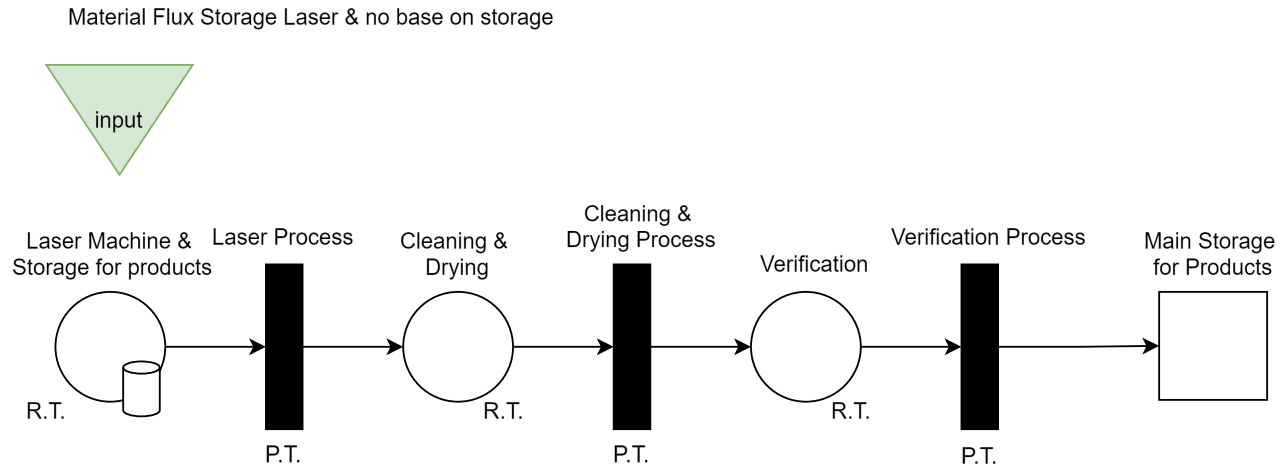

Figure 4.4: Petri Net Simulation of the environment, when the product available in storage must receive the laser stamp.

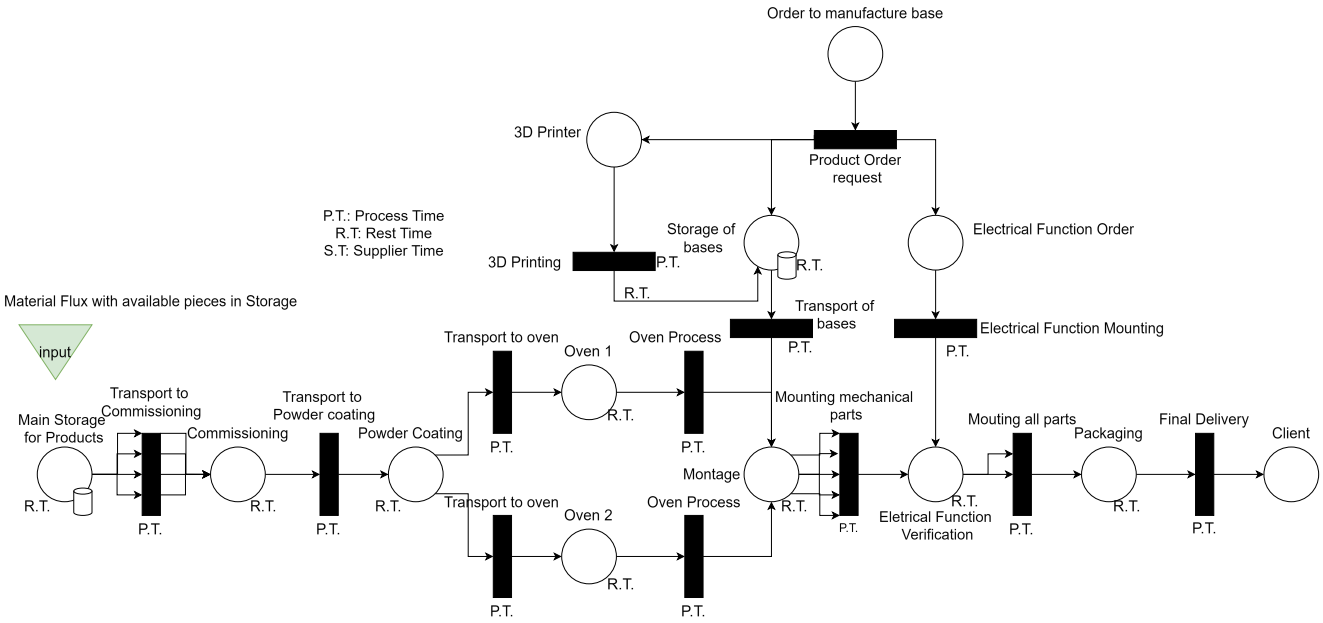

Figure 4.5: Petri Net Simulation of the environment, when the product in storage is ready to use and can be forwarded for the montage.

Figure 4.5 illustrates four essential components' availability during the model's information flow process. The progression of this model is contingent upon the completion and readiness of these four components. In the absence of their readiness, the simulation is halted, necessitating the completion of models referred to in 4.3 or 4.4 to fabricate these four essential components. After completing these components, they are advanced to the commissioning phase, which is succeeded by the powder coating process. During the oven phase, the components have the potential to be divided due to the existence of two parallel processing

paths, culminating in the montage phase. It is in the montage phase, where the base is integrated with the four components. If the base is unavailable, a new one must be fabricated using a 3D printer. Following the montage phase, the final product undergoes the electrical function verification phase, where an electrical component is integrated into the final product, which is subsequently directed to the packaging stage, thus concluding the production flow.

4.2.2 Definition of the Reinforcement Learning Agent

The application of RL was segmented into three straightforward modules: the *Prioritization Algorithm*, the *Petri Net Model Selection Algorithm*, and the *Fulfillment Storage Algorithm*. Distinct from the conventional methodology, the RL approach does not employ an optimized Petri net. Instead, the selection among the models represented in Figures 4.3, 4.4, and 4.5 is predicated upon the components' availability in storage, as delineated in Section 4.2.1. The decision-making process within this RL framework targets three objectives: firstly, to reorder the list of orders to minimize the cost function associated with the arrangement; secondly, to ascertain the appropriate Petri net model for the product of each order; and thirdly, to manage supplier deliveries, determining the necessity to procure new parts.

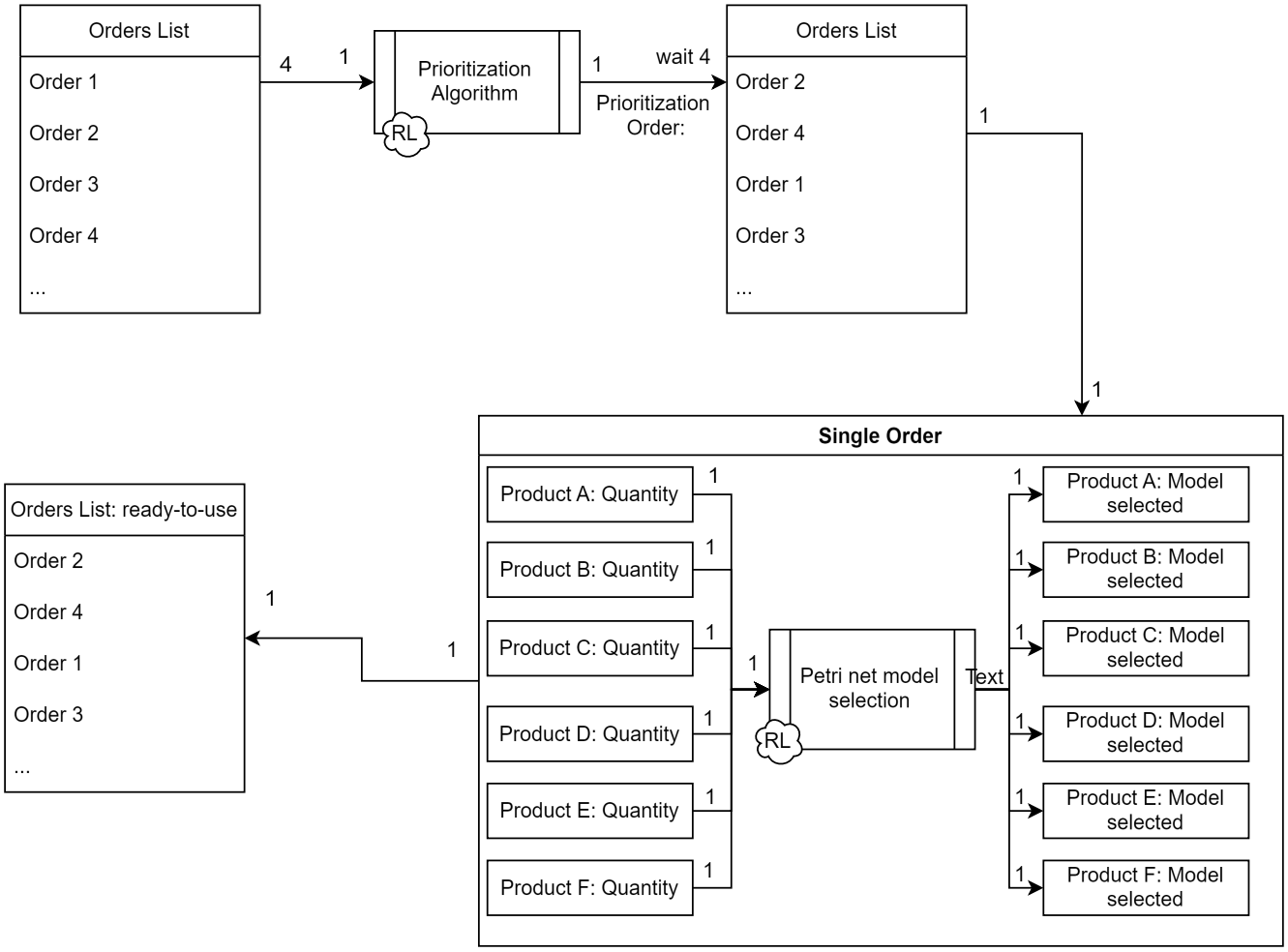

Figure 4.6: Scheme of how the inputs were sorted applying the RL Algorithm.

The illustration in Figure 4.6 delineates the methodologies employed in modeling both the *Prioritization Algorithm* and the *Petri Net Model Selection Algorithm*. The subsequent Section 4.3 is dedicated to elucidating the orders list input, encompassing all orders fed into the simulation framework. Given the non-real-time nature of the simulation, as outlined in [Ton+], there is an imperative to rearrange the orders within a list from the standpoint of descending prioritization dictated by their respective reward functions. The manuscript advocates for the maximum inclusion of four orders in this rearrangement process, which is ope rationalized by applying a Q-learning decision algorithm. After the comprehensive rearrangement of the list, each order is processed through a secondary component of the RL algorithm. This involves the assessment of each order for the quantity of distinct product types it contains, explicitly disregarding any instances of zero quantity. A further application of a Q-learning algorithm ensues, tasked with the selection of an appropriate Petri net model for each product type within an order. This selection process leverages the Petri net models showcased in Figures 4.3, 4.4, and 4.5. After data preparation, the curated order list is inputted into the simulation for processing.

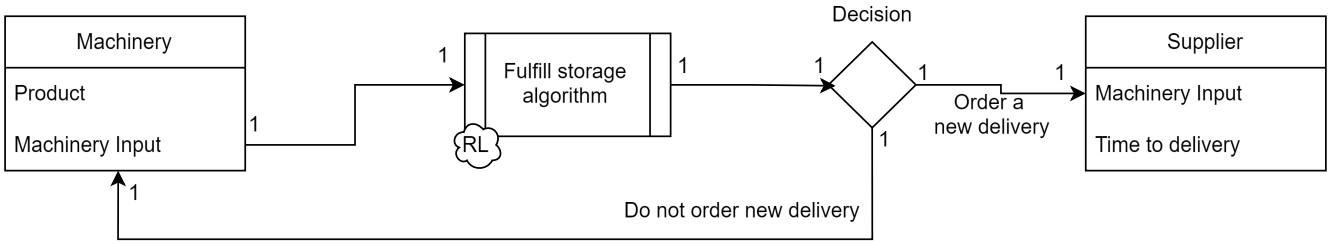

Figure 4.7: Second scheme of the RL applying to decide if new material must be delivered.

Figure 4.7 depicts the third algorithm under discussion, termed the *Fulfillment Storage Algorithm*, which operates during the simulation phase. Examination of Figure 4.1 reveals that the sawing, laser, 3D printer, and electrical function verification stations are critical for the implementation of this algorithm, focusing solely on the aspect of machine storage fulfillment. The forthcoming section will delve into the operational mechanics of the algorithm.

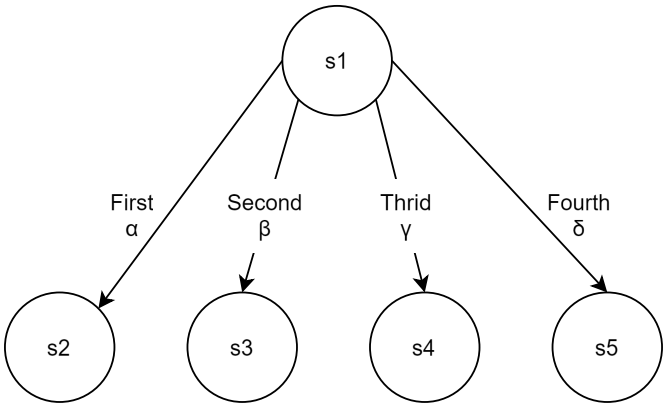

Figure 4.8: MDP Model for the Prioritization Algorithm.

The initial algorithm under examination is the *Prioritization Algorithm*, as depicted in Figure 4.8. Referencing Section 2.5, the foundational step in formulating a RL algorithm entails modeling its environment

utilizing the MDP framework. This environment is characterized by an initial state and a predefined set of potential states, designated in a hierarchical manner based on priority levels: the primary state assumes the highest priority, followed by secondary, tertiary, and quaternary priority levels. Accordingly, the environment encompasses four distinct actions, each leading to a state corresponding to its respective priority. The probability associated with each action is documented in Table 4.3. The significance of each action's probability is paramount in the algorithm's decision-making process, as it prioritizes the action associated with the highest probability value. To illustrate, should the probability denoted as α emerge as the predominant value, the algorithm deduces that the appropriate course of action is "first," subsequently transitioning to the state s_2. This principle of prioritization based on probability values is universally applicable across various algorithms.

Table 4.1: Prioritization Algorithm: Actions and states environment.

s_R	$action$	s'	$probability$	**Reward Function**
s_1	first	s_2	α	$\frac{1}{C(\theta)}$
s_1	second	s_3	β	$\frac{1}{C(\theta)}$
s_1	third	s_4	γ	$\frac{1}{C(\theta)}$
s_1	fourth	s_5	δ	$\frac{1}{C(\theta)}$

A pivotal aspect of the algorithm lies in its reward function, which employs the cost function outlined in Section 2.9. As delineated in Equation 2.17, the cost function escalates proportionally with an increase in costs. Consequently, it is imperative for the reward function to examine the inverse of the cost function to ensure the minimization of costs. Furthermore, it is crucial for the prioritization function to take into account the costs associated with all products (A, B, C, D, E, and F) within an order, as specified in Equation 4.1.

$$C(\theta) = C_A + C_B + C_C + C_D + C_E + C_F \tag{4.1}$$

The final commentary on the *Prioritization Algorithm* emphasizes its objective: to organize a set of four orders by priority. This process involves individually analyzing each order, comparing the likelihood of actions within the group, and selecting the order with the highest probability for each action sequentially. Initially, the order possessing the highest probability of executing the "first" action is chosen. Subsequently, this order is excluded from the group, leaving three orders. From there, the order with the highest probability for the "second" action is selected, followed by the "third." The last remaining order, by default, is assigned the "fourth" position, indicating the lowest priority. This systematic approach ensures that each order is prioritized based on its action probability within a given framework.

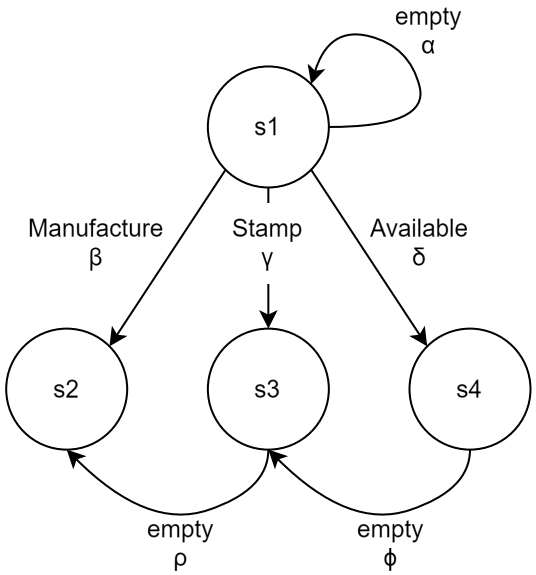

Figure 4.9: MDP Model for the Petri net model selection Algorithm.

The *Petri net model selection algorithm*, the subject of our second discussion, is designed to facilitate a choice among three distinct models as depicted in Figures 4.3, 4.4, and 4.5. This selection process is tailored to each product order individually. Products labeled A through F are each assigned a specific state, denoting the requisite production pathway for that product. The term "manufacture" is used to describe the pathway shown in Figure 4.3, indicating that the product in question is not present in the main storage and must be fabricated from the initial stage. Conversely, the term "stamp" refers to the process illustrated in Figure 4.4, where the product, although available in the main storage, requires processing through the lasering phase. Lastly, the "available" designation, corresponding to the production line depicted in Figure 4.5, signifies that the finished products are in stock within the main storage and are ready to be dispatched to the assembly phase. An illustrative example of this mechanism is when an order specifies the need for two units of Product A as components of the final assembly, and the algorithm opts for the "manufacture" action; this decision mandates the production of Product A from scratch, commencing with the milling process.
The action labeled "empty" exhibits distinct behavior, signifying whether a transition to a different final state is required or if it should persist in its initial state upon execution in state s_1. Essentially, this action possesses the capability to modify subsequent actions by serving as a subsequence, thereby altering the final state. This mechanism, as described, leads to modifications in the chosen Petri net model.

Table 4.2: Petri Nets Model Selection Algorithm: Actions and states environment

s_R	$action$	s'	$probability$	**Reward Function**
s_1	empty	s_1	α	-1
s_1	manufacture	s_2	β	$C_{product}$
s_1	stamp	s_3	γ	$C_{product}$
s_1	available	s_4	δ	$C_{product}$
s_4	empty	s_3	δ	$C_{product} - C'_{product}$
s_3	empty	s_2	δ	$C_{product} - C'_{product}$

This algorithm incorporates the application of a cost function, delineated in Equation 2.17, which pertains to a singular product. Echoing the rationale outlined in the *Prioritization Algorithm*, it is imperative to evaluate the cost function as an inverse, given that the objective of the reward function is to ascertain the

minimal possible cost. The notation $C'_{product}$ is employed to denote the computation of the cost function for the preceding state. For instance, if the current state is s_4, then $C'_{product}$ refers to the cost function value at state s_3.

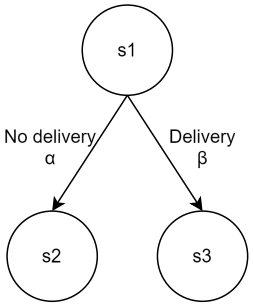

Figure 4.10: MDP Model for the Fulfill Storage Algorithm.

The last algorithm, the *Fulfill Storage Algorithm*, constitutes the third segment of the algorithmic process, wherein the AI application assesses the necessity of initiating a new delivery from the supplier. This evaluation is structured around three distinct states: the initial state, which predetermines the decision-making criteria, followed by two potential outcomes - one state indicating that no further delivery from the supplier is needed, and another state signifying the requirement for a new delivery from the supplier.

Table 4.3: Fulfill Storage Algorithm: Actions and states overview.

s_R	$action$	s'	$probability$	**Reward Function**
s_1	no delivery	s_2	α	$S_{maximum}$
s_1	delivery	s_3	β	$S_{maximum} - S_{status}$

The reward function is determined by the variance between the total capacity of the station's inventory and the quantity of components required to meet that capacity. The goal is to minimize costs. In this scenario, the main focus is on storage, while the time required for processing and transporting with additional machinery is not important when considering new deliveries.

4.2.3 Modeling the Key Performance Indexes for Analysis

According to Section 3.3.2, the performance comparison between the RL and the benchmark algorithms is based on three primary KPIs. Firstly, the computational effort is assessed by measuring the time required for the computer to execute a specific task, with consideration given to minimizing interference from other software programs running concurrently. Secondly, the makespan is focused on reducing the maximum completion time of the final operation, as outlined in [DSA23]. Lastly, MSE and MAE are crucial performance metrics, defined respectively in Sections 2.13 and 2.14.
The first step in analyzing the output is to examine what the simulation produces as a result of each simulation step. Figure 4.11 shows the expected format of the output table after running certain inputs. For each machine, there are two essential time measurements: the start time when the process begins, and the stop time after the process is completed. The time taken for production and transportation is calculated as the difference between these two times and is shown in the subsequent columns. During the sequential process phases, there was a one-minute gap between the stop time of machine one and the start time of machine two.

14×5 table

	Start_Time	Stop_time	Production_Time	Transport_Time
Milling 1	02-Aug-2024 08:42:00	02-Aug-2024 09:14:15	135	1800
Milling 2	22-Jul-2024 12:45:28	22-Jul-2024 12:45:28	128	630
Sawing	02-Aug-2024 09:15:15	02-Aug-2024 09:35:55	610	630
Cleaning & Drying	12-Nov-2024 13:20:36	12-Nov-2024 13:40:22	886	300
Verification	12-Nov-2024 13:41:22	12-Nov-2024 13:57:12	630	320
Laser	12-Nov-2024 12:58:41	12-Nov-2024 13:19:36	335	920
Commissioning	12-Nov-2024 22:28:23	12-Nov-2024 23:31:13	2850	920
Powder coating	12-Nov-2024 23:32:13	13-Nov-2024 00:21:33	2360	600
Oven 1	13-Nov-2024 00:22:33	13-Nov-2024 01:04:03	1290	1200
Oven 2	13-Nov-2024 00:03:33	13-Nov-2024 00:03:33	1344	1200
Printer	13-Nov-2024 13:38:15	13-Nov-2024 15:58:38	7823	600
Montage	13-Nov-2024 15:59:38	13-Nov-2024 16:20:26	586	662
Electrical_Function_Verification	13-Nov-2024 16:21:26	13-Nov-2024 16:30:31	225	320
Packaging	13-Nov-2024 16:31:31	13-Nov-2024 16:36:06	75	200

Figure 4.11: Output table given after running each input in simulation.

The makespan calculated in Chapter 5 was inspired by the calculation in [DSA23]. Here, in Equation 4.2, θ represents the time stamp derived from each station's start and stop time, and T represents the output time for the analysis in hours. The calculation involves subtracting the time difference between sequential stations from the postponed station's finishing time by the previous station's start time.

$$T_i = \frac{T_{i-1} + \left(\theta_{\text{stop},m_2} - \theta_{\text{Start},m_1}\right)}{n} \tag{4.2}$$

The formulas for calculating MSE and MAE were provided in Sections 2.13 and 2.14 respectively. In Equations 4.4 and 4.3, the two methods are compared directly for each station's stop time or start time-based on the respective input. The mean for each station is then calculated.

$$\overline{T_{\text{time}}} = \frac{\sum_{x=1}^{n} \left(\theta_{x,\text{ trad}} - \theta_{x,\text{RL}}\right)}{n} \tag{4.3}$$

$$\overline{T_{\text{time}}} = \frac{\sum_{x=1}^{n} \left(\theta_{x,\text{ trad}} - \theta_{x,\text{RL}}\right)^2}{n} \tag{4.4}$$

4.3 Input Values

This section is dedicated to introducing and elucidating the tables utilized to model the flow factory environment, as delineated in Section 4.1.2. Due to the absence of real-world data for the simulation, most inputs were synthesized by the author, supplemented by references. The information provided in this section encompasses the production and transportation times for each machine per product type, supplier details including minimum order quantities and delivery times, the maximal storage capacities, and scenarios for order inputs. These scenarios take into account the initial conditions of all storage facilities and the required quantities of intermediary products needed for assembling the final product.

Table 4.4: Table with the production and transport time for each product on behalf of the machinery phase.

Product	Station	P.T. Face 1 (s)	P.T. Face 2 (s)	R.T. (s)	Consum raw material	unit
ProductA	Milling 1	90	45	600	0	-
ProductB	Milling 1	88	17	620	0	-
ProductC	Milling 1	85	30	600	0	-
ProductD	Milling 1	84	45	660	0	-
ProductE	Milling 1	92	29	630	0	-
ProductF	Milling 1	95	44	690	0	-
ProductA	Milling 2	93	35	630	0	-
ProductB	Milling 2	80	49	640	0	-
ProductC	Milling 2	75	56	680	0	-
ProductD	Milling 2	89	19	650	0	-
ProductE	Milling 2	95	28	720	0	-
ProductF	Milling 2	92	22	675	0	-
ProductA	Sawing	455	155	630	1	part
ProductB	Sawing	468	184	640	1	part
ProductC	Sawing	480	154	680	1	part
ProductD	Sawing	500	105	650	1	part
ProductE	Sawing	525	163	720	1	part
ProductF	Sawing	482	150	675	1	part
ProductA	Cleaning & Drying	886	0	300	0	-
ProductB	Cleaning & Drying	930	0	320	0	-
ProductC	Cleaning & Drying	826	0	360	0	-
ProductD	Cleaning & Drying	887	0	235	0	-
ProductE	Cleaning & Drying	950	0	290	0	-
ProductF	Cleaning & Drying	961	0	336	0	-
ProductA	Verification	630	0	320	0	-
ProductB	Verification	629	0	310	0	-
ProductC	Verification	660	0	325	0	-
ProductD	Verification	580	0	324	0	-
ProductE	Verification	650	0	320	0	-
ProductF	Verification	689	0	310	0	-
ProductA	Laser	335	0	920	1	L
ProductB	Laser	380	0	910	1	L
ProductC	Laser	430	0	915	1	L
ProductD	Laser	394	0	925	1	L
ProductE	Laser	346	0	910	1	L
ProductF	Laser	349	0	900	1	L
ProductA	Commissioning	1420	1430	920	0	-
ProductB	Commissioning	1354	1433	932	0	-
ProductC	Commissioning	1376	1359	863	0	-
ProductD	Commissioning	1439	1394	920	0	-
ProductE	Commissioning	1313	1368	983	0	-
ProductF	Commissioning	1433	1418	843	0	-
ProductA	Powder coating	1426	934	600	0	-
ProductB	Powder coating	1383	1005	620	0	-
ProductC	Powder coating	1426	1049	600	0	-
ProductD	Powder coating	1370	1110	660	0	-
ProductE	Powder coating	1319	1142	630	0	-
ProductF	Powder coating	1415	1114	690	0	-
ProductA	Oven 2	1344	0	1200	0	-
ProductB	Oven 2	1420	0	1200	0	-
ProductC	Oven 2	1230	0	1200	0	-
ProductD	Oven 2	1310	0	1200	0	-
ProductE	Oven 2	1439	0	1200	0	-
ProductF	Oven 2	1425	0	1200	0	-
ProductA	Oven 1	1290	0	1200	0	-
ProductB	Oven 1	1394	0	1200	0	-
ProductC	Oven 1	1350	0	1200	0	-
ProductD	Oven 1	1290	0	1200	0	-
ProductE	Oven 1	1422	0	1200	0	-
ProductF	Oven 1	1325	0	1200	0	-
base	Montage	586	0	662	0	-
base	Printer	7823	0	600	2	m
electrical_func	Electrical_Function_Verification	225	0	320	1	part
final_prod	Packaging	75	0	200	0	-

The Table referenced as 4.4 delineates the estimated production and transportation times, names as P.T. and R.T. in the table, respectively, associated with each product across various machines. These estimations are derived from a synthesis of data found within the existing literature. Specifically, the operation time for the milling machine is reported to range from 15 to 90 seconds. This variation is primarily attributed to the size and complexity of the workpiece and the specific characteristics of the milling machinery, as detailed in the literature [Ton+]. Furthermore, for other operational stations—namely sawing, cleaning and drying, verification, laser, commissioning, powder coating, oven, assembly, electrical function verification, 3D printer, and packaging—the estimations provided are principally based on the comprehensive review found in [McC16].
An exemplar workpiece was analyzed under a hypothetical framework to enhance comprehension of the concepts discussed in [McC16]. Specifically, a 100 cm^3 aluminum sample was utilized to approximate the production timeline across various manufacturing processes. The following provides a detailed breakdown of the estimated time allocations for each stage:

- **Sawing**: The process averages 10 minutes.
- **Cleaning and Drying**: Ten minutes are allocated for manual cleaning, while five minutes are required for drying with compressed air.
- **Verification**; This stage, which varies with the complexity of the components, is estimated to take 5 to 15 minutes.
- **Laser Marking**; Depending on the power of the machinery used, this can take between 2 to 10 minutes. A commercial stamp machine was used for estimation.
- **Commissioning**: Standard industrial practices suggest a range of 1 to 4 hours, though this may vary with component complexity.
- **Powder Coating**: The process is estimated to take 45 minutes to 1 hour, depending on factors such as surface preparation, powder application, curing, and cooling.
- **Oven (Normalization)**: Estimated at 1 to 3 hours, based on the normalization method applied.
- **Assembly**: Given the product's simplicity, with only five components, the minimum time reported is about 10 minutes.
- **3D Printing (Stereolithography, SLA)**: For the model of the 3D printer used in the flow factory, the time can range from 2 to 10 hours.
- **Electrical Component Assembly**: The distinction between automated and manual assembly is noted, with manual assembly estimated to take 5 to 15 minutes.
- **Packaging**: For simple manual packaging, the estimated time is between 1 to 5 minutes.

These time estimates were deliberately varied within certain intervals to affect the cost function analysis, as detailed in Section 2.9.

Table 4.5: Supplier Information about the machinery which has additional components.

Supplier	Type Raw material	Quantity Raw Material	Unit	Time to delivery (days)
Sawing	Sawing	10	parts	2
Laser	gas	10	L	1
Printer	Printer	10	m	1
Electrical_Function_Verification	electrical_func	5	parts	3

Table 4.5 presents the supplier information for the station sawing, laser, 3D printer, and electrical function verification equipment. The data within this table are subject to significant variability due to market fluctuations. For this study, it is assumed that the supplier will replenish the inventory to full capacity with each new delivery, notwithstanding a defined minimum order quantity for raw materials. According to findings from [24], the average delivery timeframe by UPS© in Germany ranges from one to two days. Consequently, this interval has been adopted as the estimated delivery period for this analysis.

Table 4.6: Maximum Capacity of the storages.

Name of storage	Type of Product	Max Storage	Unit
Main Storage	ProductA_complete_	30	parts
Main Storage	ProductB_complete_	30	parts
Main Storage	ProductC_complete_	30	parts
Main Storage	ProductD_complete_	30	parts
Main Storage	ProductE_complete_	30	parts
Main Storage	ProductF_complete_	30	parts
Main Storage	ProductA_notLasered_	30	parts
Main Storage	ProductB_notLasered_	30	parts
Main Storage	ProductC_notLasered_	30	parts
Main Storage	ProductD_notLasered_	30	parts
Main Storage	ProductE_notLasered_	30	parts
Main Storage	ProductF_notLasered_	30	parts
Sawing Storage	cutting	30	parts
Laser Storage	gas	30	L
3D Printer Storage	filament	30	m
Electrical Part Storage	electrical_func	30	parts
Montage Storages	base	30	parts

Table 4.6 delineates the maximum capacities for the storage units as elucidated in Figure 4.1. Two distinct storage modalities are identified within the primary storage facility: the first accommodates products awaiting laser stamping, whereas the second harbors products primed for commissioning. An estimated capacity of 30 units has been determined based on the reception of 50 input orders, as specified in Table 4.8. Consequently, storage replenishment is anticipated to occur once or twice, contingent upon the initial conditions depicted in Table 4.7.

Table 4.7: Storage initial Condition: Cases which contain the storage of each product in main storage and in the phase.

Cases	PA(c)	PB(c)	PC(c)	PD(c)	PE(c)	PF(c)	PA(nL)	PB(nL)	PC(nL)	PD(nL)	PE(nL)	PF(nL)	Sawing	Laser	Printer	EVF	Montage
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	30	30	30	30	30	30	30	30	30	30	30	30	30	30	20	30	30
3	10	10	10	10	10	10	0	0	0	0	0	0	10	10	5	15	10
4	0	0	0	0	0	0	20	15	10	20	10	18	20	15	0	4	1
5	0	2	3	7	10	2	10	2	1	0	0	0	0	0	10	0	10
6	30	30	30	30	30	30	30	30	30	30	30	30	0	0	0	0	30
7	0	0	0	0	0	0	0	0	0	0	0	0	30	20	20	30	0
8	2	4	1	2	1	3	10	20	1	1	2	3	10	8	15	14	20
9	28	24	1	1	0	30	0	0	0	10	20	30	12	5	0	0	30
10	30	0	30	0	0	0	0	30	0	30	30	30	0	0	0	0	5

The Table referenced as 4.7 presents the initial storage conditions across various stages, including sawing, laser processing, 3D printing, and electrical function verification. Additionally, it details the inventory levels of products in the main storage, categorizing them into two distinct types: those that are ready to proceed to assembly and those requiring prior processing through laser stamping. To execute the simulation, it utilizes the Table referenced as 4.8 to provide input for each initial storage condition. The algorithm's performance will subsequently be analyzed in Chapter 5.1.

Table 4.8: Orders List: input order about the manufacturing of final product.

Orders	ProductA	ProductB	ProductC	ProductD	ProductE	ProductF	base	electrical_func
1	1	1	1	1	0	0	1	1
2	1	0	1	1	1	0	1	1
3	1	0	0	1	1	1	1	1
4	1	0	1	0	1	1	1	1
5	1	0	1	1	0	1	1	1
6	1	1	0	1	1	0	1	1
7	1	1	0	1	0	1	1	1
8	1	1	0	0	1	1	1	1
9	1	1	1	0	0	1	1	1
10	0	1	1	1	1	0	1	1
11	0	0	1	1	1	1	1	1
12	0	1	0	1	1	1	1	1
13	0	1	1	0	1	1	1	1
14	0	1	1	1	0	1	1	1
15	0	1	2	1	0	0	1	1
16	4	0	0	0	0	0	1	1
17	0	4	0	0	0	0	1	1
18	0	0	4	0	0	0	1	1
19	0	0	0	4	0	0	1	1
20	0	0	0	0	4	0	1	1
21	0	0	0	0	0	4	1	1
22	0	2	2	0	0	0	1	1
23	2	0	0	0	2	0	1	1
24	2	0	0	0	0	2	1	1
25	0	2	0	2	0	0	1	1
26	0	0	2	2	0	0	1	1
27	0	0	0	0	2	2	1	1
28	0	0	2	0	2	0	1	1
29	3	0	1	0	0	0	1	1
30	3	1	0	0	0	0	1	1
31	0	0	0	3	0	1	1	1
32	0	0	0	0	3	1	1	1
33	0	0	2	0	1	1	1	1
34	0	2	1	1	0	0	1	1
35	0	2	1	0	0	0	1	1
36	3	1	0	0	0	0	1	1
37	1	3	0	0	0	0	1	1
38	0	1	0	3	0	0	1	1
39	0	0	0	1	0	3	1	1
40	0	1	0	2	1	0	1	1
41	0	0	0	0	1	3	1	1
42	0	3	0	0	0	1	1	1
43	0	1	0	0	0	3	1	1
44	0	0	0	1	3	0	1	1
45	0	0	3	0	1	0	1	1
46	0	0	3	1	0	0	1	1
47	0	0	3	0	0	1	1	1
48	0	0	1	3	0	0	1	1
49	0	1	0	3	0	0	1	1
50	1	0	3	0	0	0	1	1

The Table referenced as 4.8 delineates the characterization of each order, including detailed descriptions of the products required for assembly within the specified production chain. For the entirety of the simulation, varying initial conditions were systematically assigned as documented in Table 4.7, with each scenario being executed across a series of 50 products. In contrast to the study conducted by [Ton+], which utilized 1000 inputs, the current project employs 500 inputs. This reduction to 500 inputs was deemed satisfactory for the scope of this project, primarily aimed at evaluating the robustness of decision-making processes in response to three different occurrences.

5 Results and discussions

This section elaborates on the outcomes derived from executing the simulation delineated in Chapter 4. It is structured into two principal segments. The initial segment delves into the simulation results within an ideal environment, devoid of any external disturbances, thereby concentrating on the intrinsic performance of the algorithm. The subsequent segment examines the algorithm's resilience by introducing scenarios of resilience as outlined in Section 4.1.3. The analysis for both segments is grounded on the computational methodologies detailed in Section 4.2.3.

5.1 Comparison Methods Performance Analysis

This section will compare the performance of the RL algorithm with that of the benchmark algorithm, previously defined as a conventional method in Chapter 4. Section 3.3.2 reviewed relevant literature to evaluate how other research has compared performance based on outcome scores. The analysis and discussion of the results will focus on the following KPIs: makespan values, error analysis through the time difference between the two methods, and the computational effort required for each step of the RL application and the simulation of the flow factory environment. Additionally, supplementary graphics, including storage status analysis and detailed zoom-in graphics, have been incorporated to understand performance metrics better. To initiate the comparative analysis, the initial parameter to be examined is the duration required for the simulation to process all inputs outlined in Section 4.3. Each simulation commences on July 15, 2024, at 8:00 AM. The traditional methodology simulation concludes on December 14, 2024, at 4:36:06 PM, whereas the simulation employing the RL approach terminates earlier, on December 11, 2024, at 09:46:53 AM. This yields a temporal disparity of two days, signifying an enhancement in the algorithm's efficiency in orchestrating production processes during the simulation phase. Subsequently, the potential factors contributing to the RL algorithm's performance improvement will be investigated.
The chart in Figure 5.1 shows the KPI makespan calculated using the equation presented in Equation 4.2 for the simulation that used the RL algorithm for the initial case where no issues occurred. Upon initial observation, it is clear that there is a significant difference between the laser and sawing processes. This difference arises because the AI application can parallelize sub-processes, such as "milling to sawing to cleaning & drying to verification, then store sub-product," optimizing the final simulation value. This optimization allows for the postponement of sub-products to be stored before the process characterized by the second Petri net model shown in Figure 4.4. This production optimization gains time for the overall simulation, as other sub-processes can easily use products available in the main storage. It is important to note that cleaning & drying and verification are common for both the "milling to sawing to cleaning & drying to verification, then store sub-product" and "laser to cleaning & drying to verification, then store product" sub-processes. Therefore, in the simulation, the start and stop times are assumed to be the values for the last sub-process. The 3D printer station was ignored in the results because it ran in parallel to the serialized process and was therefore not discussed.

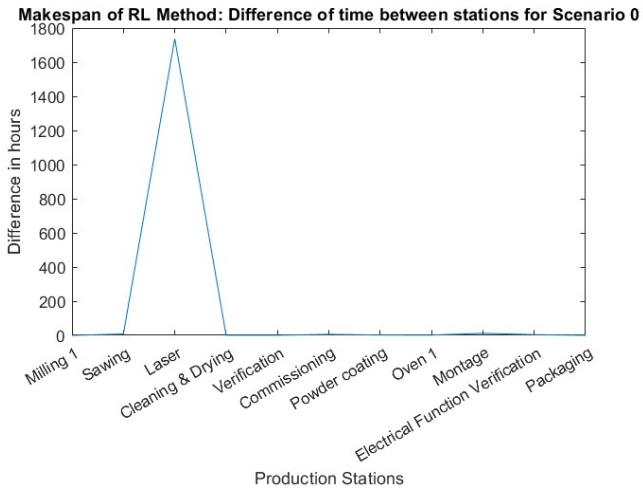

Figure 5.1: RL Algorithm: Time difference between start and stop times between stations.

Figure 5.2 shows a zoomed-in graph of Figure 5.1, specifically after the cleaning & drying station. The simulation assumes a serialized behavior for the products, requiring that they start from the lasering phase. Based on the figures, the start and stop times of the stations are not expected to have significant changes when the product enters the Petri net model shown in Figure 4.5. This is because the product must follow the process without any process being able to advance. An interesting observation about the algorithm is that it schedules the sub-process "milling-sawing-cleaning & drying-verification" in advance but not the sub-process "laser-cleaning & drying-verification," as shown in the figures. This can be understood because the second sub-process is strictly dependent on the first sub-process, so it can not run independently. Therefore, it can be concluded that based on the KPI makespan, this RL algorithm can effectively parallelize subprocesses while the primary process is being executed.

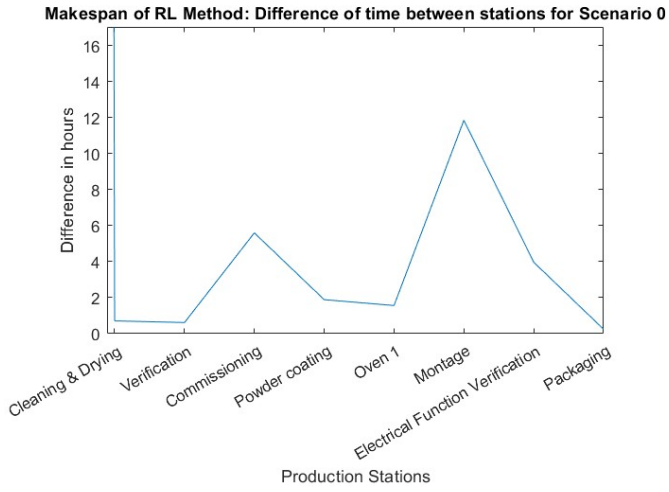

Figure 5.2: RL Algorithm: Time difference between start and stop times between stations with zoom after Clean & Drying Station.

When analyzing the Figure 5.3, it is evident that the storage of the sub-products (A, B, C, D, E and F) destined for the lasering phase is depicted. The storage begins at a certain point and rapidly reaches its

maximum capacity due to each input's parallelization of the Petri nets models. It was determined that the storage must be initialized at a specific point for every 50 inputs, which explains the drops in the graph followed by increases, as the software promptly detects the need to produce more components for storage.

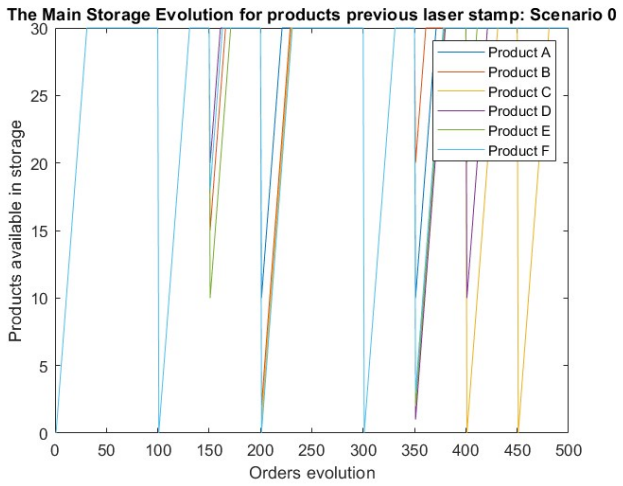

Figure 5.3: Main Storage behavior for the products that are going to be forwarded for the lasering stamp phase

When analyzing the Figure 5.4 it is evident that the process is serialized with two stages, allowing for sub-product storage. The process is serialized because, for each input, one of the Petri models discussed in Section 4.2.1 is chosen based on the availability of sub-products in the storage. If components A, B, C, D, E, and F are present in the main storage, the simulation will proceed only to the assembly phase; otherwise, it will produce the components when the storage is empty. The two stages are present because components are initially available as input, but manufacturing becomes necessary at some point.

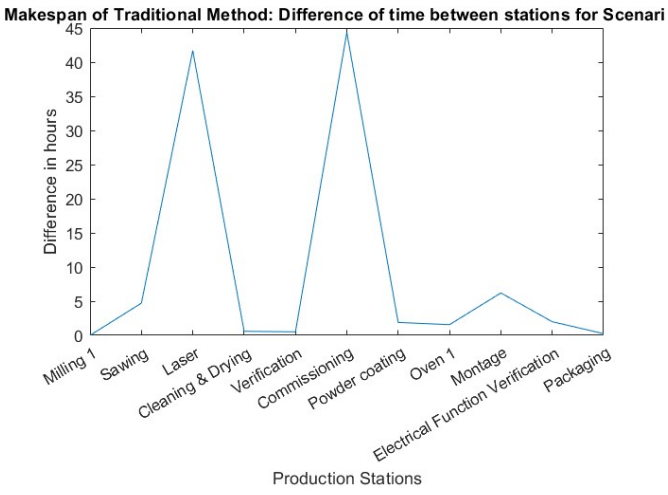

Figure 5.4: Traditional Method: Time difference between start and stop times between stations.

To evaluate the KPIs MSE and MAE, calculated as presented in Section 4.2.3, Figure 5.5 illustrates the difference in start times between the RL algorithm and the traditional method. The chart shows a significant

cumulative error for the initial sub-process, "milling-sawing–cleaning & drying-verification," attributed to the parallelization previously discussed. In the traditional method, products were manufactured only when the storage was empty, resulting in a substantial delay compared to the RL algorithm, where products were manufactured in advance and stored. The subsequent phases of laser exhibited low cumulative error. cleaning & drying and verification maintained low errors as they considered the values after lasering the stamp.

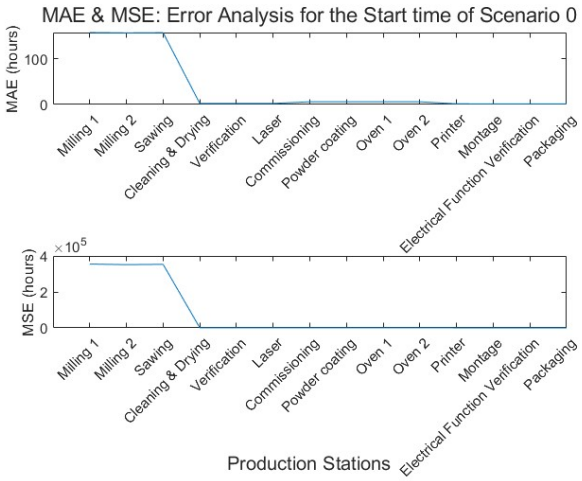

Figure 5.5: MSE and MAE between the start time difference between the RL and Traditional Method

Figure 5.6 shows similarities to Figure 5.5. A zoomed-in view of the error behavior is presented in Figure 5.7, presenting the station for 'cleaning & drying' and more.

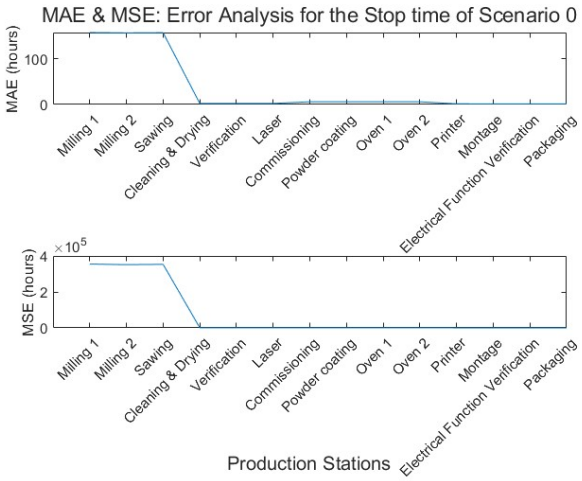

Figure 5.6: MSE and MAE between the stop time difference between the RL and Traditional Method

In Figure 5.7, it can be observed the printer station was not evaluated based on the KPI makespan. It's evident that the printer has a low error. Therefore, the station was executed in similar situations by both models, without parallelization by the RL algorithm. The conclusion is that in this case, the printing process did not attempt to create new "bases" in advance.

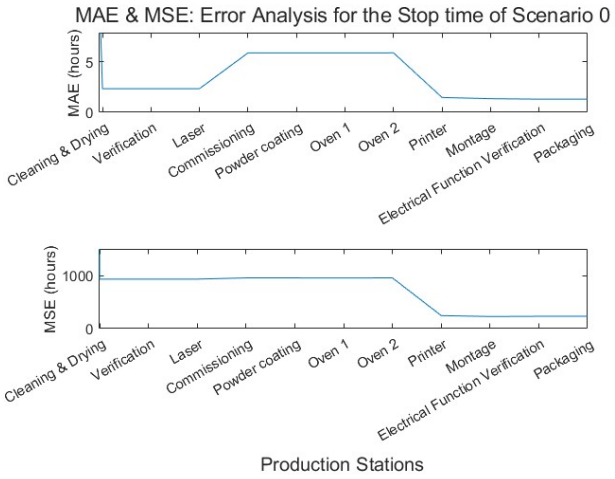

Figure 5.7: MSE and MAE between the stop time difference between the RL and Traditional Method with Zoom from the station Cleaning & Drying

The last KPI to consider is the computational effort discussed in Section 3.3.2. This KPI involves analyzing the time it takes to run each RL application, as explained in Section 4.2.2, as well as the time it takes to run the simulation. The RL algorithm also includes a third AI application, which increases the simulation time. Figure 5.8 illustrates the time evolution over the input evolution. The *prioritization algorithm* rises in steps because it is called every 50 inputs (when the initial condition is changed), while the other two algorithms increase linearly with the input evolution. The *Petri net model selection algorithm* significantly impacts the simulation time.

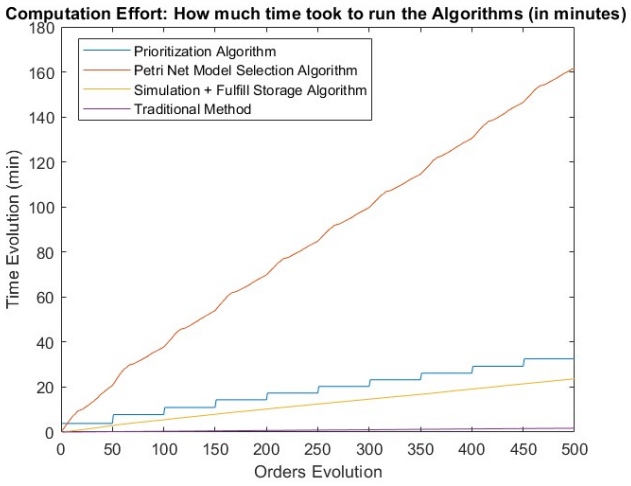

Figure 5.8: Computational Effort: Comparison between RL and Traditional Methods, and examining each RL model presented in Section 4.2.2

In the Figure 5.9, it is possible to observe that the traditional method shows a linear increase in simulation time, taking significantly less time compared to the RL algorithm. Specifically, for this simulation, the RL algorithm took 217.99 minutes, while the traditional method took only 1.72 minutes.

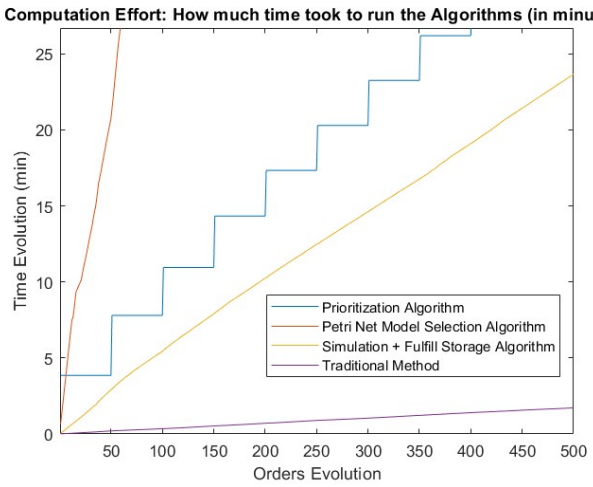

Figure 5.9: Computational Effort: Comparison between RL and Traditional Methods, and examining each RL model presented in Section 4.2.2 with zoom

The hardware configuration used for these simulations contains Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz, 16 GB of RAM, and NVidia GeForce RTX 3070. Therefore, with a better hardware configuration, the RL algorithm can improve efficiency to run.

5.2 Comparison Methods Performance Analysis

This section will assess the resilience of the RL algorithm compared to traditional methods. It evaluated three scenarios outlined in Section 4.1.3: storage disruption, machinery malfunction, and workforce disruption. The main argument presented in Section 5.1 is that the RL algorithm's ability to parallelize sub-processes is the key difference for future improvements in the simulation.

5.2.1 Scenario 1: Storage disruption

The first scenario simulated a situation where the storage could not operate at its maximum capacity. It was established that both algorithms are not real-time solutions as a potential factor during the simulation. It was determined that during the simulation, the storage could only store a maximum of 10 external components at the 3D printer stage for orders with indexes from 25 to 30; see Table 4.8. Using the traditional method, we expected an increase in the total simulation time, and indeed, the simulation finished on December 18, 2024, at 10:51:34 am. On the other hand, for the RL algorithm, we anticipated that it would mainly influence the *Fulfill Storage algorithm*, as it determines when new deliveries should be made. The simulation for this method ended on December 14, 2024, at 02:09:57 am. This demonstrates that AI has improved performance and handles occurrences better than the traditional method.
Figure 5.10 illustrates the performance of the KPI makespan for the RL algorithm in the first resilience scenario. A notable observation is that there are now two peaks in the graph as opposed to just one, as depicted in Figure 5.1. This difference is due to the software also halting the sub-process "laser-cleaning & drying-verification," in anticipation of the completion of components for use in the montage sub-process. It is also evident that the 3D printer station does not cause significant disruptions between oven 1 and montage, indicating minimal impact on the simulation. It can be concluded that the RL algorithm was able

to anticipate disruptions and adjust the preceding sub-process to optimize simulation time in instances where the stations were not functioning properly.

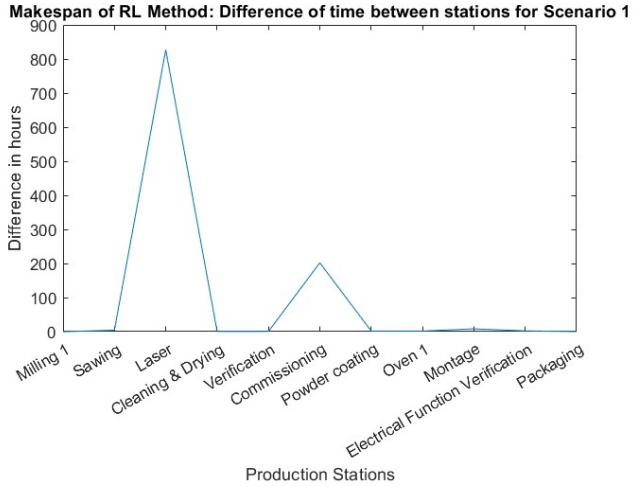

Figure 5.10: Scenario 1: RL Algorithm: Time difference between start and stop times between stations.

The trend shown in Figure 5.11 is similar to that in Figure 5.4. It's evident that the simulation exhibits three noticeable time discrepancies: between the sub-processes "milling-sawing-cleaning & drying-verification" and "laser-cleaning & drying-verification". These differences are due to the availability of component storages for the sub-products of these sub-processes. The third peak is caused by the influence of the 3D printer, which could lead to a delay in the simulation time between the oven and montage stations.

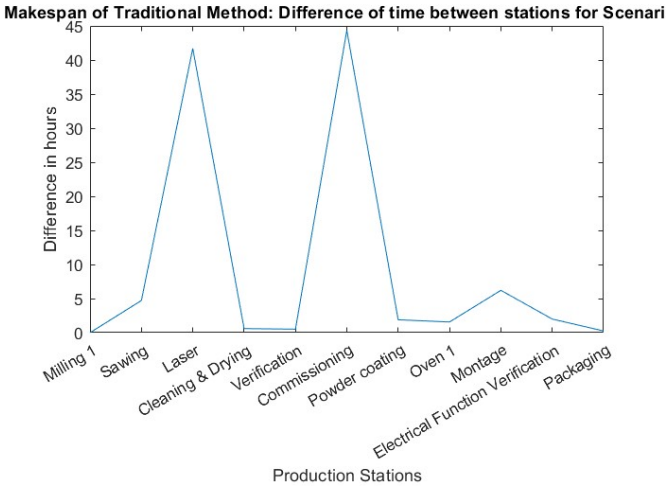

Figure 5.11: Scenario 1: Traditional Method: Time difference between start and stop times between stations.

To understand what happens with the 3D printer station, particularly with its inventory, refer to Figure 5.12. The notable observation is that the algorithm allows the inventory to reach a low capacity, but not zero, before ordering a new delivery. It is also observed that the algorithm anticipates disruptions and plans for the inventory to work with less than ten available components during five disruptive inputs. This

is evident when new deliveries are made, and the storage is not filled to maximum capacity, but rather to a lower capacity to avoid compromise during these special inputs. This behavior is only achievable because the inputs learn from previous inputs, and the RL algorithm makes decisions according to what happened in the training phase.

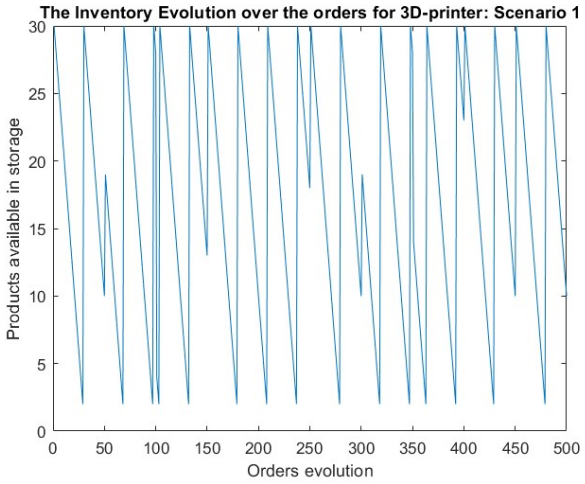

Figure 5.12: 3D-printer Inventory Evolution over the inputs scheduling.

In the graphs shown in Figure 5.13, the KPIs MAE and MSE indicate that further analysis of the 3D printer phase is possible. It is evident that for both algorithms, the cumulative error between them was low compared to other stations. This suggests that the RL algorithm did not directly optimize the simulation for the 3D printer station, but rather for the preceding sub-processes by parallelizing them. The step tendency arises from the fact that components are manufactured in advance in the RL algorithm, leading to a significant time difference in product storage between the algorithms.

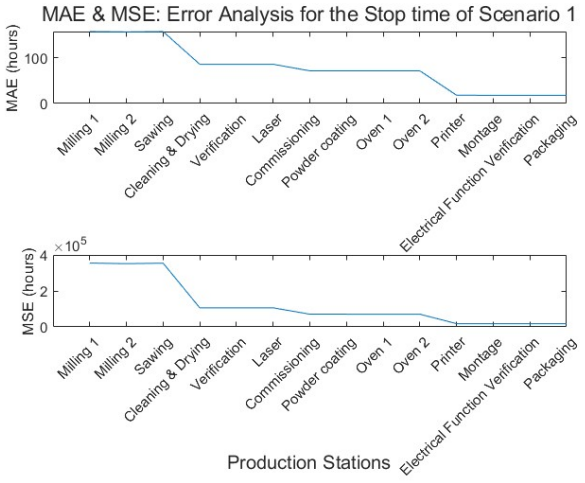

Figure 5.13: Scenario 1: MSE and MAE between the stop time difference between the RL and Traditional Method.

By the Figure 5.14 it is visible the increase of the computational effort in comparison to the Figure 5.8. The most probable argumentative cause would be external interference during the simulation, caused mainly by parallel usage of the hardware of other software applications while the simulation ran. The total time to run the RL method was 534.42 minutes, while the traditional method took only 1.76 minutes.

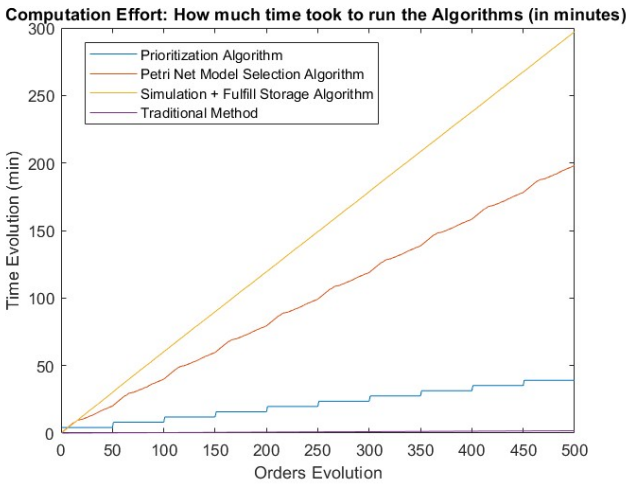

Figure 5.14: Scenario 1: Computational Effort: Comparison between RL and Traditional Methods, and examining each RL model presented in Section 4.2.2.

In further analysis of the graph in Figure 5.14, it is evident that the simulation time was the period during which the calculations took the longest to run. This indicates that the disruptive scenario had an impact on the decision-making process of the *Fulfill Storage Algorithm*, leading to increased computational effort in making decisions during the simulation.

5.2.2 Scenario 2: Machinery Malfunction

The second scenario simulates a potential machinery malfunction during production, as discussed in Section 4.1.3. The malfunction happens at the cleaning & drying station while processing orders 10 to 15 (refer to Table 4.8 for the specific order indexes). This directly impacts production time, which is adjusted by tripling the original production time (see Figure 4.4 for station production times).
The first step involves comparing the end times for simulating inputs using traditional and RL methods. The traditional method resulted in an end time of January 5, 2025, at 01:39:01 am, while the RL method resulted in an end time of December 13, 2024, at 05:03:38 pm. It's evident that the algorithm showed improved performance after nearly one month of optimization, indicating the efficient resilience of the RL algorithm for this scenario.
Figure 5.15 depicts two graphs. The first graph illustrates the KPI makespan of the total production process simulation, showing a high peak between the sawing and laser stages, indicating that this sub-process was optimized due to parallelization. The second graph demonstrates that the "laser-cleaning & drying-verification" sub-process was slowly optimized, as indicated by the low slope between the verification and commissioning stages. Another point of interest is the slope between the oven and montage stages, which shows the execution of the 3D printer. However, when compared with the graph in Figure 5.2, no significant difference is noticeable.

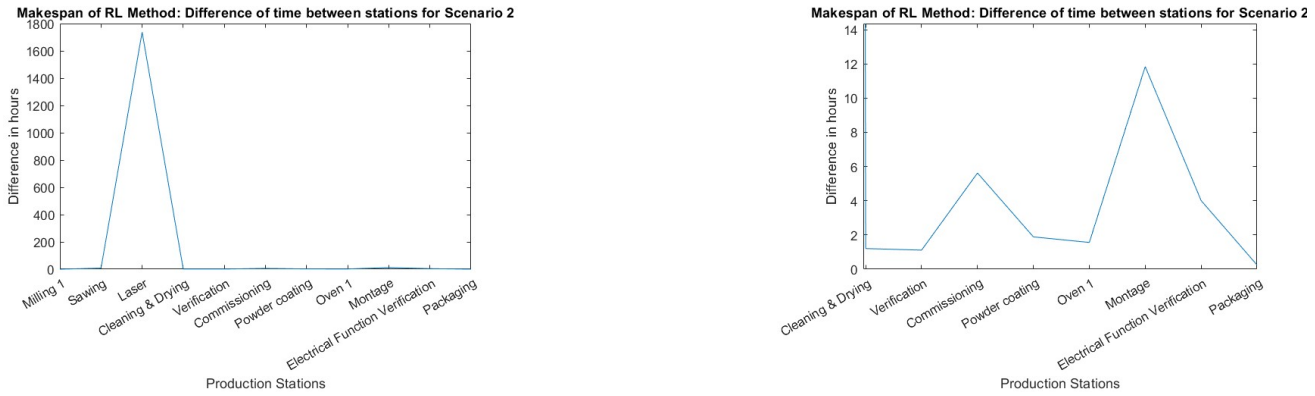

Figure 5.15: Scenario 2: RL Algorithm: Time difference between start and stop times between stations.

In Figure 5.16, the KPI makespan of the traditional method is depicted. A notable increase is observed between the sawing and laser phases compared to Figure 5.4. This increase is due to a delay in the sub-process "milling-sawing-cleaning & drying-verification," which goes through cleaning & drying once, causing a delay in forwarding the component to laser. This delay slightly affects the verification and commissioning phases, resulting in an increase in the peak compared to the ideal scenario.

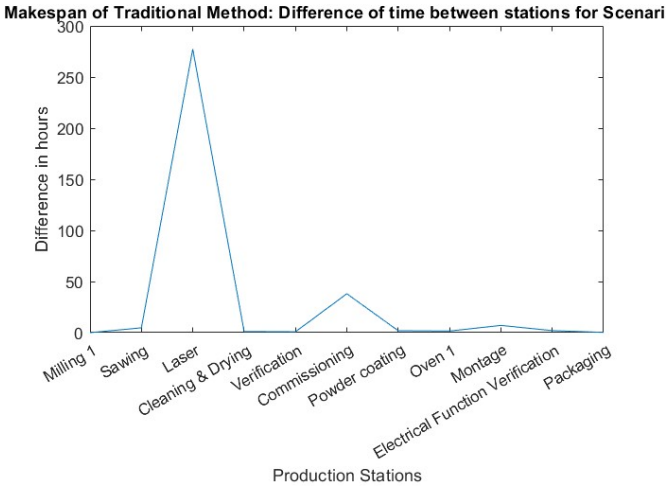

Figure 5.16: Scenario 2: Traditional Method: Time difference between start and stop times between stations.

The graph in Figure 5.17 demonstrates that the KPIs MSE and MAE exhibit a similar trend to what is shown in Figure 5.6, which represents the ideal scenario without disruptions. Despite the nearly one-month gap in the end time, both methods operated with a similar trend during the simulation until the operations were parallelized in the first sub-process. This indicates that the simulation was quite similar in the beginning and middle stages, and the traditional method took longer to complete towards the end.

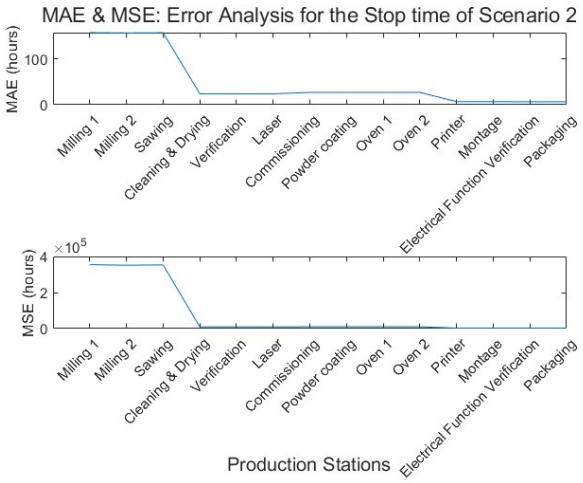

Figure 5.17: Scenario 2: MSE and MAE between the stop time difference between the RL and Traditional Method.

Figure 5.18 can be compared to the behavior of the ideal scenario without any disruptions. However, in scenario one (storage disruption), external influences increased the computational effort to simulate the AI applications; in this simulation, the computational effort remained similar to the comparison in Section 5.1. For this scenario with machines malfunctioning, the traditional method took 1.72 minutes to run, while the RL method took 262.28 minutes.

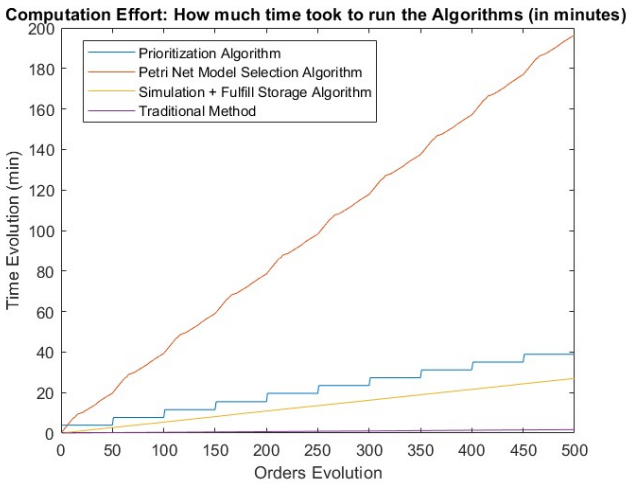

Figure 5.18: Scenario 2: Computational Effort: Comparison between RL and Traditional Methods, and examining each RL model presented in Section 4.2.2.

Despite the high computational effort required to generate the AI simulation, it has shown better resilience than the traditional method in simulating machinery malfunctioning for this scenario. As a result, it is recommended for simulating solutions for problems with similar roots and parameters.

5.2.3 Scenario 3: Workforce disruption

The third scenario simulates a situation where there is a disruption in the workforce during the simulation, as described in Section 4.1.3. This labor disruption was simulated by directly interfering with the transport time, as defined in the project's hypothesis in Section 4.1.1. It was explicitly related to the transport time of the station called milling 1. Between order indexes 43 to 47, the transport time was three times higher than the original duration.
To compare the two methods, the end times of each simulation need to be analyzed. The traditional method concluded on January 15th at 09:46:53 a.m., while the RL method concluded on December 13th at 04:36:06 p.m. This demonstrates that the AI method performed better, finishing more than a month earlier than the traditional method.
In Figure 5.19, the makespan for the RL algorithm in the third scenario is shown. The graph follows a similar trend to the one in Figure 5.1. As a result, it can be drawn conclusions about the parallelization of the first sub-process. Due to the longer transport time in the milling 1 station, it has resulted in a negative makespan between milling 1 and sawing. The likely reason for the negative makespan is that sub-products A, B, and C are manufactured in milling 1, while products D, E, and F are manufactured in milling 2. As a result, several products from milling 2 were manufactured in advance, causing the start time in sawing to be before milling 1, resulting in the negative makespan, which does not take into account the values of milling 2.

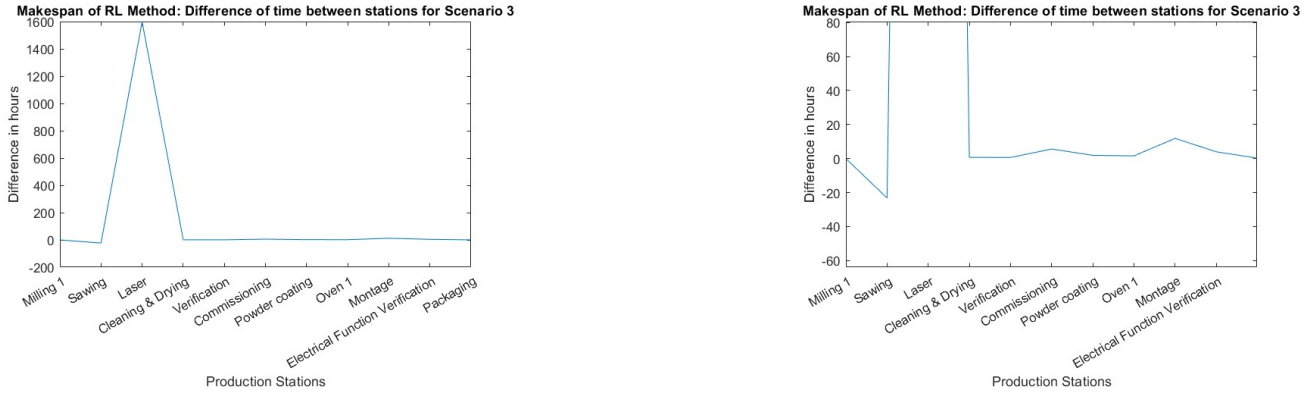

Figure 5.19: Scenario 3: RL Algorithm: Time difference between start and stop times between stations.

Figure 5.20 depicts the trend in makespan values for the traditional method. This trend aligns with the simulation in the scenario without disruptions (refer to Section 5.1). While the transport time has increased for milling 1 under certain conditions, there is no noticeable difference compared to Figure 5.4, especially in the makespan value between the milling 1 and sawing stations.

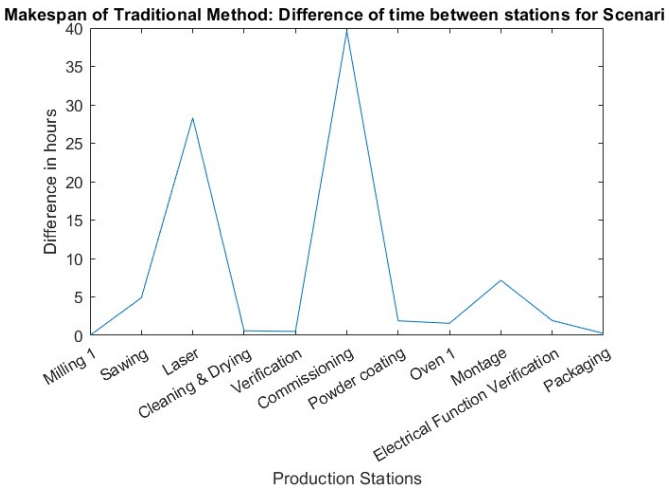

Figure 5.20: Scenario 3: Traditional Method: Time difference between start and stop times between stations.

After analyzing the KPI MSE and MAE for the third scenario, as shown in Figure 5.21, it can be concluded that the behavior is similar to what was found for scenarios one and two (Sections 5.2.1 and 5.2.2). Therefore, the same discussion made in these sections can be applied here. An interesting point to note is the parabolic tendency over the stations milling 1, milling 2, and sawing, demonstrating that the time difference between the RL and traditional methods was slightly lower for milling 1. However, due to the manufacturing of the sub-products at the beginning of the simulation, the MSE and MAE are still high compared to the rest of the simulation. The errors were more cumulative in the milling 2 station, which did not experience any disruptions. As a result, the RL algorithm was not as well optimized in advance compared to the similar station that did not encounter any further issues.

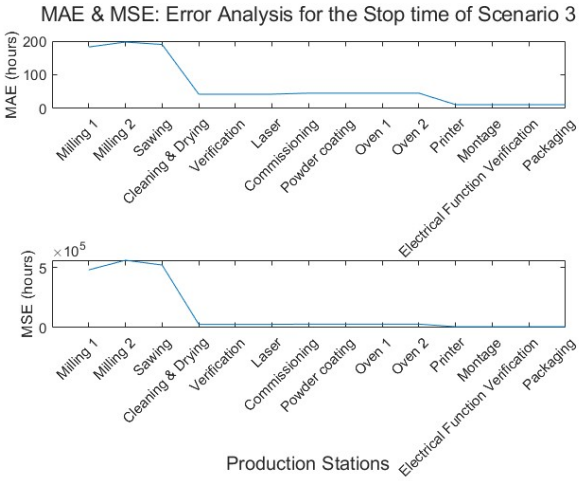

Figure 5.21: Scenario 3: MSE and MAE between the stop time difference between the RL and Traditional Method.

The behavior of the scenario shown in Figure 5.22 is similar to the one presented in Section 5.1. Therefore, it can be drawn the same conclusions as before. In this scenario, the total simulation time for the traditional and RL methods was 1.61 and 267.67 minutes, respectively. Thus, no external influences were detected.

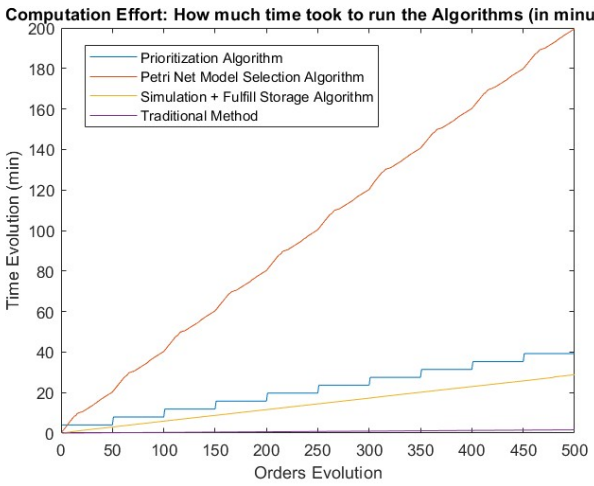

Figure 5.22: Scenario 3: Computational Effort: Comparison between RL and Traditional Methods, and examining each RL model presented in Section 4.2.2.

The analysis of the three scenarios concluded that the RL method has demonstrated better resilience compared to the traditional method. It has also resulted in postponed processes and lower stop times in the simulations.

6 Conclusion

The research aimed to develop an RL-based decision algorithm to enhance the resilience of order scheduling within an industrial environment. The study utilized the laboratory environment of the FlowFactory at the PTW of TUDa as a case study. An initial literature review was conducted to achieve this goal, focusing on scientific applications of RL within manufacturing and order scheduling domains. The review helped identify the most appropriate RL method and KPIs to evaluate performance against a traditional scheduling approach.

The Q-learning algorithm was chosen as the RL method due to its effectiveness in decision-making tasks and the simplicity of its application. At the same time, Petri Net models were selected as the traditional approach for comparison. These methods were then applied to develop a simulated environment of the FlowFactory, which enabled replicating real-world conditions and inputs related to order scheduling. The simulation environment was designed to handle dynamic inputs, such as varying production orders and inventory levels, allowing the algorithm to make proactive decisions based on changing scenarios.

Three distinct RL-based environments were created, the *Prioritization algorithm*, *Petri nets model selection algorithm* and *Fulfill storage algorithm* to explore various decision-making strategies during the simulation. The two first models aimed to organize incoming orders, while the last requested dynamic new stock replenishment for inventory. The goal was to optimize scheduling performance by minimizing downtime and maintaining a balanced flow of operations. Through this process, the RL algorithm demonstrated a significant capacity to handle parallel sub-processes, optimizing and anticipating production needs within the chain. Consequently, the proposed RL method resulted in higher stock availability and improved efficiency than traditional scheduling methods.

The experimental results were promising, showing that the RL-based approach outperformed traditional methods regarding resilience and responsiveness to changes in the production environment. The RL algorithm shortened simulation times and enhanced overall production throughput. These improvements were attributed to the algorithm's ability to parallelize decision-making and anticipate resource shortages, which minimized delays and optimized order completion rates. The resilience of the RL approach was further demonstrated through its adaptability to unforeseen changes, allowing it to maintain operational stability.

Despite these positive outcomes, several aspects require further refinement. Future research should focus on integrating additional production parameters, such as machine-specific constraints, dynamic inventory control, and fixed working hours, to create a more comprehensive model. The implementation of other RL algorithms, such as SARSA and DQN, is also welcome as future work to generate a better performance comparison. Additionally, the algorithm's performance in a real-time environment remains a critical area for investigation since the algorithm evaluates all orders that will be simulated before it is simulated.

In conclusion, the proposed RL-based decision algorithm has demonstrated the potential to enhance the resilience and efficiency of order scheduling in complex industrial environments. However, its application in real-world scenarios necessitates further validation and refinement. Incorporating more detailed production variables, checking other RL algorithms, and testing in real-time environments will be crucial next steps to confirm the algorithm's practical value and scalability.

Bibliography

[24] create a shipment. 2024. URL: https://www.ups.com/ship/basic/smallbusiness?tx=08808500012385492&loc=en_DE&WT.mc_id=EsbsToFWSRedirect.

[AFI17] Alexandru Iulian Orhean, Florin Pop, and Ioan Raicu. "New scheduling approach using reinforcement learning for heterogeneous distributed systems". In: *J. Parallel Distrib. Comput.* (2017).

[And20] Kuhnle Andreas. "Adaptive order dispatching based on reinforcement learning: application in a complex job shop in the semiconductor industry. Dissertion". MA thesis. Technische Universität Darmstadt, 2020.

[ARA22] Anand Singh Rajawat, Omair Mohammed and Rabindra Nath Shaw, and Ankush Ghosh. "Renewable energy system for industrial internet of things model using fusion-AI". In: *Applications of AI and IOT in Renewable Energy.* (2022). DOI: https://doi.org/10.1016/B978-0-323-91699-8.00006-1.

[AS90] A.G. Barto and S.E Singh. "On the computational economics of reinforcement learning. In D.S. Touretzky, J. Elman, T.J. Sejnowski G.E. Hinton, (Eds.)," in: *Proceedings of the 1990 Connectionist Models Summer School San Mateo, CA: Morgan Kaufmann.* (1990).

[BAB21] Bruno Cunha, Ana Madureira, and Benjamim Fonseca and João Matos. "Intelligent Scheduling with Reinforcement Learning". In: *Appl. Sci. 2021, 11, 3710* (2021). DOI: https://doi.org/10.3390/app11083710.

[Bel54] R. Bellman. "The theory of dynamic programming". In: *Bulletin of the American Mathematical Society, 60(6), 503-515* (1954). DOI: https://doi.org/10.1090/S0002-9904-1954-09848-8.

[BG20] Behice Meltem Kayhan and Gokalp Yildiz. "Reinforcement learning applications to machine scheduling problems: a comprehensive literature review". In: *Journal of Intelligent Manufacturing (2023) 34:905–929* (2020). DOI: https://doi.org/10.1007/s10845-021-01847-3.

[Bhu+20] Bhushan Pawar et al. "Applications of resilience engineering principles in different fields with a focus on industrial systems: A literature review". In: *Journal of Loss Prevention in the Process Industries* (2020). DOI: https://doi.org/10.1016/j.jlp.2020.104366.

[BN01] Robert W. Brennan and Douglas H. Norrie. "Evaluating the performance of reactive control architectures for manufacturing production control". In: *Computers in Industry 46 (3), p. 235-245* (2001). DOI: 10.1016/S0166-3615(01)00108-7.

[Col12] Allan Collard-Wexler. "Production and Cost Functions". MA thesis. 2012. URL: https://pages.stern.nyu.edu/~acollard/productivity.pdf.

[CP04] Martin Christopher and Helen Peck. "Building the Resilience Supply Chain". In: *International Journal of Logistics Management 15, P. 1-13*. 2004. DOI: 10.1108/09574090410700275.

[DBW91] D.M. Dilts, N.P. Boyd, and H.H. Whorms. "The evolution of control architectures for automated manufacturing systems". In: *Journal of Manufacturing Systems 10 (1), p. 79-93* (1991). DOI: 10.1016/0278-6125(91)90049-8.

[Dec24] Alexander Decher. “Systematischer Literaturrecherche zur Integration von Resilienz in die autonome Produktionssteuerung”. PhD thesis. Technische Universität Darmstadt, 2024.

[DSA23] David Tremblet, Simon Thevenin, and Alexandre Dolgui. “Makespan estimation in a flexible job-shop scheduling environment using machine learning”. In: *ARTICLE TEMPLATE* (2023).

[FK24] Fadi AlMahamid and Katarina Grolinger. “Reinforcement Learning Algorithms: An Overview and Classification”. In: *Department of Electrical and Computer Engineering Western University* (2024). DOI: 0000-0002-6907-76260000-0003-0062-8212. URL: https://ar5iv.labs.arxiv.org/html/2209.14940.

[Gri] Michael Grieves. *Origins of the Digital Twin Concept: Unpublished*. 2016.

[Hao+22] Hao Li et al. “DoSimpler Statistical Methods Perform Better in Multivariate Long Sequence Time-Series Forecasting?” In: *Association for Computing Machinery* (2022). DOI: https://doi.org/10.1145/3511808.3557585.

[JBD21] Janis Brammer, Bernhard Lutz, and Dirk Neumann. “Permutation flow shop scheduling with multiple lines and demand plans using reinforcement learning”. In: *European Journal of Operational Research* (2021).

[JP92] J.C.H. CHRISTOPHER and PETER DAYAN WATKINS. “Technical Note: Q-Learning”. In: *Kluwer Academic Publishers, Boston. Manufactured in The Netherlands. Machine Learning, 8, 279-292* (1992).

[Len+23] Jiewu Leng et al. “Towards resilience in Industry 5.0: A decentralized autonomous manufacturing paradigm”. In: *Journal of Manufacturing Systems 71, P. 95-114*. 2023. DOI: 10.1016/j.jmsy.2023.08.023.

[Lih+11] Lihn T.T. Dinh et al. “Resilience engineering of industrial processes: Principles and contributing factors”. In: *Journal of Loss Prevention in the Process Industries* (2011). DOI: 10.1016/j.jlp.2011.09.003.

[Luo20] Shu Luo. “Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning”. In: *Applied Soft Computing Journal* (2020).

[MAY20] Michael W. Berry, Aylinah Hj. Mohamed, and Yap Bee Wah. “Supervised and unsupervised learning for data science”. In: *Springer (Unsupervised and semi-supervised learning)* (2020).

[McC16] Christopher J. McCauley. *Machinery’s Handbook*. Industrial Press Inc.,U.S., 30ª edição (21 abril 2016), 2016.

[MKH88] M. Sato, K. Abe, and H. Takeda. “Learning control of finite Markov chains with explicit trade-off between estimation and control.” In: *IEEE Transactions on Systems, Man and Cybernetics, 18, pp. 67-684.* (1988).

[Muk+22] Avik Mukherjee et al. “Designing Resilient Manufacturing Systems using Cross-Domain Application of Machine Learning Resilience”. In: *Procedia CIRP 115, p. 83-88* (2022). DOI: 10.1016/j.procir.2022.10.054.

[Mur89] Tadao Murata. “Petri Nets: Properties, Analysis and Applications”. In: *PROCEEDINGS OF THE IEEE, VOL. 77, NO. 4, APRIL 1989* (1989).

[Mus+17] Mustafa Seçkin Durmuş et al. “Enhanced V-Model”. In: *Informatica 42 (2018) 577–585* (2017). DOI: https://doi.org/10.31449/inf.v42i4.2027.

[Naj+23] Najeeb Alam Khan et al. “Accruement of nonlinear dynamical system and its dynamics: electronics and cryptographic engineering”. In: *Fractional Order Systems and Applications in Engineering* (2023). DOI: https://doi.org/10.1016/B978-0-32-390953-2.00015-3.

[NJ22] Frick Nicholas and Metternich Joachim. “The Digital Value Stream Twin”. In: *Systems 10 (4), p. 102* (2022). DOI: 10.3390/systems10040102.

[PU11] Sultana Parveen and Hafiz Ullah. “Review on Job-Shop and Flow-Shop Scheduling using In: J. mech. eng.” In: *41 (2), p. 130-146* (2011). DOI: 10.3329/jme.v41i2.7508.

[QL21] Qin Zhaojun and Lu Yuqian. “Self-organizing manufacturing network: A paradigm towards smart manufacturing in mass personalization”. In: *Journal of Manufacturing Systems 60, p. 35-47* (2021). DOI: 10.1016/j.jmsy.2021.04.016.

[RA20] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction (2nd ed.)*. The MIT Press., 2020.

[Rat11] A. Ratcliffe. “SAS Software Development with the V-Model”. In: *3SAS Global Forum, Coder’s Corner, 4-7 April, Las Vegas, Nevada, USA, pp. 1-9* (2011).

[Ris23] Rafael Ris-Ala. *Fundamentals of Reinforcement Learning*. Springer, 2023.

[Rol06] Sturm Roland. “Modelbasirtes Verfahren zur Online-Leistungsbewertung von automatisierten Transportsystemen in der Halbleiterfertigung”. MA thesis. Unter Mitarbeit von Universität Stuttgart, 2006.

[RS62] R.E. Bellman and S.E. Dreyfus. *Applied dynamic programming.* RAND Corporation., 1962.

[SAS21] Sebastian Schwendemann, Zubair Amjad, and Axel Sikora. “A survey of machine-learning techniques for condition monitoring and predictive maintenance of bearing in grinding machines”. In: *Computers in Industry 125, p.103380* (2021). DOI: 10.1016/j.compind.2020.103380.

[SGI24] Seongbae Jo, Gyu M. Lee, and Ilkyeong Moon. “Airline dynamic pricing with patient customers using deep exploration-based reinforcement learning”. In: *Engineering Applications of Artificial Intelligence* (2024). DOI: https://doi.org/10.1016/j.engappai.2024.108073.

[Sha+23] Shaojun Lu et al. “A Double Deep Q-Network framework for a flexible job shop scheduling problem with dynamic job arrivals and urgent job insertions”. In: *Engineering Applications of Artificial Intelligence* (2023). DOI: https://doi.org/10.1016/j.engappai.2024.108487.

[Sin93] N. Singh. “Design of cellulcar manufacturing systems: An invited review”. In: *European Journal of Operational Research 69 (3), p. 284-291* (1993). DOI: 10.1016/0377-2217(93)90016-G.

[SM23] Maximilian Steinmeyer and Joachim Metternich. “Resilienz aus der Wertstromperspektive”. In: *118 (9), p. 605-609* (2023). DOI: 10.1515/zwf-2023-1123.

[SR22] Ananya Sheth and Andrew Rusiak. “Resiliency of Smart Manufacturing Enterprises via Information Integration”. In: *Journal of Industrial Integration 28, P. 100370*. 2022. DOI: 10.1016/j.jii.2022.100370.

[Sum+24] Sumin Hwangbo et al. “Production rescheduling via explorative reinforcement learning while considering nervousness”. In: *Computers and Chemical Engineering* (2024).

[Sut84] R.S. Sutton. “Temporal credit assignment in reinforcement learning”. MA thesis. PhD Thesis, University of Massachusetts, Amherst, MA, 1984.

[SYL23] Sirui Chen, Yuming Tian, and Lingling An. “Multi-Objective Order Scheduling via Reinforcement Learning”. In: *Algorithms 2023, 16, 495* (2023). DOI: https://doi.org/10.3390/a16110495.

[Ton+] Tong Zhou et al. “Multi-agent reinforcement learning for online scheduling in smart factories”. In: *Robotics and Computer-Integrated Manufacturing* ().

[Vla17] Nasteski Vladimir. “An overview of the supervised machine learning methods”. In: *HORIZONS 4, p.51-62* (2017). DOI: 10.20544/horizons.b.04.1.17.p05.

[Wan12] Jiancun Wang. “Chapter 15. Petri Nets”. In: *In: Jiacun Wang, Handbook of finite state based models and applications, CRC Press, 2012* (2012).

[Wat89] C.J.C.H. Watldns. “Learning from delayed rewards.” MA thesis. PhD Thesis, University of Cambridge, England., 1989.

[WS22] Johann-Dietrich Wörner and Christoph M. Schmidt. *Sicherheit, Resilienz und Nachhaltigkeit*. 2022.

[WT96] Wei Zhang and Thomas G. Dietterich. “A Reinforcement Learning Approach to Job-shop Scheduling”. In: (1996).

[Xia+24] Xiao Wang et al. “A novel method-based reinforcement learning with deep temporal difference network for flexible double shop scheduling problem”. In: *Scientific Reports* (2024). DOI: https://doi.org/10.1038/s41598-024-59414-8.

[XJR12] Xueping Li, Jiao Wang, and Rapinder Sawhney. “Reinforcement learning for joint pricing, lead-time and scheduling decisions in make-to-order systems”. In: *European Journal of Operational Research* (2012). DOI: http://dx.doi.org/10.1016/j.ejor.2012.03.020.

[Yon+24] Yong Lei et al. “Deep reinforcement learning for dynamic distributed job shop scheduling problem with transfers”. In: *Expert Systems with Applications* (2024). DOI: https://doi.org/10.1016/j.eswa.2024.123970.

[You+23] Youshan Liu et al. “Integration of deep reinforcement learning and multi-agent system for dynamic scheduling for re-entrant hybrid flow shop considering worker fatigue and skill levels”. In: *Robotics and Computer-Integrated Manufacturing 84, p. 102605* (2023). DOI: 10.1016/j.rcim.2023.102605.

[Zhi+23] Zhiwei Liu et al. “Low-latency Virtual Network function Scheduling Algorithm Based on Deep Reinforcement Learning”. In: *Computer Networks* (2023). DOI: https://doi.org/10.1016/j.comnet.2024.110418.

[Zop00] Christopher Zoppou. “Review of urban storm water models”. In: *Environmental Modelling Software* (2000). DOI: https://doi.org/10.1016/S1364-8152(00)00084-0.

```
clc;clear; close all;
% Title: Development of a resilient Reinforcement Learning-based decision
% algorithm for order scheduling
%
% Author: Fabio Serra Pereira
%
% Description: Traditional Method: Here it will simulate the Petri Nets
% Model. The idea is to simulate process per process, no inteligence will
% be processed.
%
% reading input data
%
TimesProcess = readtable('inputData_Times.csv');
TimesProcessAux = readtable('inputData_Times.csv');
Storages = readtable('inputData-Storages.csv');
StoragesAux = readtable('inputData-Storages.csv');
StorageStatus = readtable('inputData-StorageStatus.csv');
Supplier = readtable('inputData-Supplier.csv');
%
% declaring variables
%
finalProductStages =  {"Printer", "Montage","Electrical_Function_Verification",↙
"Packaging"};
afterMainStorage = {"Commissioning", "Powder coating", "Oven 1"};
fromMontage = {"Montage", "Electrical Function Verification", "Packaging"};
noMainStorage = {"Milling 1", "Sawing", "Cleaning & Drying", "Verification"};
missingLaser = {"Laser", "Cleaning & Drying", "Verification"};
labels = {"_complete_", "_notLasered_"};
startWorkingDay = datetime(2024,07,15,8,0,0);
%closeWorkingDay = datetime(2024,07,15,17,0,0);
stopWorkingDay = datetime(2024,07,15,8,0,0);
VariableNames = ["Start_Time", "Stop_time", "Machine_Storage", "Production_Time",↙
"Transport_Time"];
rowName = ["Milling 1", "Milling 2", "Sawing", "Cleaning & Drying", "Verification",↙
"Laser", "Commissioning", "Powder coating", "Oven 1", "Oven 2", "Printer", "Montage"↙
,"Electrical_Function_Verification", "Packaging"];
pr = ["ProductA", "ProductB", "ProductC", "ProductD", "ProductE", "ProductF",↙
"final_prod", "base", "electrical_func"];
insertPlusDay = 0;
cont = 1;
file_nr = 1;
%
% Simulating
%
productsOutput = {};
MachineStatus = {};
aux_MachineStatus = MachineStatus;
sumStartWorkingTime = false;
value2Sum = 0;
```

```
% start with the main storage scenario
d1 =  datetime("now");
for k = 1:height(StorageStatus)
    Orders = readtable('inputData-Orders.csv');
    for i = 1:height(Orders)
        d2 = datetime("now");
        % check  Product
        productsOutput = {};
        MachineStatus = {};
        usageOfOven = true;
        for j = 1:length(pr(1,1:6))
            % variable of Labels
            %if 25 <= Orders{i, "Orders"} <= 30
            %    b = Storages.TypeOfProduct == "Printer";
            %    Storages{b, "MaxStorage"}  = 10;
            %else
            %    b = Storages.TypeOfProduct == "Printer";
            %    Storages{b, "MaxStorage"} = StoragesAux{b, "MaxStorage"};
            %end
            if 10 <= Orders{i, "Orders"} <= 15
                a = TimesProcess.Product == pr(1,j) & TimesProcess.Station == ↙
"Cleaning & Drying";
                TimesProcess{a, "P_T_Face1_min_"} = TimesProcessAux{a,↙
"P_T_Face1_min_"} * 3;
            else
                a = TimesProcess.Product == pr(1,j) & TimesProcess.Station == ↙
"Cleaning & Drying";
                TimesProcess{a, "P_T_Face1_min_"} = TimesProcessAux{a,↙
"P_T_Face1_min_"};
            end
            %if 43 <= Orders{1, "Orders"} <= 47
            %    a = TimesProcess.Product == pr(1,j) & TimesProcess.Station == ↙
"Milling 1";
            %    TimesProcess{a, "R_T__min_"} = TimesProcessAux{a, "R_T__min_"} * 3;
            %else
            %    a = TimesProcess.Product == pr(1,j) & TimesProcess.Station == ↙
"Milling 1";
            %    TimesProcess{a, "R_T__min_"} = TimesProcessAux{a, "R_T__min_"};
            %end
            %
            while Orders{i,[pr(1,j)]} > 0
                % create output
                MachineStatusTime = array2table(zeros(14,5), "RowNames", rowName,↙
"VariableNames", VariableNames);
                % add each table to a dict to separate each table per product
                productsOutput{end+1} = pr(1,j);
                if isempty(aux_MachineStatus)
                    % setting values in the row
                    MachineStatusTime.Start_Time = zeros(length(rowName),1) + ↙
```

```
startWorkingDay;
                    %MachineStatusTime.Close_Hour = zeros(length(rowName),1) + ↙
closeWorkingDay;
                    MachineStatusTime.Stop_time = zeros(length(rowName),1) + ↙
stopWorkingDay;
                    MachineStatusTime.Machine_Storage = zeros(length(rowName),1) - 1;
                    % set the storage available for each product
                    MachineStatusTime("Sawing", "Machine_Storage") = array2table↙
(StorageStatus{k, "Sawing"});
                    MachineStatusTime("Laser", "Machine_Storage") = array2table↙
(StorageStatus{k, "Laser"});
                    MachineStatusTime("Printer", "Machine_Storage") = array2table↙
(StorageStatus{k, "Printer"});
                    MachineStatusTime("Electrical_Function_Verification",↙
"Machine_Storage") = array2table(StorageStatus{k,↙
"Electrical_Function_Verification"});
                    MachineStatusTime("Montage", "Machine_Storage") = array2table↙
(StorageStatus{k, "Montage"});
                else
                    % Getting the values from the first processed product
                    MachineStatusTime.Start_Time = aux_MachineStatus{end}.Stop_time;
                    %MachineStatusTime.Close_Hour = aux_MachineStatus{end}.↙
Close_Hour;
                    MachineStatusTime.Stop_time = aux_MachineStatus{end}.Stop_time;
                    MachineStatusTime.Machine_Storage = aux_MachineStatus{end}.↙
Machine_Storage;
                end
                %disp(MachineStatusTime)
                % add transport and production time to matrix
                for p = 1:length(rowName)
                    % get the production time and transportation time
                    if p <= 10
                        a = TimesProcess.Product == pr(1,j) & TimesProcess.Station == ↙
rowName(p);
                    else
                        a = TimesProcess.Station == rowName(p);
                    end
                    % actualize values
                    MachineStatusTime(rowName(p), "Production_Time") = array2table↙
(table2array(TimesProcess(a, "P_T_Face1_min_")) + table2array(TimesProcess(a,↙
"P_T_Face2_min_")));
                    MachineStatusTime(rowName(p), "Transport_Time") = TimesProcess(a,↙
"R_T__min_");
                end
                % if product must be complete
                if StorageStatus{k, pr(1,j)+labels(1,1)} < 1 && StorageStatus{k, pr↙
(1,j)+labels(1,2)} < 1
                    % run simulating
                    for q = 1:length(noMainStorage)
```

```
                        % adjust if use milling 1 or 2
                        if q == 1
                            if ismember(pr(1,j), ["ProductD", "ProductE",↙
"ProductF"])
                                noMainStorage{q} = "Milling 2";
                            else
                               noMainStorage{q} = "Milling 1";
                            end
                            MachineStatusTime{noMainStorage{q}, "Start_Time"} =↙
MachineStatusTime{"Sawing", "Stop_time"};
                        end
                        % define start time
                        if q ~= 1
                            if MachineStatusTime{noMainStorage{q}, "Start_Time"} -↙
MachineStatusTime{noMainStorage{q-1}, "Stop_time"} > hours(10)
                                MachineStatusTime{noMainStorage{q}, "Start_Time"} =↙
table2array(MachineStatusTime(noMainStorage{q}, "Stop_time")) + seconds(60);
                            else
                                MachineStatusTime{noMainStorage{q}, "Start_Time"} =↙
table2array(MachineStatusTime(noMainStorage{q-1}, "Stop_time")) + seconds(60);
                            end
                        end
                        % check if need to wait untill delivery
                        stopTime =  table2array(MachineStatusTime(noMainStorage{q}, ↙
"Start_Time")) + seconds(table2array(MachineStatusTime(noMainStorage{q}, "↙
Production_Time")))+ seconds(table2array(MachineStatusTime(noMainStorage{q}, "↙
Transport_Time")));
                       b =  TimesProcess.Station == noMainStorage{q} & TimesProcess.↙
Product == pr(1,j);
                        if MachineStatusTime{noMainStorage{q}, "Machine_Storage"} <↙
TimesProcess{b,"ConsumRawMaterial"} && MachineStatusTime{noMainStorage{q},↙
"Machine_Storage"} >= 0
                            % parms for datetime
                            tag = day(MachineStatusTime{noMainStorage{q},↙
"Start_Time"}) + day(3);
                            monat = month(MachineStatusTime{noMainStorage{q},↙
"Start_Time"});
                            jahr = year(MachineStatusTime{noMainStorage{q},↙
"Start_Time"});
                            % parms for the matrix
                            a = Supplier.Supplier == noMainStorage{q};
                            indice = find(MachineStatusTime.Row == noMainStorage{q});
                            % update the matrix
                            MachineStatusTime{noMainStorage{q}, "Start_Time"} =↙
datetime(jahr,monat,tag,8,0,0);
                            stopTime =  datetime(jahr,monat,tag,8,0,0) + seconds↙
(table2array(MachineStatusTime(noMainStorage{q},"Production_Time")))+ seconds↙
(table2array(MachineStatusTime(noMainStorage{q},"Transport_Time")));
                            % MachineStatusTime{indice:end, "Close_Hour"} = datetime↙
```

```
(jahr,monat,tag,17,0,0) + days(table2array(Supplier(a, "TimeToDelivery_days_")));
                        % fulfill the storage
                        b = Storages.TypeOfProduct == noMainStorage{q};
                        MachineStatusTime{noMainStorage{q}, "Machine_Storage"} =↙
Storages{b,"MaxStorage"};
                    end
                    % set stop time
                    MachineStatusTime{noMainStorage{q}, "Stop_time"} = stopTime;
                    b =  TimesProcess.Station == noMainStorage{q} & TimesProcess.↙
Product == pr(1,j);
                    MachineStatusTime{noMainStorage{q}, "Machine_Storage"} =↙
MachineStatusTime{noMainStorage{q}, "Machine_Storage"} - TimesProcess{b,"↙
ConsumRawMaterial"};
                    % make sure to work in available time
                    %if MachineStatusTime{noMainStorage{q}, "Start_Time"} >↙
MachineStatusTime{noMainStorage{q}, "Close_Hour"}
                    %    tag = day(MachineStatusTime{noMainStorage{q},↙
"Start_Time"});
                    %    monat = month(MachineStatusTime{noMainStorage{q},↙
"Start_Time"});
                    %    jahr = year(MachineStatusTime{noMainStorage{q},↙
"Start_Time"});
                    %    startWorkingDay = startWorkingDay + days(1);
                    %    MachineStatusTime{noMainStorage{q}, "  _Hour"} =↙
MachineStatusTime{noMainStorage{q}, "Close_Hour"} + days(1);
                    %    MachineStatusTime{noMainStorage{q}, "Start_Time"} =↙
datetime(jahr,monat,tag,8,0,0) + days(1);
                    %    MachineStatusTime{noMainStorage{q}, "Stop_time"} =↙
datetime(jahr,monat,tag,8,0,0) + days(1) + seconds(table2array(MachineStatusTime↙
(noMainStorage{q},"Production_Time")))+ seconds(table2array(MachineStatusTime↙
(noMainStorage{q},"Transport_Time")));
                    %end
                end
                % increase product in the storage
                StorageStatus{k, pr(1,j)+labels(1,2)} = StorageStatus{k, pr(1,j)↙
+labels(1,2)} + 1;
            end
            % if product must be just laser
            if StorageStatus{k, pr(1,j)+labels(1,1)} < 1 && StorageStatus{k, pr↙
(1,j)+labels(1,2)} >= 1
                % run simulating
                for q = 1:length(missingLaser)
                    % define start time
                    if q == 1
                        if MachineStatusTime{missingLaser{q}, "Start_Time"} <↙
MachineStatusTime{noMainStorage{end}, "Stop_time"}
                            MachineStatusTime{missingLaser{q}, "Start_Time"} =↙
MachineStatusTime{noMainStorage{end}, "Stop_time"} + seconds(60);
                        else
```

```
                                MachineStatusTime{missingLaser{q}, "Start_Time"} =↙
MachineStatusTime{missingLaser{q}, "Start_Time"} + seconds(60);
                            end
                        else
                            MachineStatusTime{missingLaser{q}, "Start_Time"} =↙
table2array(MachineStatusTime(missingLaser{q-1}, "Stop_time")) + seconds(60);
                        end
                        % check if need to wait untill delivery
                        stopTime =  table2array(MachineStatusTime(missingLaser{q}, ↙
"Start_Time")) + seconds(table2array(MachineStatusTime(missingLaser{q}, "↙
Production_Time")))+ seconds(table2array(MachineStatusTime(missingLaser{q}, "↙
Transport_Time")));
                        b =  TimesProcess.Station == missingLaser{q} & TimesProcess. ↙
Product == pr(1,j);
                        if MachineStatusTime{missingLaser{q}, "Machine_Storage"} <↙
TimesProcess{b,"ConsumRawMaterial"} && MachineStatusTime{missingLaser{q}, ↙
"Machine_Storage"} >= 0
                            % parms for datetime
                            tag = day(MachineStatusTime{missingLaser{q}, ↙
"Start_Time"}) + day(1);
                            monat = month(MachineStatusTime{missingLaser{q}, ↙
"Start_Time"});
                            jahr = year(MachineStatusTime{missingLaser{q}, ↙
"Start_Time"});
                            % parms for the matrix
                            a = Supplier.Supplier == missingLaser{q};
                            indice = find(MachineStatusTime.Row == missingLaser{q});
                            % updating the matrix
                            MachineStatusTime{missingLaser{q}, "Start_Time"} =↙
datetime(jahr,monat,tag,8,0,0);
                            stopTime =  datetime(jahr,monat,tag,8,0,0) + seconds ↙
(table2array(MachineStatusTime(missingLaser{q},"Production_Time")))+ seconds↙
(table2array(MachineStatusTime(missingLaser{q},"Transport_Time")));
                            % MachineStatusTime{indice:end, "Close_Hour"} = datetime ↙
(jahr,monat,tag,17,0,0) + days(table2array(Supplier(a, "TimeToDelivery_days_")));
                            % fulfill the storage
                            b = Storages.TypeOfProduct == missingLaser{q};
                            MachineStatusTime{missingLaser{q}, "Machine_Storage"} =↙
Storages{b,"MaxStorage"};
                        end
                        % set stop time
                        MachineStatusTime{missingLaser{q}, "Stop_time"} = stopTime;
                        b =  TimesProcess.Station == missingLaser{q} & TimesProcess. ↙
Product == pr(1,j);
                        MachineStatusTime{missingLaser{q}, "Machine_Storage"} =↙
MachineStatusTime{missingLaser{q}, "Machine_Storage"} - TimesProcess{b,"↙
ConsumRawMaterial"};
                        % make sure to work in available time
                        %if MachineStatusTime{missingLaser{q}, "Start_Time"} >↙
```

```
MachineStatusTime{missingLaser{q}, "Close_Hour"}
                        %    tag = day(MachineStatusTime{missingLaser{q}, ↙
"Start_Time"});
                        %    monat = month(MachineStatusTime{missingLaser{q}, ↙
"Start_Time"});
                        %    jahr = year(MachineStatusTime{missingLaser{q}, ↙
"Start_Time"});
                        %    startWorkingDay = startWorkingDay + days(1);
                        %    MachineStatusTime{missingLaser{q}, "Close_Hour"} = ↙
MachineStatusTime{missingLaser{q}, "Close_Hour"} + days(1);
                        %    MachineStatusTime{missingLaser{q}, "Start_Time"} = ↙
datetime(jahr,monat,tag,8,0,0) + days(1);
                        %    MachineStatusTime{missingLaser{q}, "Stop_time"} = ↙
datetime(jahr,monat,tag,8,0,0) + days(1) + seconds(table2array(MachineStatusTime ↙
(missingLaser{q},"Production_Time")))+ seconds(table2array(MachineStatusTime ↙
(missingLaser{q},"Transport_Time")));
                        %end
                    end
                    % increase product in the storage
                    StorageStatus{k, pr(1,j)+labels(1,1)} = StorageStatus{k, pr(1,j) ↙
+labels(1,1)} + 1;
                    StorageStatus{k, pr(1,j)+labels(1,2)} = StorageStatus{k, pr(1,j) ↙
+labels(1,2)} - 1;
                end

                % if product available is in Main Storage
                if StorageStatus{k, pr(1,j)+labels(1,1)} >= 1
                    % run simulating
                    for q = 1:length(afterMainStorage)
                        % paralelize the oven process
                        if usageOfOven == true && (afterMainStorage{q} == "Oven 2" ||↙
afterMainStorage{q} == "Oven 1")
                            afterMainStorage{q} = "Oven 1";
                            usageOfOven = false;
                        elseif usageOfOven == false && (afterMainStorage{q} == "Oven↙
2" || afterMainStorage{q} == "Oven 1")
                            afterMainStorage{q} = "Oven 2";
                            usageOfOven = true;
                        end
                        % define start time
                        if q == 1
                            if MachineStatusTime{afterMainStorage{q}, "Start_Time"} <↙
MachineStatusTime{missingLaser{end}, "Stop_time"}
                                MachineStatusTime{afterMainStorage{q}, "Start_Time"}↙
= MachineStatusTime{missingLaser{end}, "Stop_time"} + seconds(60);
                            else
                                MachineStatusTime{afterMainStorage{q}, "Start_Time"}↙
= MachineStatusTime{afterMainStorage{q}, "Start_Time"} + seconds(60);
                            end
```

```
                        else
                            MachineStatusTime{afterMainStorage{q}, "Start_Time"} =↙
table2array(MachineStatusTime(afterMainStorage{q-1}, "Stop_time")) + seconds(60);
                        end
                        % check if need to wait untill delivery
                        stopTime =  table2array(MachineStatusTime(afterMainStorage↙
{q}, "Start_Time")) + seconds(table2array(MachineStatusTime(afterMainStorage{q}, "↙
Production_Time")))+ seconds(table2array(MachineStatusTime(afterMainStorage{q}, "↙
Transport_Time")));
                        b =  TimesProcess.Station == afterMainStorage{q} &↙
TimesProcess.Product == pr(1,j);
                        if MachineStatusTime{afterMainStorage{q}, "Machine_Storage"}↙
< TimesProcess{b,"ConsumRawMaterial"} && MachineStatusTime{afterMainStorage{q},↙
"Machine_Storage"} >= 0
                            % parms for datetime
                            tag = day(MachineStatusTime{afterMainStorage{q},↙
"Start_Time"});
                            monat = month(MachineStatusTime{afterMainStorage{q},↙
"Start_Time"});
                            jahr = year(MachineStatusTime{afterMainStorage{q},↙
"Start_Time"});
                            % parms for the matrix
                            a = Supplier.Supplier == afterMainStorage{q};
                            indice = find(MachineStatusTime.Row == afterMainStorage↙
{q});
                            % updating the matrix
                            MachineStatusTime{afterMainStorage{q}, "Start_Time"} =↙
datetime(jahr,monat,tag,8,0,0);
                            stopTime =  datetime(jahr,monat,tag,8,0,0) + seconds↙
(table2array(MachineStatusTime(afterMainStorage{q},"Production_Time")))+ seconds↙
(table2array(MachineStatusTime(afterMainStorage{q},"Transport_Time")));
                            % MachineStatusTime{indice:end, "Close_Hour"} = datetime↙
(jahr,monat,tag,17,0,0) + days(table2array(Supplier(a, "TimeToDelivery_days_")));
                            % fulfill the storage
                            b = Storages.TypeOfProduct == afterMainStorage{q};
                            MachineStatusTime{afterMainStorage{q}, "Machine_Storage"}↙
= Storages{b,"MaxStorage"};
                        end
                        % set stop time
                        MachineStatusTime{afterMainStorage{q}, "Stop_time"} =↙
stopTime;
                        b =  TimesProcess.Station == afterMainStorage{q} &↙
TimesProcess.Product == pr(1,j);
                        MachineStatusTime{afterMainStorage{q}, "Machine_Storage"} =↙
MachineStatusTime{afterMainStorage{q}, "Machine_Storage"} - TimesProcess{b,"↙
ConsumRawMaterial"};
                        % make sure to work in available time
                        %if MachineStatusTime{afterMainStorage{q}, "Start_Time"} >↙
MachineStatusTime{afterMainStorage{q}, "Close_Hour"}
```

```
                        %    tag = day(MachineStatusTime{afterMainStorage{q}, ↙
"Start_Time"});
                        %    monat = month(MachineStatusTime{afterMainStorage{q}, ↙
"Start_Time"});
                        %    jahr = year(MachineStatusTime{afterMainStorage{q}, ↙
"Start_Time"});
                        %    startWorkingDay = startWorkingDay + days(1);
                        %    MachineStatusTime{afterMainStorage{q}, "Close_Hour"} =↙
MachineStatusTime{afterMainStorage{q}, "Close_Hour"} + days(1);
                        %    MachineStatusTime{afterMainStorage{q}, "Start_Time"} =↙
datetime(jahr,monat,tag,8,0,0) + days(1);
                        %    MachineStatusTime{afterMainStorage{q}, "Stop_time"} =↙
datetime(jahr,monat,tag,8,0,0)  + days(1) + seconds(table2array(MachineStatusTime ↙
(afterMainStorage{q},"Production_Time")))+ seconds(table2array(MachineStatusTime ↙
(afterMainStorage{q},"Transport_Time")));
                        %end
                    end
                    % increase product in the storage
                    StorageStatus{k, pr(1,j)+labels(1,1)} = StorageStatus{k, pr(1,j) ↙
+labels(1,1)} - 1;
                end
                % make sure to not repeat over the same order
                Orders{i,[pr(1,j)]} = Orders{i,[pr(1,j)]} - 1;
                MachineStatus{end+1} = MachineStatusTime;
                %disp(MachineStatusTime)
                aux_MachineStatus = MachineStatus;
            end
        end
        % actualize  the last  machines
        % here we will consider the final product, as the time which is
        % equal for all products, therefore it is the final_product
        % define start time
        for q = 1:length(finalProductStages)
            for w = 1:length(MachineStatus)
                if MachineStatus{w}.Machine_Storage("Montage") >= 1 &&↙
finalProductStages{q} == "Printer"
                    % just pass
                else
                    if q ~= 1
                        if MachineStatus{w}.Machine_Storage("Montage") >= 1 &&↙
finalProductStages{q} == "Montage"
                            MachineStatus{w}.Start_Time(finalProductStages{q}) =↙
MachineStatus{end}.Stop_time("Oven 2") + seconds(60);
                        else
                            MachineStatus{w}.Start_Time(finalProductStages{q}) =↙
MachineStatus{w}.Stop_time(finalProductStages{q-1}) + seconds(60);
                        end
                    else
                        MachineStatus{w}.Start_Time(finalProductStages{q}) =↙
```

```
MachineStatus{end}.Stop_time("Oven 2") + seconds(60);
                    end
                    % check if need to wait untill delivery
                    stopTime =  MachineStatus{w}.Start_Time(finalProductStages{q}) + seconds(MachineStatus{w}.Production_Time(finalProductStages{q}))+ seconds(MachineStatus{w}.Transport_Time(finalProductStages{q}));
                    b =  TimesProcess.Station == finalProductStages{q};
                    if MachineStatus{w}.Machine_Storage(finalProductStages{q}) < TimesProcess{b,"ConsumRawMaterial"} && MachineStatus{w}.Machine_Storage(finalProductStages{q}) >= 0
                        % parms for datetime
                        tag = day(MachineStatus{w}.Start_Time(finalProductStages{q})) + day(2);
                        monat = month(MachineStatus{w}.Start_Time(finalProductStages{q}));
                        jahr = year(MachineStatus{w}.Start_Time(finalProductStages{q}));
                        % parms for matrix
                        a = Supplier.Supplier == finalProductStages{q};
                        indice = find(MachineStatusTime.Row == finalProductStages{q});
                        % ajust matrix
                        MachineStatus{w}.Start_Time(finalProductStages{q}) = datetime(jahr,monat,tag,8,0,0);
                        % MachineStatus{w}.Close_Hour(indice:end) = datetime(jahr,monat,tag,17,0,0) + days(table2array(Supplier(a, "TimeToDelivery_days_")));
                        stopTime =  datetime(jahr,monat,tag,8,0,0) + seconds(MachineStatus{w}.Production_Time(finalProductStages{q})) + seconds(MachineStatus{w}.Transport_Time(finalProductStages{q}));
                        % fulfill the storage
                        b = Storages.TypeOfProduct == finalProductStages{q};
                        MachineStatus{w}.Machine_Storage(finalProductStages{q}) = Storages{b,"MaxStorage"};
                    end
                    % set stop time
                    MachineStatus{w}.Stop_time(finalProductStages{q}) = stopTime;
                    b =  TimesProcess.Station == finalProductStages{q};
                    MachineStatus{w}.Machine_Storage(finalProductStages{q}) = MachineStatus{w}.Machine_Storage(finalProductStages{q}) - TimesProcess{b,"ConsumRawMaterial"};
                    if finalProductStages{q} == "Printer"
                        MachineStatus{w}.Machine_Storage("Montage") = MachineStatus{w}.Machine_Storage("Montage") + 1;
                    end
                    if finalProductStages{q} == "Montage"
                        MachineStatus{w}.Machine_Storage("Montage") = MachineStatus{w}.Machine_Storage("Montage") - 1;
                    end
                    % make sure to work in available time
```

```
                    %if MachineStatus{w}.Start_Time(finalProductStages{q}) >↙
MachineStatus{w}.Close_Hour(finalProductStages{q})
                    %    tag = day(MachineStatus{w}.Start_Time(finalProductStages↙
{q}));
                    %    monat = month(MachineStatus{w}.Start_Time(finalProductStages↙
{q}));
                    %    jahr = year(MachineStatus{w}.Start_Time(finalProductStages↙
{q}));
                    %    indice = find(MachineStatusTime.Row == finalProductStages↙
{q});
                    %    MachineStatus{w}.Close_Hour(finalProductStages{q}) =↙
datetime(jahr,monat,tag,17,0,0) + days(1);
                    %    MachineStatus{w}.Start_Time(finalProductStages{q}) =↙
datetime(jahr,monat,tag,8,0,0) + days(1);
                    %    MachineStatus{w}.Stop_time(finalProductStages{q}) = datetime↙
(jahr,monat,tag,8,0,0) + days(1) + seconds(MachineStatus{w}.Production_Time↙
(finalProductStages{q})) + seconds(MachineStatus{w}.Transport_Time(finalProductStages↙
{q}));
                    %    if q == length(finalProductStages) && w == length↙
(MachineStatus)
                    %        %startWorkingDay = startWorkingDay + days(1);
                    %    end
                    %end
                end
            end
        end
        for w = 1:length(MachineStatus)
            disp(productsOutput{w})
            disp(MachineStatus{w})
        end
        storage_status_nr = StorageStatus{k, "Cases"};
        Order_nr = Orders{i,"Orders"};
        save("traditional/TM_"+file_nr, "MachineStatus", "storage_status_nr",↙
"Order_nr");
        file_nr = file_nr + 1;
        aux_MachineStatus = MachineStatus;
        d3 = datetime("now");
        totalTime = between(d1,d3);
        timePerOrder = between(d2,d3);
        save("time_trad/Time_"+cont, "totalTime","timePerOrder" );
        cont = cont + 1;
    end
end
```

```
clc;clear; close all; warning('off');
% Title: Development of a resilient Reinforcement Learning-based decision
% algorithm for order scheduling
%
% Author: Fabio Serra Pereira
%
% Description: Model with application of AI to simulate the production process
%
% reading input data
%
Storages = readtable('inputData-Storages.csv');
StorageStatus = readtable('inputData-StorageStatus.csv');
Supplier = readtable('inputData-Supplier.csv');
%
% declaring variables
%
pr = ["ProductA", "ProductB", "ProductC", "ProductD", "ProductE", "ProductF"];
%
% Simulating
%
productsOutput = {};
MachineStatus = {};
output = {};
sumStartWorkingTime = false;
value2Sum = 0;
cont = 1;
file_nr = 1;

% start with the main storage scenario
for k = 1:height(StorageStatus)
    Orders = readtable('inputData-Orders.csv');
    aux = Orders;
    d1 =  datetime("now");
    % then we need to get one order and see each product where to start
    for i = 1:4:height(Orders)
        % organizing the order list using the AI solution
        final = i + 3;
        if final <= height(Orders)
            prior = prioritization(StorageStatus(k, :), Orders(i:final, :));
            rowIndices = zeros(1, height(prior));
            for k_2 = 1:1:length(rowIndices)
                [maxValues, indices] = max(prior);
                rowIndices(k_2) = indices(k_2);
                prior(indices(k_2), :) = -100*ones(1, width(prior));
            end
            for j  = 1:length(rowIndices)
                aux(i+rowIndices(j)-1,:) = Orders(i+j-1,:);
            end
        end
```

```
    end
    d2 = datetime("now");
    for i = 1:height(aux)
        d3 = datetime("now");
        product_order = zeros(1, 6);
        for j = 1:length(pr)
            if aux{i, pr(j)} > 0
                qtable = decideStage(StorageStatus(k, :), pr(j), aux{i, pr(j)}, aux{i, "Orders"});
                result = qtable{1};
                [maxValues, indices] = max(result(1,:));
                product_order(j) = indices;
                % cases to manufacture:
                % case 1: indices == 0 or 1: not manufacture
                % case 2: indices == 2: manufacture from beginning
                % case 3: indices == 3: go to laser to be stamped
                % case 4: indices == 4: go directly to assembly phase
            end
        end
        d4 = datetime("now");
        [output, ss] = simulationWithAI(aux(i,:), product_order, StorageStatus(k,:), output, file_nr);
        StorageStatus(k,:) = ss
        d5 = datetime("now");
        file_nr = file_nr + 1;
        % saving datetime difference
        firstAI = between(d1,d2);
        secondAI = between(d3,d4);
        simulationTime = between(d4,d5);
        save("time/Time_"+cont, "simulationTime", "secondAI", "firstAI");
        cont = cont + 1;
    end

    % Then we need to
end
```

```
% Title: Development of a resilient Reinforcement Learning-based decision
% algorithm for order scheduling
%
% Author: Fabio Serra Pereira
%
% Description: Here we create the function to simulate the first part of
% the AI. We gave the order quantity of each product to manufacture and
% receive as output which section it must be start.
%
function result = prioritization(StorageStatus, Orders)
result = zeros(height(Orders));
% generate the production function
    function p = prodfunc(StorageStatus, Orders, order_nr)
        p = 0;
        if Orders{order_nr,"ProductA"} > 0
            p = p + (productionfunction_2("ProductA",StorageStatus(1,:),Orders
{order_nr,"ProductA"}, Orders{order_nr, "Orders"}));
        end
        if Orders{order_nr,"ProductB"} > 0
            p = p + (productionfunction_2("ProductB",StorageStatus(1,:),Orders
{order_nr,"ProductB"}, Orders{order_nr, "Orders"}));
        end
        if Orders{order_nr,"ProductC"} > 0
            p = p + (productionfunction_2("ProductC",StorageStatus(1,:),Orders
{order_nr,"ProductC"}, Orders{order_nr, "Orders"}));
        end
        if Orders{order_nr,"ProductD"} > 0
            p = p + (productionfunction_2("ProductD",StorageStatus(1,:),Orders
{order_nr,"ProductD"}, Orders{order_nr, "Orders"}));
        end
        if Orders{order_nr,"ProductE"} > 0
            p = p + (productionfunction_2("ProductE",StorageStatus(1,:),Orders
{order_nr,"ProductE"}, Orders{order_nr, "Orders"}));
        end
        if Orders{order_nr,"ProductF"} > 0
            p = p + (productionfunction_2("ProductF",StorageStatus(1,:),Orders
{order_nr,"ProductF"}, Orders{order_nr, "Orders"}));
        end
    end

    for n = 1:1:height(Orders)
        % create the markov chain process
        MDP = createMDP(5, ["first"; "second";"third";"fourth"]);
        % create rewards
        % state 1
        MDP.T(1,2,1) = 1
        MDP.R(1,2,1) = 1/(prodfunc(StorageStatus, Orders, n));
        MDP.T(1,3,2) = 1;
        MDP.R(1,3,2) = 1/(prodfunc(StorageStatus, Orders, n));
```

```
MDP.T(1,4,3) = 1;
MDP.R(1,4,3) = 1/(prodfunc(StorageStatus, Orders, n));
MDP.T(1,5,4) = 1;
MDP.R(1,5,4) = 1/(prodfunc(StorageStatus, Orders, n));
% state 2
MDP.T(2,2,1) = 1;
MDP.R(2,2,1) = 0;
MDP.T(2,2,2) = 1;
MDP.R(2,2,2) = 0;
MDP.T(2,2,3) = 1;
MDP.R(2,2,3) = 0;
MDP.T(2,2,4) = 1;
MDP.R(2,2,4) = 0;
% state 3
MDP.T(3,3,1) = 1;
MDP.R(3,3,1) = 0;
MDP.T(3,3,2) = 1;
MDP.R(3,3,2) = 0;
MDP.T(3,3,3) = 1;
MDP.R(3,3,3) = 0;
MDP.T(3,3,4) = 1;
MDP.R(3,3,4) = 0;
% state 4
MDP.T(4,4,1) = 1;
MDP.R(4,4,1) = 0;
MDP.T(4,4,2) = 1;
MDP.R(4,4,2) = 0;
MDP.T(4,4,3) = 1;
MDP.R(4,4,3) = 0;
MDP.T(4,4,4) = 1;
MDP.R(4,4,4) = 0;

% terminal states
MDP.TerminalStates = ["s2";"s3";"s4";"s5"];
% create environment
env = rlMDPEnv(MDP);
% specify a reset function that returns the initial agent state
env.ResetFcn = @() 1;
% fix the random generator seed for repoducibility
rng(0)
% create Q-Learning Agent
obsInfo = getObservationInfo(env);
actInfo = getActionInfo(env);
qTable = rlTable(obsInfo, actInfo);
qFunction = rlQValueFunction(qTable, obsInfo, actInfo);
qOptions = rlOptimizerOptions(LearnRate=1);
% create Q-Learning agent using this table representation
agentOpts = rlQAgentOptions;
agentOpts.DiscountFactor = 1;
```

```
        agentOpts.EpsilonGreedyExploration.Epsilon = 0.9;
        agentOpts.EpsilonGreedyExploration.EpsilonDecay = 0.01;
        agentOpts.CriticOptimizerOptions = qOptions;
        qAgent = rlQAgent(qFunction,agentOpts); %#ok<NASGU>
        % Train Q-Learning Agent
        trainOpts = rlTrainingOptions;
        trainOpts.MaxStepsPerEpisode = 10;
        trainOpts.MaxEpisodes = 500;
        trainOpts.StopTrainingCriteria = "AverageReward";
        trainOpts.StopTrainingValue = 13;
        trainOpts.ScoreAveragingWindowLength = 30;
        trainOpts.Plots = "none";
        doTraining = true;
        if doTraining
            % Train the agent.
            trainingStats = train(qAgent,env,trainOpts); %#ok<UNRCH>
        else
            % Load pretrained agent for the example.
            load("genericMDPQAgent.mat","qAgent");
        end
        Data = sim(qAgent,env);
        cumulativeReward = sum(Data.Reward);
        QTable = getLearnableParameters(getCritic(qAgent));
        result(n,:) = QTable{1}(1,:);
    end
end
```

```
% Title: Development of a resilient Reinforcement Learning-based decision
% algorithm for order scheduling
%
% Author: Fabio Serra Pereira
%
% Description: Here we create a function which reaturn the production
% function given a product
%
function f = productionfunction_2(pr, Storagestatus, qt, order_nr)
    % declaring variables
    stage1 = {"Commissioning", "Powder coating", "Oven 1", "Oven 2"};
    stage2 = {"Laser", "Cleaning & Drying", "Verification"};
    stage3 = {"Milling 1", "Milling 2", "Sawing", "Cleaning & Drying",↙
"Verification"};
    products = {"ProductA";"ProductB";"ProductC";"ProductD";"ProductE";"ProductF"};
    E_sum = 0; Pr_sum = 0; Tr_sum = 0;
    % reading input data required
    TimesProcess = readtable('inputData_Times.csv');
    Storages = readtable('inputData-Storages.csv');
    % resilience Scenario adjustment
    %if 25 <= order_nr <= 30
    %    b = Storages.TypeOfProduct == "Printer";
    %    Storages{b, "MaxStorage"}  = 10;
    %end
    %if 10 <= order_nr <= 15
    %    a = TimesProcess.Product == pr & TimesProcess.Station == "Cleaning & ↙
Drying";
    %    TimesProcess{a, "P_T_Face1_min_"} = TimesProcess{a, "P_T_Face1_min_"} * 3;
    %end
    if 43 <= order_nr <= 47
        a = TimesProcess.Product == pr & TimesProcess.Station ==  "Milling 1";
        TimesProcess{a, "R_T__min_"} = TimesProcess{a, "R_T__min_"} * 3;
    end
    % define the sum value of production time and transport time
    for j = 1:length(products)
        for i = 1:length(stage1)
            a = TimesProcess.Product == products{j} & TimesProcess.Station == stage1 ↙
{i};
            Pr_sum = Pr_sum + TimesProcess{a, "P_T_Face1_min_"} + TimesProcess{a,↙
"P_T_Face2_min_"};
            Tr_sum = Tr_sum + TimesProcess{a, "R_T__min_"};
        end
        for i = 1:length(stage2)
            a = TimesProcess.Product == products{j} & TimesProcess.Station == stage2 ↙
{i};
            Pr_sum = Pr_sum + TimesProcess{a, "P_T_Face1_min_"} + TimesProcess{a,↙
"P_T_Face2_min_"};
            Tr_sum = Tr_sum + TimesProcess{a, "R_T__min_"};
        end
```

```
        for i = 1:length(stage3)
            a = TimesProcess.Product == products{j} & TimesProcess.Station == stage3{i};
            Pr_sum = Pr_sum + TimesProcess{a, "P_T_Face1_min_"} + TimesProcess{a, "P_T_Face2_min_"};
            Tr_sum = Tr_sum + TimesProcess{a, "R_T__min_"};
        end
    end
    % define the sum value for the storage
    b = Storages.TypeOfProduct == pr+"_complete_";
    E_sum = E_sum + Storages{b, "MaxStorage"};
    b = Storages.TypeOfProduct == pr+"_notLasered_";
    E_sum = E_sum + Storages{b, "MaxStorage"};
    % now calculate the stage status
    % stage = 1: Only the main storage stage
    % stage = 2: Laser and main storage stages
    % stage = 3: All the stages
    Pr_status = 0; Tr_status = 0;
    for i =  1:length(stage1)
        a = TimesProcess.Product == pr & TimesProcess.Station == stage1{i};
        Pr_status = Pr_status + TimesProcess{a, "P_T_Face1_min_"} + TimesProcess{a, "P_T_Face2_min_"};
        Tr_status = Tr_status + TimesProcess{a, "R_T__min_"};
    end
    E_status = Storagestatus{1, pr+"_complete_"} - qt;
    E_status = E_status + Storagestatus{1, pr+"_notLasered_"};
    for i =  1:length(stage2)
        a = TimesProcess.Product == pr & TimesProcess.Station == stage2{i};
        Pr_status = Pr_status + TimesProcess{a, "P_T_Face1_min_"} + TimesProcess{a, "P_T_Face2_min_"};
        Tr_status = Tr_status + TimesProcess{a, "R_T__min_"};
    end
    for i =  1:length(stage3)
        a = TimesProcess.Product == pr & TimesProcess.Station == stage3{i};
        Pr_status = Pr_status + TimesProcess{a, "P_T_Face1_min_"} + TimesProcess{a, "P_T_Face2_min_"};
        Tr_status = Tr_status + TimesProcess{a, "R_T__min_"};
    end

    % adjust according the quantity of product to use
    if qt == 0
        factor = 2;
    else
        factor = 1/qt;
    end

    % return
    f = (Pr_status/Pr_sum) + (Tr_status/Tr_sum) + (E_status/E_sum);
end
```

```
% Title: Development of a resilient Reinforcement Learning-based decision
% algorithm for order scheduling
%
% Author: Fabio Serra Pereira
%
% Description: Here we create the function to simulate the first part of
% the AI. We gave the order quantity of each product to manufacture and
% receive as output which section it must be start.
%
function qtable = decideStage(StorageStatus, product, qt, order_nr)
    % create the markov chain process
    MDP = createMDP(4, ["empty";"manufacture"; "stample"; "available"]);
    % create rewards
    % state 1
    MDP.T(1,1,1) = 1;
    MDP.R(1,1,1) = -1;
    MDP.T(1,2,2) = 1;
    MDP.R(1,2,2) = productionfunction(product,StorageStatus,3,qt, order_nr);
    MDP.T(1,3,3) = 1;
    MDP.R(1,3,3) = productionfunction(product,StorageStatus,2,qt, order_nr);
    MDP.T(1,4,4) = 1;
    MDP.R(1,4,4) = productionfunction(product,StorageStatus,1,qt, order_nr);
    % state 2
    MDP.T(2,1,1) = 1;
    MDP.R(2,1,1) = 0;
    MDP.T(2,2,2) = 1;
    MDP.R(2,2,2) = 0;
    MDP.T(2,3,3) = 1;
    MDP.R(2,3,3) = 0;
    MDP.T(2,4,4) = 1;
    MDP.R(2,4,4) = 0;
    % state 3
    MDP.T(3,2,1) = 1;
    MDP.R(3,2,1) = (productionfunction(product,StorageStatus,3,qt, order_nr) - ↙
productionfunction(product,StorageStatus,2,qt, order_nr));
    MDP.T(3,1,2) = 1;
    MDP.R(3,1,2) = 0;
    MDP.T(3,3,3) = 1;
    MDP.R(3,3,3) = 0;
    MDP.T(3,4,4) = 1;
    MDP.R(3,4,4) = 0;
    % state 4
    MDP.T(3,2,1) = 1;
    MDP.R(3,2,1) = (productionfunction(product,StorageStatus,2,qt, order_nr) - ↙
productionfunction(product,StorageStatus(1,:),1,qt, order_nr));
    MDP.T(3,1,2) = 1;
    MDP.R(3,1,2) = 0;
    MDP.T(3,3,3) = 1;
    MDP.R(3,3,3) = 0;
```

```
    MDP.T(3,4,4) = 1;
    MDP.R(3,4,4) = 0;

    % terminal states
    MDP.TerminalStates = ["s2"; "s3"; "s4"];
    % create environment
    env = rlMDPEnv(MDP);
    % specify a reset function that returns the initial agent state
    env.ResetFcn = @() 1;
    % fix the random generator seed for repoducibility
    rng(0)
    % create Q-Learning Agent
    obsInfo = getObservationInfo(env);
    actInfo = getActionInfo(env);
    qTable = rlTable(obsInfo, actInfo);
    qFunction = rlQValueFunction(qTable, obsInfo, actInfo);
    qOptions = rlOptimizerOptions(LearnRate=1);
    % create Q-Learning agent using this table representation
    agentOpts = rlQAgentOptions;
    agentOpts.DiscountFactor = 1;
    agentOpts.EpsilonGreedyExploration.Epsilon = 0.9;
    agentOpts.EpsilonGreedyExploration.EpsilonDecay = 0.01;
    agentOpts.CriticOptimizerOptions = qOptions;


    qAgent = rlQAgent(qFunction,agentOpts); %#ok<NASGU>
    % Train Q-Learning Agent
    trainOpts = rlTrainingOptions;
    trainOpts.MaxStepsPerEpisode = 10;
    trainOpts.MaxEpisodes = 1000;
    trainOpts.StopTrainingCriteria = "AverageReward";
    trainOpts.StopTrainingValue = 13;
    trainOpts.ScoreAveragingWindowLength = 30;
    trainOpts.Plots = "none";
    doTraining = true;
    if doTraining
        % Train the agent.
        trainingStats = train(qAgent,env,trainOpts); %#ok<UNRCH>
    else
        % Load pretrained agent for the example.
        load("genericMDPQAgent.mat","qAgent");
    end
    Data = sim(qAgent,env);
    cumulativeReward = sum(Data.Reward);
    qtable = getLearnableParameters(getCritic(qAgent));
end
```

```
% Title: Development of a resilient Reinforcement Learning-based decision
% algorithm for order scheduling
%
% Author: Fabio Serra Pereira
%
% Description: Here we create a function which reaturn the production
% function given a product
%
function f = productionfunction(pr, Storagestatus, stage, qt, order_nr)
    % declaring variables
    stage1 = {"Commissioning", "Powder coating", "Oven 1", "Oven 2"};
    stage2 = {"Laser", "Cleaning & Drying", "Verification"};
    stage3 = {"Milling 1", "Milling 2", "Sawing", "Cleaning & Drying",↙
"Verification"};
    E_sum = 0; Pr_sum = 0; Tr_sum = 0;
    % reading input data required
    TimesProcess = readtable('inputData_Times.csv');
    Storages = readtable('inputData-Storages.csv');
    % resilience Scenario adjustment
    %if 25 <= order_nr <= 30
    %    b = Storages.TypeOfProduct == "Printer";
    %    Storages{b, "MaxStorage"}  = 10;
    %end
    %if 10 <= order_nr <= 15
    %    a = TimesProcess.Product == pr & TimesProcess.Station == "Cleaning &↙
Drying";
    %    TimesProcess{a, "P_T_Face1_min_"} = TimesProcess{a, "P_T_Face1_min_"} * 3;
    %end
    if 43 <= order_nr <= 47
        a = TimesProcess.Product == pr & TimesProcess.Station ==  "Milling 1";
        TimesProcess{a, "R_T__min_"} = TimesProcess{a, "R_T__min_"} * 3;
    end
    % define the sum value of production time and transport time
    for i = 1:length(stage1)
        a = TimesProcess.Product == pr & TimesProcess.Station == stage1{i};
        Pr_sum = Pr_sum + TimesProcess{a, "P_T_Face1_min_"} + TimesProcess{a,↙
"P_T_Face2_min_"};
        Tr_sum = Tr_sum + TimesProcess{a, "R_T__min_"};
    end
    for i = 1:length(stage2)
        a = TimesProcess.Product == pr & TimesProcess.Station == stage2{i};
        Pr_sum = Pr_sum + TimesProcess{a, "P_T_Face1_min_"} + TimesProcess{a,↙
"P_T_Face2_min_"};
        Tr_sum = Tr_sum + TimesProcess{a, "R_T__min_"};
    end
    for i = 1:length(stage3)
        a = TimesProcess.Product == pr & TimesProcess.Station == stage3{i};
        Pr_sum = Pr_sum + TimesProcess{a, "P_T_Face1_min_"} + TimesProcess{a,↙
"P_T_Face2_min_"};
```

```
        Tr_sum = Tr_sum + TimesProcess{a, "R_T__min_"};
    end
    % define the sum value for the storage
    b = Storages.TypeOfProduct == pr+"_complete_";
    E_sum = E_sum + Storages{b, "MaxStorage"};
    b = Storages.TypeOfProduct == pr+"_notLasered_";
    E_sum = E_sum + Storages{b, "MaxStorage"};
    % now calculate the stage status
    % stage = 1: Only the main storage stage
    % stage = 2: Laser and main storage stages
    % stage = 3: All the stages
    Pr_status = 0; Tr_status = 0; E_status = 0;
    for i =  1:length(stage1)
        a = TimesProcess.Product == pr & TimesProcess.Station == stage1{i};
        Pr_status = Pr_status + TimesProcess{a, "P_T_Face1_min_"} + TimesProcess{a,↙
"P_T_Face2_min_"};
        Tr_status = Tr_status + TimesProcess{a, "R_T__min_"};
    end
    if Storagestatus{1, pr+"_complete_"} >= qt
        E_status = Storagestatus{1, pr+"_complete_"} - qt;
    end
    if stage == 3 || stage == 2
        for i =  1:length(stage2)
            a = TimesProcess.Product == pr & TimesProcess.Station == stage2{i};
            Pr_status = Pr_status + TimesProcess{a, "P_T_Face1_min_"} + TimesProcess↙
{a, "P_T_Face2_min_"};
            Tr_status = Tr_status + TimesProcess{a, "R_T__min_"};
        end
        if Storagestatus{1, pr+"_complete_"} < qt
            E_status = Storagestatus{1, pr+"_notLasered_"} - qt;
        else
            E_status = E_status + Storagestatus{1, pr+"_notLasered_"};
        end
    end
    if stage == 3
        for i =  1:length(stage3)
            a = TimesProcess.Product == pr & TimesProcess.Station == stage3{i};
            Pr_status = Pr_status + TimesProcess{a, "P_T_Face1_min_"} + TimesProcess↙
{a, "P_T_Face2_min_"};
            Tr_status = Tr_status + TimesProcess{a, "R_T__min_"};
        end
    end
    % return
    if E_status <= 0 || qt <=  0
        f = -10;
    else
        f = (Pr_status/Pr_sum) + (Tr_status/Tr_sum) + (E_status/E_sum);
    end
end
```

```
% Title: Development of a resilient Reinforcement Learning-based decision
% algorithm for order scheduling
%
% Author: Fabio Serra Pereira
%
% Description: Simulation process using AI applied previosly to organize
% the data as input to here.
%
function [output, ss] = simulationWithAI(Orders, stage, StorageStatus, output, ↙
file_nr)
    %
    % reading input data
    %
    TimesProcess = readtable('inputData_Times.csv');
    Storages = readtable('inputData-Storages.csv');
    Supplier = readtable('inputData-Supplier.csv');
    %
    % declaring variables
    %
    finalProductStages =  {"Printer", "Montage","Electrical_Function_Verification",↙
"Packaging"};
    afterMainStorage = {"Commissioning", "Powder coating", "Oven 1"};
    fromMontage = {"Montage", "Electrical Function Verification", "Packaging"};
    noMainStorage = {"Milling 1", "Sawing", "Cleaning & Drying", "Verification"};
    missingLaser = {"Laser", "Cleaning & Drying", "Verification"};
    needStorage = {"Laser", "Electrical_Function_Verification", "Printer", "Sawing"};
    labels = {"_complete_", "_notLasered_"};
    startWorkingDay = datetime(2024,07,15,8,0,0);
    closeWorkingDay = datetime(2024,07,15,17,0,0);
    stopWorkingDay = datetime(2024,07,15,8,0,0);
    VariableNames = ["Start_Time", "Stop_time", "Production_Time", "Transport_Time"];
    rowName = ["Milling 1", "Milling 2", "Sawing", "Cleaning & Drying",↙
"Verification", "Laser", "Commissioning", "Powder coating", "Oven 1", "Oven 2",↙
"Printer", "Montage" ,"Electrical_Function_Verification", "Packaging"];
    pr = ["ProductA", "ProductB", "ProductC", "ProductD", "ProductE", "ProductF",↙
"final_prod", "base", "electrical_func"];
    insertPlusDay = 0;
    %
    % Simulation
    %
    productsOutput = {};
    MachineStatus = {};
    sumStartWorkingTime = false;
    value2Sum = 0;
    % variable of Labels
    % check  Product
    productsOutput = {};
    MachineStatus = {};
    usageOfOven = true;
```

```
    for j = 1:length(pr(1,1:6))
        % resilience Scenario adjustment
        %if 25 < Orders{1, "Orders"} < 30
        %    b = Storages.TypeOfProduct == "Printer";
        %    Storages{b, "MaxStorage"}  = 10;
        %end
        %if 10 <= Orders{1, "Orders"} <= 15
        %    a = TimesProcess.Product == pr(1,j) & TimesProcess.Station == "Cleaning & Drying";
        %    TimesProcess{a, "P_T_Face1_min_"} = TimesProcess{a, "P_T_Face1_min_"} * 3;
        %end
        if 43 <= Orders{1, "Orders"} <= 47
            a = TimesProcess.Product == pr(1,j) & TimesProcess.Station == "Milling 1";
            TimesProcess{a, "R_T__min_"} = TimesProcess{a, "R_T__min_"} * 3;
        end
        while Orders{1,[pr(1,j)]} > 0
            % create output
            MachineStatusTime = array2table(zeros(14,4), "RowNames", rowName, "VariableNames", VariableNames);
            % add each table to a dict to separate each table per product
            productsOutput{end+1} = pr(1,j);
            if isempty(output)
                % setting values in the row
                MachineStatusTime.Start_Time = zeros(length(rowName),1) + startWorkingDay;
                %MachineStatusTime.Close_Hour = zeros(length(rowName),1) + closeWorkingDay;
                MachineStatusTime.Stop_time = zeros(length(rowName),1) + stopWorkingDay;
                %MachineStatusTime.Machine_Storage = zeros(length(rowName),1) - 1;
                % set the storage available for each product
                MachineStatusTime("Sawing", "Machine_Storage") = array2table(StorageStatus{1, "Sawing"});
                MachineStatusTime("Laser", "Machine_Storage") = array2table(StorageStatus{1, "Laser"});
                %MachineStatusTime("Printer", "Machine_Storage") = array2table(StorageStatus{1, "Printer"});
                MachineStatusTime("Electrical_Function_Verification", "Machine_Storage") = array2table(StorageStatus{1, "Electrical_Function_Verification"});
                MachineStatusTime("Montage", "Machine_Storage") = array2table(StorageStatus{1, "Montage"});
            else
                % Getting the values from the first processed product
                MachineStatusTime.Start_Time = output{end}.Stop_time;
                %MachineStatusTime.Close_Hour = output{end}.Close_Hour;
                MachineStatusTime.Stop_time = output{end}.Stop_time;
```

```
                MachineStatusTime.Machine_Storage = output{end}.Machine_Storage;
            end
            %disp(MachineStatusTime)
            % add transport and production time to matrix
            for p = 1:length(rowName)
                % get the production time and transportation time
                if p <= 10
                    a = TimesProcess.Product == pr(1,j) & TimesProcess.Station == ↙
rowName(p);
                else
                    a = TimesProcess.Station == rowName(p);
                end
                % actualize values
                MachineStatusTime(rowName(p), "Production_Time") = array2table↙
(table2array(TimesProcess(a, "P_T_Face1_min_")) + table2array(TimesProcess(a,↙
"P_T_Face2_min_")));
                MachineStatusTime(rowName(p), "Transport_Time") = TimesProcess(a,↙
"R_T__min_");
            end
            % if product must be complete
            if stage(j) == 2
                % run simulating
                for q = 1:length(noMainStorage)
                    % adjust if use milling 1 or 2
                    if q == 1
                        if ismember(pr(1,j), ["ProductD", "ProductE", "ProductF"])
                            noMainStorage{q} = "Milling 2";
                        else
                           noMainStorage{q} = "Milling 1";
                        end
                        %MachineStatusTime{noMainStorage{q}, "Start_Time"} =↙
MachineStatusTime{noMainStorage{end}, "Stop_time"};
                    end
                    % define start time
                    if q ~= 1
                        MachineStatusTime{noMainStorage{q}, "Start_Time"} =↙
table2array(MachineStatusTime(noMainStorage{q-1}, "Stop_time")) + seconds(60);
                    end
                    % check if need to wait untill delivery
                    stopTime =  table2array(MachineStatusTime(noMainStorage{q}, ↙
"Start_Time")) + seconds(table2array(MachineStatusTime(noMainStorage{q},"↙
Production_Time")))+ seconds(table2array(MachineStatusTime(noMainStorage{q},"↙
Transport_Time")));
                    b =  TimesProcess.Station == noMainStorage{q} & TimesProcess.↙
Product == pr(1,j);
                    if noMainStorage{q} == "Sawing"
                        % parms for datetime
                        tag = day(MachineStatusTime{noMainStorage{q}, "Start_Time"})↙
+ day(3);
```

```
                        monat = month(MachineStatusTime{noMainStorage{q}, "Start_Time"});
                        jahr = year(MachineStatusTime{noMainStorage{q}, "Start_Time"});
                        StorageStatus{1, "Sawing"} = StorageStatus{1, "Sawing"} - 1;
                        if StorageStatus{1, "Sawing"} < 2
                            % update the matrix
                            MachineStatusTime{noMainStorage{q}, "Start_Time"} = datetime(jahr,monat,tag,8,0,0);
                            stopTime =  datetime(jahr,monat,tag,8,0,0) + seconds(table2array(MachineStatusTime(noMainStorage{q},"Production_Time")))+ seconds(table2array(MachineStatusTime(noMainStorage{q},"Transport_Time")));
                            % fulfill the storage
                            StorageStatus{1, "Sawing"} = 30;
                        else
                            decision = StorageDecisionFunction(StorageStatus, noMainStorage{q}, Orders{1, "Orders"});
                            [maxValues, indices] = max(decision(1,:));
                            if indices ~= 1
                                % update the matrix
                                MachineStatusTime{noMainStorage{q}, "Start_Time"} = datetime(jahr,monat,tag,8,0,0);
                                stopTime =  datetime(jahr,monat,tag,8,0,0) + seconds(table2array(MachineStatusTime(noMainStorage{q},"Production_Time")))+ seconds(table2array(MachineStatusTime(noMainStorage{q},"Transport_Time")));
                                % fulfill the storage
                                StorageStatus{1, "Sawing"} = 30;
                            end
                        end
                    end
                    % set stop time
                    MachineStatusTime{noMainStorage{q}, "Stop_time"} = stopTime;
                end
                % increase product in the storage
                StorageStatus{1, pr(1,j)+labels(1,2)} = StorageStatus{1, pr(1,j)+labels(1,2)} + 1;
            end
            % if product must be just laser
            if stage(j) == 2 || stage(j) == 3
                % run simulating
                for q = 1:length(missingLaser)
                    % define start time
                    if q == 1
                        if MachineStatusTime{missingLaser{q}, "Start_Time"} < MachineStatusTime{noMainStorage{end}, "Stop_time"}
                            MachineStatusTime{missingLaser{q}, "Start_Time"} = MachineStatusTime{noMainStorage{end}, "Stop_time"} + seconds(60);
                        else
                            MachineStatusTime{missingLaser{q}, "Start_Time"} =
```

```
MachineStatusTime{missingLaser{q}, "Start_Time"} + seconds(60);
                        end
                        %MachineStatusTime{"Laser", "Start_Time"} = MachineStatusTime
{missingLaser{end}, "Stop_time"};
                    else
                        MachineStatusTime{missingLaser{q}, "Start_Time"} =
table2array(MachineStatusTime(missingLaser{q-1}, "Stop_time")) + seconds(60);
                    end
                    % check if need to wait untill delivery
                    stopTime =  table2array(MachineStatusTime(missingLaser{q},
"Start_Time")) + seconds(table2array(MachineStatusTime(missingLaser{q},"
Production_Time"))) + seconds(table2array(MachineStatusTime(missingLaser{q},"
Transport_Time")));
                    b =  TimesProcess.Station == missingLaser{q} & TimesProcess.
Product == pr(1,j);
                    if missingLaser{q} == "Laser"
                        % parms for datetime
                        tag = day(MachineStatusTime{missingLaser{q}, "Start_Time"})+
day(1);
                        monat = month(MachineStatusTime{missingLaser{q},
"Start_Time"});
                        jahr = year(MachineStatusTime{missingLaser{q},
"Start_Time"});
                        StorageStatus{1,"Laser"} = StorageStatus{1,"Laser"} - 1;
                        if StorageStatus{1,"Laser"} < 2
                            % parms for the matrix
                            a = Supplier.Supplier == missingLaser{q};
                            indice = find(MachineStatusTime.Row == missingLaser{q})
                            % updating the matrix
                            MachineStatusTime{missingLaser{q}, "Start_Time"} =
datetime(jahr,monat,tag,8,0,0);
                            stopTime =  datetime(jahr,monat,tag,8,0,0) + seconds
(table2array(MachineStatusTime(missingLaser{q},"Production_Time")))+ seconds
(table2array(MachineStatusTime(missingLaser{q},"Transport_Time")));
                            % fulfill the storage
                            StorageStatus{1,"Laser"} = 30;
                        else
                            decision = StorageDecisionFunction(StorageStatus,
missingLaser{q}, Orders{1, "Orders"});
                            [maxValues, indices] = max(decision(1,:));
                            if indices ~= 1
                                % parms for the matrix
                                a = Supplier.Supplier == missingLaser{q};
                                indice = find(MachineStatusTime.Row == missingLaser
{q});
                                % updating the matrix
                                MachineStatusTime{missingLaser{q}, "Start_Time"} =
datetime(jahr,monat,tag,8,0,0);
                                stopTime =  datetime(jahr,monat,tag,8,0,0) + seconds
```

```
(table2array(MachineStatusTime(missingLaser{q},"Production_Time")))+ seconds↙
(table2array(MachineStatusTime(missingLaser{q},"Transport_Time")));
                                % fulfill the storage
                                StorageStatus{1,"Laser"} = 30;
                            end
                        end
                    end
                    % set stop time
                    MachineStatusTime{missingLaser{q}, "Stop_time"} = stopTime;
                end
                % increase product in the storage
                StorageStatus{1, pr(1,j)+labels(1,1)} = StorageStatus{1, pr(1,j) ↙
+labels(1,1)} + 1;
                StorageStatus{1, pr(1,j)+labels(1,2)} = StorageStatus{1, pr(1,j) ↙
+labels(1,2)} - 1;
            end
            fullfilStorage = 0;
            % if product available is in Main Storage
            if stage(j) == 2 || stage(j) == 3 || stage(j) == 4
                % run simulating
                for q = 1:length(afterMainStorage)
                    % paralelize the oven process
                    if usageOfOven == true && (afterMainStorage{q} == "Oven 2" ||↙
afterMainStorage{q} == "Oven 1")
                        afterMainStorage{q} = "Oven 1";
                        usageOfOven = false;
                    elseif usageOfOven == false && (afterMainStorage{q} == "Oven 2"↙
|| afterMainStorage{q} == "Oven 1")
                        afterMainStorage{q} = "Oven 2";
                        usageOfOven = true;
                    end
                    % define start time
                    if q == 1
                        if MachineStatusTime{afterMainStorage{q}, "Start_Time"} <↙
MachineStatusTime{missingLaser{end}, "Stop_time"}
                            MachineStatusTime{afterMainStorage{q}, "Start_Time"} =↙
MachineStatusTime{missingLaser{end}, "Stop_time"} + seconds(60);
                        else
                            MachineStatusTime{afterMainStorage{q}, "Start_Time"} =↙
MachineStatusTime{afterMainStorage{q}, "Start_Time"} + seconds(60);
                        end
                    else
                        MachineStatusTime{afterMainStorage{q}, "Start_Time"} =↙
table2array(MachineStatusTime(afterMainStorage{q-1}, "Stop_time")) + seconds(60);
                    end
                    % set stop time
                    stopTime =  table2array(MachineStatusTime(afterMainStorage{q}, ↙
"Start_Time")) + seconds(table2array(MachineStatusTime(afterMainStorage{q},"↙
Production_Time"))) + seconds(table2array(MachineStatusTime(afterMainStorage{q},"↙
```

```
Transport_Time")));
                    MachineStatusTime{afterMainStorage{q}, "Stop_time"} = stopTime;
                end
                % increase product in the storage
                StorageStatus{1, pr(1,j)+labels(1,1)} = StorageStatus{1, pr(1,j)
+labels(1,1)} - 1;
            end
            % make sure to not repeat over the same order
            Orders{1,[pr(1,j)]} = Orders{1,[pr(1,j)]} - 1;
            MachineStatus{end+1} = MachineStatusTime;
            output = MachineStatus;
        end
    end
    % actualize  the last  machines
    % here we will consider the final product, as the time which is
    % equal for all products, therefore it is the final_product
    % define start time
    for q = 1:length(finalProductStages)
        if MachineStatus{1}.Machine_Storage("Montage") >= 1 && finalProductStages{q}
== "Printer"
            % just pass
        else
            if q ~= 1
                if MachineStatus{1}.Machine_Storage("Montage") >= 1 &&
finalProductStages{q} == "Montage"
                    MachineStatus{1}.Start_Time(finalProductStages{q}) =
MachineStatus{end}.Stop_time("Oven 2") + seconds(60);
                else
                    MachineStatus{1}.Start_Time(finalProductStages{q}) =
MachineStatus{1}.Stop_time(finalProductStages{q-1}) + seconds(60);
                end
            else
                MachineStatus{1}.Start_Time(finalProductStages{q}) = MachineStatus
{end}.Stop_time("Oven 2") + seconds(60);
            end
            % check if need to wait untill
            stopTime =  MachineStatus{1}.Start_Time(finalProductStages{q}) + seconds
(MachineStatus{1}.Production_Time(finalProductStages{q}))+ seconds(MachineStatus{1}.
Transport_Time(finalProductStages{q}));
            b =  TimesProcess.Station == finalProductStages{q};
            if finalProductStages{q} == "Printer" || finalProductStages{q} ==
"Electrical_Function_Verification"
                % parms for datetime
                tag = day(MachineStatus{1}.Start_Time(finalProductStages{q})) + day
(2);
                monat = month(MachineStatus{1}.Start_Time(finalProductStages{q}));
                jahr = year(MachineStatus{1}.Start_Time(finalProductStages{q}));
                StorageStatus{1, finalProductStages{q}} = StorageStatus{1,
finalProductStages{q}} - 1;
```

```
                if StorageStatus{1, finalProductStages{q}} < 2
                    % ajust matrix
                    MachineStatus{1}.Start_Time(finalProductStages{q}) = datetime ↙
(jahr,monat,tag,8,0,0);
                    stopTime =  datetime(jahr,monat,tag,8,0,0) + seconds ↙
(MachineStatus{1}.Production_Time(finalProductStages{q})) + seconds(MachineStatus{1}. ↙
Transport_Time(finalProductStages{q}));
                    % fulfill the storage
                    StorageStatus{1, finalProductStages{q}}  = 30;
                else
                    decision = StorageDecisionFunction(StorageStatus, ↙
finalProductStages{q}, Orders{1, "Orders"});
                    [maxValues, indices] = max(decision(1,:));
                    if indices ~= 1
                        % ajust matrix
                        MachineStatus{1}.Start_Time(finalProductStages{q}) = datetime ↙
(jahr,monat,tag,8,0,0);
                        stopTime =  datetime(jahr,monat,tag,8,0,0) + seconds ↙
(MachineStatus{1}.Production_Time(finalProductStages{q})) + seconds(MachineStatus{1}. ↙
Transport_Time(finalProductStages{q}));
                        % fulfill the storage
                        StorageStatus{1, finalProductStages{q}}  = 30;
                    end
                end
            end
            % set stop time
            MachineStatus{1}.Stop_time(finalProductStages{q}) = stopTime;
        end
    end
    for w = 2:length(MachineStatus)
        % copiar o 1 para o resto
        for q = 1:length(finalProductStages)
            MachineStatus{w}.Start_Time(finalProductStages{q}) = MachineStatus{1}. ↙
Start_Time(finalProductStages{q});
            MachineStatus{w}.Stop_time(finalProductStages{q}) = MachineStatus{1}. ↙
Stop_time(finalProductStages{q});
        end
    end
    for w = 1:length(MachineStatus)
        disp(productsOutput{w})
        disp(MachineStatus{w})
    end
    % save the output
    storage_status_nr = StorageStatus{1, "Cases"};
    Order_nr = Orders{1,"Orders"};
    save("results/AI_"+file_nr, "MachineStatus", "storage_status_nr", "Order_nr",↙
"StorageStatus")
    output = MachineStatus;
    ss = StorageStatus;
```

```
end
```

```
% Title: Development of a resilient Reinforcement Learning-based decision
% algorithm for order scheduling
%
% Author: Fabio Serra Pereira
%
% Description: Here we create the function to simulate the first part of
% the AI. We gave the order quantity of each product to manufacture and
% receive as output which section it must be start.
%
function result = StorageDecisionFunction(StorageStatus, station, order_nr)
    E_sum = 30;
    % resilience Scenario adjustment
    %if 25 <= order_nr <= 30
    %    E_sum = 10;
    %end
    % create the markov chain process
    MDP = createMDP(3, ["not_delivery"; "delivery"]);
    % state 1
    MDP.T(1,2,1) = 1;
    MDP.R(1,2,1) = E_sum;
    MDP.T(1,3,2) = 1;
    MDP.R(1,3,2) = E_sum - StorageStatus{1, station};
    % state 2
    MDP.T(2,2,1) = 1;
    MDP.R(2,2,1) = 0;
    MDP.T(2,2,2) = 1;
    MDP.R(2,2,2) = 0;
    % state 3
    MDP.T(3,3,1) = 1;
    MDP.R(3,3,1) = 0;
    MDP.T(3,3,2) = 1;
    MDP.R(3,3,2) = 0;

    % terminal states
    MDP.TerminalStates = ["s2";"s3"];
    % create environment
    env = rlMDPEnv(MDP);
    % specify a reset function that returns the initial agent state
    env.ResetFcn = @() 1;
    % fix the random generator seed for repoducibility
    rng(0)
    % create Q-Learning Agent
    obsInfo = getObservationInfo(env);
    actInfo = getActionInfo(env);
    qTable = rlTable(obsInfo, actInfo);
    qFunction = rlQValueFunction(qTable, obsInfo, actInfo);
    qOptions = rlOptimizerOptions(LearnRate=1);
    % create Q-Learning agent using this table representation
    agentOpts = rlQAgentOptions;
```

```
    agentOpts.DiscountFactor = 1;
    agentOpts.EpsilonGreedyExploration.Epsilon = 0.9;
    agentOpts.EpsilonGreedyExploration.EpsilonDecay = 0.01;
    agentOpts.CriticOptimizerOptions = qOptions;
    qAgent = rlQAgent(qFunction,agentOpts); %#ok<NASGU>
    % Train Q-Learning Agent
    trainOpts = rlTrainingOptions;
    trainOpts.MaxStepsPerEpisode = 10;
    trainOpts.MaxEpisodes = 500;
    trainOpts.StopTrainingCriteria = "AverageReward";
    trainOpts.StopTrainingValue = 13;
    trainOpts.ScoreAveragingWindowLength = 30;
    trainOpts.Plots = "none";
    doTraining = true;
    if doTraining
        % Train the agent.
        trainingStats = train(qAgent,env,trainOpts); %#ok<UNRCH>
    else
        % Load pretrained agent for the example.
        load("genericMDPQAgent.mat","qAgent");
    end
    Data = sim(qAgent,env);
    cumulativeReward = sum(Data.Reward);
    QTable = getLearnableParameters(getCritic(qAgent));
    result = QTable{1}(1,:);
end
```

```
clc;clear; close all;
% Title: Development of a resilient Reinforcement Learning-based decision
% algorithm for order scheduling
%
% Author: Fabio Serra Pereira
%
% Description: Reading the output from the simulations and generate plots
% to analyze the perfomance of the algos
%
% reading the files and storing in a variable
%
path = "C:\Users\serra\OneDrive\Documentos\TU Darmstadt\Masterthesis\4.stage - Resultados\scenario 0";
ai_path = "\results\AI_";
trad_path = "\traditional\TM_";
format = ".mat";
rowName = ["Milling 1", "Milling 2", "Sawing", "Cleaning & Drying", "Verification", "Laser", "Commissioning", "Powder coating", "Oven 1", "Oven 2", "Printer", "Montage","Electrical_Function_Verification", "Packaging"];
dispName = {"Milling 1", "Milling 2", "Sawing", "Cleaning & Drying", "Verification", "Laser", "Commissioning", "Powder coating", "Oven 1", "Oven 2", "Printer", "Montage","Electrical Function Verification", "Packaging"};
makeSpanName = ["Milling 1", "Sawing", "Laser", "Cleaning & Drying", "Verification" , "Commissioning", "Powder coating", "Oven 1", "Montage" ,"Electrical_Function_Verification", "Packaging"];
makeSpandispName = ["Milling 1", "Sawing", "Laser", "Cleaning & Drying", "Verification" , "Commissioning", "Powder coating", "Oven 1", "Montage" ,"Electrical Function Verification", "Packaging"];
products_with_label = ["ProductA_complete_", "ProductB_complete_", "ProductC_complete_", "ProductD_complete_", "ProductE_complete_", "ProductF_complete_", "ProductA_notLasered_", "ProductB_notLasered_", "ProductC_notLasered_","ProductD_notLasered_" ,"ProductE_notLasered_", "ProductF_notLasered_"];

% for the mse and mae
proc_stoptime = 0;
mse_stoptime = zeros(1,length(rowName));
mse_starttime = zeros(1, length(rowName));
mae_stoptime = zeros(1,length(rowName));
mae_starttime = zeros(1, length(rowName));

% for the makespan
makespan_trad = zeros(1,length(makeSpanName));
makespan_ai = zeros(1,length(makeSpanName));


ai = {};
trad = {};
storage = zeros(length(products_with_label),500);
```

```
for i = 1:1:500
    ai{end+1} = load(path+ai_path+i+format);
    trad{end+1} = load(path+trad_path+i+format);
end

aux = 1;

% find the corresponding values of comparance
for i = 1:length(trad)
    k = 1;
    for j = 1:length(ai)
        if trad{i}.Order_nr == ai{j}.Order_nr && trad{i}.storage_status_nr == ai{j}.
storage_status_nr
            k = j;
            break;
        end
    end
    %
    % analysing the errors: mse and mae
    %
    for j  = 1:length(rowName)
        if j <= 10
            for l = 1:length(trad{i}.MachineStatus)
                pr_trad = trad{i}.MachineStatus{l};
                pr_ai = ai{k}.MachineStatus{l};
                mse_stoptime(j) = mse_stoptime(j) + hours(pr_trad{rowName{j},
"Stop_time"} - pr_ai{rowName{j}, "Stop_time"})^2;
                mae_stoptime(j) = mae_stoptime(j) + hours(pr_trad{rowName{j},
"Stop_time"} - pr_ai{rowName{j}, "Stop_time"});
                mse_starttime (j) = mse_starttime (j) + hours(pr_trad{rowName{j},
"Start_Time"} - pr_ai{rowName{j}, "Start_Time"})^2;
                mae_starttime (j) = mae_starttime (j) + hours(pr_trad{rowName{j},
"Start_Time"} - pr_ai{rowName{j}, "Start_Time"});
                proc_stoptime = proc_stoptime + 1;
            end
        else
            pr_trad = trad{i}.MachineStatus{1};
            pr_ai = ai{k}.MachineStatus{1};
            mse_stoptime(j) = mse_stoptime(j) + hours(pr_trad{rowName{j},
"Stop_time"} - pr_ai{rowName{j}, "Stop_time"})^2;
            mae_stoptime(j) = mae_stoptime(j) + hours(pr_trad{rowName{j},
"Stop_time"} - pr_ai{rowName{j}, "Stop_time"});
            mse_starttime(j) = mse_starttime(j) + hours(pr_trad{rowName{j},
"Start_Time"} - pr_ai{rowName{j}, "Start_Time"})^2;
            mae_starttime(j) = mae_starttime(j) + hours(pr_trad{rowName{j},
"Start_Time"} - pr_ai{rowName{j}, "Start_Time"});
            proc_stoptime = proc_stoptime + 1;
        end
```

```
    end

    %
    % analysing the makespan and generating plot
    %

    for j = 2:length(makeSpanName)
        pr_trad = trad{i}.MachineStatus{1};
        pr_ai = ai{k}.MachineStatus{1};
        makespan_ai(j) = makespan_ai(j) + hours(pr_ai{makeSpanName{j}, "Stop_time"} -↙
pr_ai{makeSpanName{j-1}, "Start_Time"});
        makespan_trad(j) = makespan_trad(j) + hours(pr_trad{makeSpanName {j}, ↙
"Stop_time"} - pr_trad{makeSpanName{j-1}, "Start_Time"});
    end
    %
    % analysing the storage
    %
    for c = 1:length(products_with_label)
        st =  ai{k}.StorageStatus;
        previous = ai{aux}.StorageStatus;
        storage(c,i) = st{1, products_with_label(c)};
        if c >= 7 && i >= 2
            if st{1, "Cases"} == previous{1, "Cases"}
                if storage(c,i-1) < 30
                    storage(c,i) = storage(c,i-1) + 1;
                else
                    storage(c,i) = storage(c,i-1);
                end
            end
        end
    end
    aux = k;
end

for i = 1:length(mse_stoptime)
    mse_stoptime(i) = mse_stoptime(i) / 500;
    mae_stoptime(i) = mae_stoptime(i) / 500;
    mse_starttime(i) = mse_starttime(i) / 500;
    mae_starttime(i) = mae_starttime(i) / 500;
end

for i = 1:length(makespan_ai)
    makespan_ai(i) = makespan_ai(i)/500;
    makespan_trad(i) = makespan_trad(i)/500;
end

% plot the makespan for RL Method
hfig_1 = figure;
```

```
plot(makespan_ai)
title('Makespan of RL Method: Difference of time between stations for Scenario 0')
xlabel("Production Stations")
ylabel('Difference in hours')
xticklabels(makeSpandispName);
hFig_1.WindowState = 'maximized';

%plot the makespan for the Traditional Method
hfig_2 = figure;
plot(makespan_trad)
title('Makespan of Traditional Method: Difference of time between stations for ↙
Scenario 0')
xlabel("Production Stations")
ylabel('Difference in hours')
xticklabels(makeSpandispName);
hFig_2.WindowState = 'maximized';

% plot the MSE and MAE: Stop Time
hfig = figure;
t1 = tiledlayout(2,1);

ax1 = nexttile;
plot(mae_stoptime)
ylabel(ax1, 'MAE (hours)')
x_label_locations = [1:1:14];
ax = gca; % Get current axes being plotted to
ax.XTick = x_label_locations;
Ax.XTickLabel = dispName;
set(ax, 'XTickLabel', dispName);  % Definir os labels do eixo x

ax2 = nexttile;
plot(mse_stoptime)
ylabel(ax2, 'MSE (hours)')

title(t1, 'MAE & MSE: Error Analysis for the Stop time of Scenario 0')
xlabel(t1, "Production Stations")
ax5 = gca; % Get current axes being plotted to
ax5.XTick = x_label_locations;
ax5.XTickLabel = dispName;
set(ax5, 'XTickLabel', dispName);  % Definir os labels do eixo x
%xticklabels(ax2, dispName);
%xticklabels(ax1, dispName);
hFig.WindowState = 'maximized';

% plot the MSE and MAE: Start Time
hfig_5 = figure;
t2 = tiledlayout(2,1);

ax3 = nexttile;
```

```
plot(mae_starttime)
ylabel(ax3, 'MAE (hours)')
ax6 = gca; % Get current axes being plotted to
ax6.XTick = x_label_locations;
ax6.XTickLabel = dispName;
set(ax6, 'XTickLabel', dispName);  % Definir os labels do eixo x

ax4 = nexttile;
plot(mse_starttime)
ylabel(ax4, 'MSE (hours)')
ax7 = gca; % Get current axes being plotted to
ax7.XTick = x_label_locations;
ax7.XTickLabel = dispName;
set(ax7, 'XTickLabel', dispName);  % Definir os labels do eixo x

title(t2, 'MAE & MSE: Error Analysis for the Start time of Scenario 0')
xlabel(t2, "Production Stations")
xticklabels(ax3, dispName);
xticklabels(ax4, dispName);
hFig_5.WindowState = 'maximized';

% plot the Storage Analysis

hfig_4 = figure;
plot(storage(7,:), 'DisplayName', 'Product A')
hold on
plot(storage(8,:), 'DisplayName', 'Product B')
plot(storage(9,:), 'DisplayName', 'Product C')
plot(storage(10,:), 'DisplayName', 'Product D')
plot(storage(11,:), 'DisplayName', 'Product E')
plot(storage(12,:), 'DisplayName', 'Product F')
hold off

title('The Main Storage Evolution for products previous laser stamp: Scenario 0')
xlabel("Orders evolution")
ylabel('Products available in storage')
hFig_4.WindowState = 'maximized';

%
% analysing the computational effort
%
aiT1 = zeros(1,500);
aiT2 = zeros(1,500);
aiT3 = zeros(1,500);
tradT = zeros(1,500);
x = 0;
for i = 1:500
    time_ai = load(path+"\time\Time_"+i+format);
    aiT1(i) = x + minutes(time(time_ai.firstAI));
```

```
    if i == 1
        aiT2(i) = minutes(time(time_ai.secondAI));
        aiT3(i) = minutes(time(time_ai.simulationTime));
    else
        aiT2(i) = aiT2(i-1) + minutes(time(time_ai.secondAI));
        aiT3(i) = aiT3(i-1) + minutes(time(time_ai.simulationTime));
        if i == 50 || i == 100 || i == 150 || i == 200 || i == 250 || i == 300 || i ↙
== 350 || i == 400 || i == 450
            x = x + minutes(time(time_ai.firstAI));
        end

    end
    time_trad = load(path+"\time_trad\Time_"+i+format);
    tradT(i) = minutes(time(time_trad.totalTime));
end

% plot the Storage Analysis

hfig_5 = figure;
plot(aiT1, 'DisplayName', 'Prioritization Algorithm')
hold on
plot(aiT2, 'DisplayName', 'Petri Net Model Selection Algorithm')
plot(aiT3, 'DisplayName', 'Simulation + Fulfill Storage Algorithm')
plot(tradT, 'DisplayName', 'Traditional Method')
hold off

title('Computation Effort: How much time took to run the Algorithms (in minutes) ')
ylabel("Time Evolution (min)")
xlabel('Orders Evolution')
hFig_4.WindowState = 'maximized';

% display the total time
disp("The TM took " + tradT(end) + " min to run")
totalTime = aiT1(500) + aiT2(500) + aiT3(500);
disp("The RLM took " + totalTime + " min to run")
```