

**Faculdade de Engenharia da Universidade do Porto**



# **Reinforcement Learning for Production Scheduling Applications**

João Pedro Silva Casal

Dissertação realizada no âmbito do  
Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Catarina Moreira Marques  
Co-supervisor: Jorge Manuel Pinho de Sousa

30/6/2023





# Resumo

A Revolução Industrial 4.0 teve um impacto significativo nos processos produtivos, impulsionado pela convergência da tecnologia digital, automação avançada e inteligência artificial. No contexto dos ambientes job-shop, onde a programação da produção é complexa e altamente dinâmica, o impacto desta revolução tem sido especialmente notável. Nesta nova era, a aprendizagem por reforço surge como uma abordagem promissora para otimizar a programação da produção, oferecendo soluções adaptáveis e eficientes em tempo real. A este respeito, as novas técnicas de planeamento e programação devem considerar não só a dinâmica na fábrica, mas também a colaboração interfuncional e a integração de dados.

O projeto será construído com base num caso de estudo, que corresponde a um sistema de produção em ambiente de produção job-shop, criado para atender às características-chaves de um ambiente industrial real.

Neste contexto, o objetivo desta dissertação é a combinação de técnicas de Reinforcement Learning com abordagens de simulação para a otimização de um problema de agendamento de tarefas, relativamente à produtividade e a sua comparação com um algoritmo baseado em técnicas de otimização tradicionais, como as meta-heurísticas.

Para além de concluir que a abordagem baseada em Reinforcement Learning proporcionou ótimos resultados de produtividade, esta dissertação também tirou conclusões sobre a robustez destes modelos, a fim de avaliar a sua adaptabilidade quando sujeitos a contextos diferentes, simulando um ambiente do mundo real.

*Página em branco*

# Abstract

The Fourth Industrial Revolution has had a significant impact on production processes, driven by the convergence of digital technology, advanced automation, and artificial intelligence. In the context of job-shop environments, where production scheduling is complex and highly dynamic, the impact of this revolution has been particularly noteworthy. In this new era, reinforcement learning emerges as a promising approach to optimize production scheduling, offering adaptable and efficient real-time solutions. In this regard, new planning and scheduling techniques must consider not only the dynamics within the factory but also interfunctional collaboration and data integration.

The project will be built based on a case study, which corresponds to a production system in a job-shop environment created to meet the key characteristics of a real industrial setting.

In this context, the objective of this dissertation is to combine Reinforcement Learning techniques with simulation approaches for optimizing a task scheduling problem in terms of productivity, comparing it with an algorithm based on traditional optimization techniques such as metaheuristics.

Besides concluding that the Reinforcement Learning-based approach yielded excellent productivity results, this dissertation also drew conclusions about the robustness of these models to assess their adaptability when subjected to different contexts, simulating a real-world environment.

*Página em branco*



# Acknowledgements

Thank you to Engineers Romão Santos, Catarina Marques, and Professor José Pinho de Sousa for their availability, comprehension, and support during the completion of this master's thesis. I am extremely appreciative of their assistance throughout this entire procedure.

In this section, I would also like to thank INESC-TEC, in particular the Centre for Enterprise Systems Engineering and all of its members, for the outstanding facilities and environment provided throughout the dissertation.

I would like to thank the University of Porto's Faculty of Engineering and its professors for everything they have taught me over the past five years, both academically and personally.

In closing, I would like to express my gratitude to all my friends and family for their support and encouragement throughout this period of my life, both in the difficult and the happy times.

*Página em branco*

# Contents

|   |             |
|---|-------------|
| <b>Resumo</b> .....   | <b>v</b>    |
| <b>Abstract</b> .....                                       | <b>vii</b>  |
| <b>Acknowledgements</b> .....                               | <b>ix</b>   |
| <b>Contents</b> .....                                       | <b>xi</b>   |
| <b>List of figures</b> .....                                | <b>xiii</b> |
| <b>List of tables</b> .....                                 | <b>xv</b>   |
| <b>Abbreviations and Symbols</b> .....                      | <b>xvii</b> |
| <b>Introduction</b> .....                                   | <b>19</b>   |
| 1.1 - Contextualization .....                               | 19          |
| 1.2 - Motivation .....                                      | 20          |
| 1.3 - Objectives and Methodology .....                      | 21          |
| 1.4 - Structure of the thesis .....                         | 21          |
| <b>Literature Review</b> .....                              | <b>23</b>   |
| 2.1 - The role of Scheduling .....                          | 23          |
| 2.2 - Reinforcement Learning .....                          | 29          |
| 2.3 - Simulation .....                                      | 33          |
| <b>Problem and Simulation Base Model</b> .....              | <b>37</b>   |
| 3.1 - Job Shop Scheduling Problem .....                     | 37          |
| 3.2 - Discrete-Event Simulation Model - FlexSim .....       | 39          |
| <b>Adopted Methodology</b> .....                            | <b>45</b>   |
| 4.1 - Contextualization of the RL implementation .....      | 45          |
| 4.2 - Neural Network General Architecture .....             | 46          |
| 4.3 - Makespan Minization using RL.....                     | 48          |
| 4.4 - Makespan Minization using OptQuest .....              | 50          |
| <b>Design of Experiments and Results Analysis</b> .....     | <b>51</b>   |
| 5.1 - Results of makespan minimization using OptQuest ..... | 51          |
| 5.2 - Preliminary Experiments.....                          | 52          |
| 5.3 - Results of makespan minimization using RL .....       | 54          |
| 5.4 - Robustness of RL Agent .....                          | 60          |
| <b>Conclusions and Future Work</b> .....                    | <b>63</b>   |
| 6.1 - Conclusions .....                                     | 63          |
| 6.2 - Future Work .....                                     | 64          |

References ..... 65

# List of figures

|  |    |
|--|----|
| Figure 2.1 - Agent-Environment interaction diagram, adapted from [14].....                     | 31 |
| Figure 3.1 - Block diagram of the problem .....  | 38 |
| Figure 3.2 - Diagram of the model used in FlexSim .....  | 38 |
| Figure 3.3 - Properties of an entity in FlexSim.....   | 40 |
| Figure 3.4 - Library in FlexSim .....  | 41 |
| Figure 3.5 - Graphical representation of a Machine in FlexSim.....                             | 41 |
| Figure 3.6 - Representation of the "Parts Creation" Block.....                                 | 42 |
| Figure 3.7 - Representation of the "Transportation and Parts Processing" Block.....            | 43 |
| Figure 3.8 - Routings Table .....  | 43 |
| Figure 3.9 - Model 3D in FlexSim .....   | 44 |
| Figure 4.1 - Agent-Server-FlexSim relationships, adapted from [13] .....                       | 46 |
| Figure 4.2 - SLP Architecture .....  | 47 |
| Figure 4.3 - Action Sorting Example .....  | 48 |
| Figure 4.4 - Representation of the block that assigns values to makespan, reward and done..... | 48 |
| Figure 4.5 - Representation of the block that update the "totalPenChangeover".....             | 49 |
| Figure 4.6 - FlexSim RL module .....   | 50 |
| Figure 5.1 - Settings of OptQuest in FlexSim .....   | 51 |
| Figure 5.2 - Makespan minimization in FlexSim using OptQuest .....                             | 52 |
| Figure 5.3 - Reward related to Experiment 1, for a total of 250k time steps .....              | 54 |
| Figure 5.4 - Reward related to Experiment 2, for a total of 250k time steps ".....             | 54 |
| Figure 5.5 - Reward related to Experiment 3, for a total of 250k time steps ".....             | 56 |
| Figure 5.6 - Reward related to Experiment 4, for a total of 250k time steps ".....             | 57 |
| Figure 5.7 - Parameters of an actions in FlexSim.....  | 57 |

|   |    |
|---|----|
| Figure 5.8 - Reward related to Experiment 5, for a total of 100k time steps ..... | 58 |
| Figure 5.9 - Parameters of an action in FlexSim.....                              | 58 |
| Figure 5.10 - Action Space of Reinforcement Learning in FlexSim.....              | 59 |
| Figure 5.11 - Reward related to Experiment 6, for a total of 100k time steps..... | 59 |
| Figure 5.12 - Variation of observations by 20% .....                              | 60 |
| Figure 5.13 - Reward study, for a total of 100k time steps .....                  | 61 |

# List of tables

|   |    |
|---|----|
| Table 2.1 - The entities, the corresponding attributes, and their options or values in JSSP, adapted from [6] .....                 | 25 |
| Table 2.2 - The JSSP basic types, subtypes, adapted from [5] .....  | 26 |
| Table 3.1 - Duration (in minutes) of the operation to be carried out on each machine, according to the type of product .....        | 39 |
| Table 3.2 - Duration (in minutes) of the total changeover to be carried out on all machines, according to the type of product ..... | 39 |
| Table 3.3 - Parts Transportation (based on destination machine). .....  | 39 |
| Table 5.1 - List of Experiments.....  | 52 |
| Table 5.2 - Table of parameters tests.....  | 53 |
| Table 5.3 - Numerical analyse of makespan referring to Experiment 1 (in seconds) .....  | 55 |
| Table 5.4 - Numerical analyse of makespan referring to Experiment 2 (in seconds) .....  | 55 |
| Table 5.5 - Numerical analyse of makespan referring to Experiment 3 (in seconds) .....  | 56 |
| Table 5.6 - Numerical analyse of makespan referring to Experiment 4 (in seconds) .....  | 57 |
| Table 5.7 - Numerical analyse of makespan referring to Experiment 5 (in seconds) .....  | 58 |
| Table 5.8 - Numerical analyse of makespan referring to Experiment 6 (in seconds) .....  | 59 |
| Table 5.9 - Numerical analyse of makespan .....   | 61 |





# Abbreviations and Symbols

## List of abbreviations

|      |                              |
|------|------------------------------|
| AI   | Artificial intelligence      |
| DL   | Deep Learning                |
| DRL  | Deep Reinforcement Learning  |
| DRs  | Dispatching Rules            |
| IoT  | Internet of Things           |
| JSSP | Job Shop Scheduling Problem  |
| ML   | Machine Learning             |
| MLP  | Multi Layer Perceptron       |
| PPO  | Proximal Policy Optimisation |
| RL   | Reinforcement Learning       |
| SLP  | Single Layer Perceptron      |



# Chapter 1

## Introduction

This introductory chapter presents a contextualisation, the motivation, the objectives and methodologies and the structure of this dissertation.

### 1.1 - Contextualization

One of the oldest and most significant combinatorial optimization issues in operational research and management science is a Job Shop Scheduling Problem. Scholars in engineering and academic sectors have given JSSP a great deal of attention due to the extraordinarily diverse engineering and social application backgrounds [1].

The focus of traditional job shop scheduling is on centralized or semi-distributed scheduling. A smart, distributed production system supported by cutting-edge manufacturing technologies including mass customization, Cyber-Physical Systems, Digital Twin, and SMAC should be the focus of scheduling under Industry 4.0. [2]. The modelling and optimization of intelligent distributed scheduling must become the primary focus of scheduling research.

JSSP is a type of production scheduling problem that involves scheduling the processing of a series of jobs on a set of machines or resources. In a job shop, each job consists of a series of tasks that must be performed in a specific order, and each task requires the use of a specific machine or resource. The goal of JSSP is to find a schedule that minimizes the overall completion time for all the jobs, subject to any constraints on the available resources and the order in which tasks must be completed [3].

Reinforcement Learning is a type of machine learning that involves training an agent to make decisions in an environment in order to maximize a reward. In the context of production scheduling, RL can be used to optimize the scheduling of production processes in order to meet demand while minimizing costs and maximizing efficiency. RL algorithms learn through trial

and error, continuously adjusting their actions based on the feedback received from the environment [1].

In a production scheduling application, the RL agent can be trained to consider various factors such as equipment utilization, raw material availability, and lead times in order to make decisions about when and how to schedule production. By learning from past experiences, the RL agent can improve its decision-making over time and adapt to changing conditions in the production environment [4].

RL has the potential to significantly improve the efficiency and effectiveness of production scheduling, leading to cost savings and increased competitiveness for businesses. However, implementing RL in a production environment can be challenging and requires careful consideration of the goals, constraints, and resources of the organization. Overall, RL has the potential to be a valuable tool for optimizing production scheduling in a variety of industries [1].

RL can be used to solve job shop scheduling problems by training an agent to make decisions about which tasks to schedule on which machines at each time step. The RL agent can consider factors such as the availability of machines, the processing time for each task, and the dependencies between tasks in order to make scheduling decisions that minimize the overall completion time for all the jobs [5].

One challenge in using RL for job shop scheduling is that the environment may be highly dynamic, with new jobs and tasks arriving and resources becoming unavailable at unpredictable times. RL algorithms are suitable for changing environments, but they may require frequent updates and retraining in order to maintain their performance in a dynamic job shop setting.

As a basis for comparison with the results obtained through RL, a method based on meta-heuristics, called OptQuest, was used.

## **1.2 - Motivation**

There are several reasons why RL may be a useful approach for solving production scheduling problems, including job shop scheduling.

One motivation for using RL in production scheduling is to improve efficiency and reduce costs. By optimizing the scheduling of production processes, businesses can minimize the time it takes to complete jobs and reduce the idle time of machines and other resources. This can lead to cost savings in terms of both labour and materials, as well as increased competitiveness in the market.

Another motivation for using RL in production scheduling is to increase flexibility and adaptability. RL algorithms are able to learn from experience and adapt to changing conditions in the environment, making them well-suited to handle unpredictable events such as equipment failures or changes in demand. This can be particularly valuable in a job shop setting, where

new jobs and tasks may arrive at any time and resources may become unavailable unexpectedly.

An additional motivation is the use of a new FlexSim module, which will be explained in subchapter 5.1.

Finally, RL can be a powerful method for improving the overall performance of production systems. By continuously learning and adapting to new information, RL algorithms can help businesses identify and optimize bottlenecks in their production processes, leading to increased productivity and improved quality.

Overall, the use of RL in production scheduling can provide significant benefits in terms of efficiency, adaptability, and performance, making it a valuable tool for businesses in a variety of industries.

### **1.3 - Objectives and Methodology**

The main goal of this dissertation is the optimization of JSSPs by exploring the application of machine learning techniques, in particular RL, and heuristics-based strategies.

Firstly, it is necessary to make a literature review on JSSP and main methods used to answer the problem and respective characterization/definition of the problem that will be addressed.

In a second phase the work will focus on the development of a simulation model to represent the real system, in FlexSim.

Next, directly apply the OptQuest algorithm to the FlexSim model, representing traditional optimization methods. Then, the development of the model prepared for the implementation of RL methods and their respective application.

Finally, the developed models are validated considering specific case studies, comparing results in several scenarios.

### **1.4 - Structure of the thesis**

Regarding the structure and arrangement of this work, it is separated into six chapters.

The first chapter pretends to introduce the theme of the dissertation, including its contextualization, motivation, objectives and methodology.

Chapter [2](#) presents the state-of-the-art of the subjects covered by the dissertation, like the production scheduling, in particular the job-shop problem, its types and approaches, the Industry 4.0, and finally machine learning techniques, especially deep reinforcement learning and its applications.

The description and characteristics of the problem, as well as the description of its methodology, are covered in chapter [3](#), even as the implementation of the optimisation model.

Chapter [4](#) reflects the implementation of Reinforcement Learning in combination with simulation approaches.

The obtained results are presented and discussed in Chapter [5](#).

Chapter [6](#) concludes the dissertation with its conclusions and recommendations for future research.

# Chapter 2

## Literature Review

The purpose of this chapter is to present the results of a bibliographic search in order to facilitate a simple internalisation of the concepts that will be discussed throughout the dissertation.

Scheduling is a decision-making procedure utilized often in several manufacturing and service industries. The purpose is to maximize one or more objectives by the allocation of resources to activities across specified time intervals.

### 2.1 - The role of Scheduling

A variety of resources and tasks might exist inside an organization. The resources may consist of machinery at a workshop, runways at an airport, building workers, processing units in a computing environment, etc. The duties may include production process activities, airport takeoffs and landings, building project phases, computer program executions, etc. Each job may be assigned a particular priority level, earliest feasible start time, and deadline date. Furthermore, objectives might take on a variety of shapes. One target may be the reduction of the time required to finish the final assignment, while another may be the minimization of the number of tasks done past their respective due dates [6].

Scheduling, as a decision-making process, plays a significant part in most of the manufacturing and production systems, as well as in most of information processing settings [6]. It is also essential in the transportation and distribution sectors, as well as other service businesses.

In a production system or service organization, the scheduling function must interact with several other processes. These interactions are system-dependent and may vary considerably between situations. They frequently occur inside a company-wide information system.

Consider the following generic production setting and the significance of its scheduling. In a production environment, orders must be converted into tasks with corresponding due dates. Frequently, these tasks must be executed on the computers in a workcenter in sequential order. Occasionally, the processing of jobs may be delayed if machines are occupied, and pre-emptions may occur when high priority jobs arrive at busy machines. Unanticipated occurrences on the shop floor, such as machine breakdowns or longer-than-anticipated processing times, must also be accounted for since they may have a significant influence on the schedules. In such a setting, the formulation of a precise job plan aids in maintaining operational efficiency and command.

### 2.1.1 - Production Scheduling

In order to give the appropriate background for our JSSP, this part investigates a number of scheduling issues, variations, and solution techniques.

Production scheduling [6] is a major topic in operations management. It is necessary to establish when and on which machine a work should be processed. The manufacturing environment may be exposed to a variety of uncertainties, such as dynamic job arrivals, variations in execution time, and machine failures. The idea is to schedule jobs so that they are as efficient as feasible in terms of some fitness parameter.

Our primary problem will be the JSSP.

According to [2] a scheduling problem may be defined by the triplet  $\{\alpha, \beta, \gamma\}$ ,  $\alpha$  is the machine environment,  $\beta$  is the process characteristics and  $\gamma$  is the objective to be optimized. The goal is to schedule a job and assign a machine to optimise  $\gamma$ , providing that all restrictions  $\beta$  are satisfied.

The machine environments represented by  $\alpha$  can be:

- **Single machine:** It refers to a single machine. It is the simplest scenario.
- **Identical machines in Parallel ( $Pm$ ):** There are  $M$  identical machines. A job  $j$  needs a single operation and can be performed on any machine or a subset of machines.
- **Machines in parallel with different speeds ( $Qm$ ):** There are  $m$  machines in parallel with different speeds. The speed of machine  $i$  denoted by  $v_i$ . So the speed only depends on the machine.
- **Unrelated machines in parallel ( $Rm$ ):** There are  $m$  different machines in parallel. The machine  $i$  can process job  $j$  at speed  $v_{ij}$ . The speed depends on the machine and the job.
- **Flow shop ( $Fm$ ):** There are  $M$  machines in series. Each job has to be processed on each machine. All jobs must follow a given route. A job goes to the next machine queue after its completion on the machine it was [7].

The preceding contexts serve as building blocks for our primary concerns.



### 2.1.2 - JSSP Characterization and Definition

- **Job Shop ( $Jm$ ):** Each job  $j$  has its own set of operations  $O_j$ , which are executed in a specified sequence. In addition, each operation must be executed on a predetermined machine and has a specific processing time  $p_{ij}$ . The  $\beta$  can differentiate between job shops where a job can visit a machine just once or several times. In the latter case,  $\beta$  has the recirculating parameter  $rcrc$ . Moreover, a machine can only process a job at a time [7].

The goal of the problem is to build a scheduler that maximizes a specified objective function for each of the preceding machine settings.

Some common objective functions in production scheduling [8]:

- **Makespan:** completion time of the last job that leaves the system.
- **Maximum flowtime:** maximum flowtime achieved by any of the jobs.
- **Maximum tardiness:** is the maximum tardiness achieved by any of the jobs.
- **Total weighted completion time:** denotes the weighted sum of all completion times.
- **Total weighted tardiness:** is the weighted sum of tardiness values of all jobs.
- **Mean tardiness:** is the mean of the tardiness of all jobs.
- **Total flowtime:** is the sum of flow times of all jobs.
- **Weighted number of tardy jobs:** is the weighted sum of all tardy jobs.
- **Weighted earliness, and weighted tardiness:** is the sum of the total weighted tardiness and the total weighted earliness.
- **Machine utilization:** the difference between the maximum utilization and minimum utilization of all machines.

### 2.1.3 - JSSP Types

As with other types of machine scheduling challenges, JSSP requires two distinct entities. One is the tasks to be processed (e.g., mechanical parts, electrical components, or other objects), which are commonly referred to as jobs; the other is the facilities used to process jobs. In addition to these, the production activities of manufacturing systems frequently involve one or more of the supplemental resources listed below: operators, transportation devices, industrial robots, auxiliary equipment and appliances, warehouses, buffers, and containers [9].

All this is represented in table 2.1.

Table 2.1 - The entities, the corresponding attributes, and their options or values in JSSP, adapted from [10]

| Entity | Attribute    |          |                 |                |                       |
|--------|--------------|----------|-----------------|----------------|-----------------------|
| Job    | Release time | Due date | Processing Time | Auxiliary time | Categorical attribute |

|                                |                        |                             |                               |                                 |               |
|--------------------------------|------------------------|-----------------------------|-------------------------------|---------------------------------|---------------|
|                                | Batch attribute        | Weight attribute (priority) | Technological attribute       | Release mode                    | Delivery mode |
| <b>Machine</b>                 | Functional attribute   |                             | Availability                  | Categorical Attribute           |               |
|                                | Affiliation attribute  |                             | Energy consumption attribute  | Carbon emission attribute       |               |
| <b>Job-Machine</b>             | Processing suitability | Processing model            |                               | Processing successive attribute |               |
|                                | Processing preemption  | Processing reentrancy       |                               | Processing overlap              |               |
| <b>Supplementary resources</b> | Operator-involved      |                             | Robot-involved                |                                 |               |
|                                | Buffer size            |                             | Other supplementary resources |                                 |               |

The attributes of entities are the primary aspects that reflect the distinctions and features of a JSSP model. Furthermore, any of the criteria may often be applied to different JSSP models [9], visible in table 2.2.

Table 2.2 - The JSSP basic types, subtypes, adapted from [9]

| JSSP types                                       | Subtypes                                  |                                  |   |                                     |
|--|---|----------------------------------|---|-------------------------------------|
| <b>Classical JSSP</b>                            |   |                                  |   |                                     |
| <b>Dynamic JSSP</b>                              | Related to time attributes of jobs        | Related to the number of jobs    |   | Related to qualities of jobs        |
| <b>JSSP considering the machine availability</b> | Considering machine breakdown             | Considering periodic maintenance |   | Considering state-based maintenance |
| <b>Flexible JSSP</b>                             | With machine independent processing times |                                  | With machine dependent processing times |                                     |
| <b>JSSP considering batches</b>                  | Parallel batch                            |                                  | Batch decision                          |                                     |
| <b>JSSP considering setup times</b>              | With sequence dependent setup time        |                                  | With sequence independent setup time    |                                     |
| <b>JSSP with nondeterministic or</b>             | With start time dependent                 | With controllable                | With random distribution                | With fuzzy processing times         |

|  |  |                    |                  |                        |
|--|--|--------------------|------------------|------------------------|
| <b>nonconstant processing time</b>         | deteriorating jobs                                   | processing times   | processing times |                        |
| <b>Distributed JSSP</b>                    | In different cells                                   | In different lines |                  | In different factories |
| <b>JSSP with dual-resource constraints</b> | Considering availabilities of machines and operators |                    |                  |                        |

#### 2.1.4 - JSSP Approaches

The problem can be described as follows: Let  $M$  define a collection of machines and  $J$  a set of to-be-processed jobs. Each job  $j \in J$  consists of several operations (set  $O_j$ ) that must be performed on a particular machine. In addition, each operation ( $i \in O_j$ ) has its own processing time  $p_{ij}$  [6]. Also, we assume that a machine cannot be pre-empted and include setup time in the processing time. In the dynamic variant, the production setup may be exposed to many uncertainties, such as the dynamic arrival of jobs, fluctuations in execution time, and machine problems. We will not examine variance in processing times, however dynamic job arrivals will be considered (i.e. the jobs arrival time are unknown in advance) [7].

Multiple strategies were explored to address this problem. Approaches that are proactive generate an offline schedule that is resistant to the variance of execution time events [10]. Therefore, the jobs will be sent according to the offline scheduler's set sequence at the time of execution. However, its performance is heavily dependent on the data collected during offline scheduler production. There is a hybrid predictive-reactive scheduling method in which a portion of the production system is continually modified during operations. Furthermore, there are reactive approaches that make no decisions in advance. When a task arises in real time, a local choice is taken. Usually, this uses simple and constructive heuristics such as Dispatching Rules, Insert algorithms or Bottleneck base heuristics methods to prioritize jobs.

Use priority dispatch rules such as the shortest processing time, the longest remaining total processing time, the earliest delivery time, and the selection of the same machine for the first working process. All operations are sent according to their respective priorities, with the operation with the greatest priority being scheduled first. Therefore, the key technology focuses on finding the optimal priority rules for various real-world challenges. For instance, if minimizing the average flow time of all tasks is the top priority, we may select the rules with the quickest processing time. But if minimizing the maximum delay is of the utmost importance, the earliest delivery time regulations should be utilized. Typically, many priority dispatch rules are constructed concurrently to get a satisfactory solution [1].

The Insert algorithm was created to address the travelling salesman problem. Inserting operations or jobs sequentially into partial schedules might often outperform priority rules.

Later, an improved inserting algorithm (IA) was developed, in which a heuristic approach was used to generate a pre-schedule, and then maintenance tasks were inserted into the pre-schedule scheme to provide dynamic scheduling [1].

Bottleneck-based heuristics methods, such as the Shifting Bottleneck Process and the Bean Search, are more advanced techniques for balancing good outcomes and time consumption. With Shifting Bottleneck Process (SBP), the original problem was simplified and broken into the subproblems of single machine scheduling, each of which was later handled independently. In each round of iterations, one bottle machine was selected and the process order of all the jobs on that bottle machine was set, allowing the procedure to be repeated until full machine orders were determined [1].

Sometimes, constructive methods, such as dispatching rules, insert algorithms, and bottleneck-based heuristics, can acquire a JSSP solution extremely fast, but when the issue is complex, infeasible solutions may be developed. Complex heuristic criteria are typically required in order to increase the quality of results. For a complex system, there are so many rules that constrain each other and are sometimes conflicting or caught in a loop. Therefore, it is challenging to discover a workable solution that satisfies all rules.

### 2.1.5 - The impact of Industry 4.0

Industry 4.0, also known as the Fourth Industrial Revolution, refers to the current trend of automation and data exchange in manufacturing technologies, including developments in artificial intelligence, the Internet of Things, and cloud computing. These technologies have the potential to greatly impact production scheduling, particularly in JSSP[6].

One key aspect of Industry 4.0 that can impact job shop scheduling is the use of real-time data and analytics. With the ability to collect and analyse data from all aspects of the production process, companies can make more informed and accurate scheduling decisions. For example, data on machine utilization, production rates, and worker productivity can be used to optimize the scheduling of tasks and resources.

Another aspect of Industry 4.0 that can impact job shop scheduling is the use of advanced technologies such as robotics and machine learning. These technologies can help automate certain tasks and processes, improving efficiency and freeing up human workers to focus on more complex and value-added tasks. This can also allow companies to schedule production in a more flexible and responsive manner, as they can quickly adjust to changing demand or other factors [6].

Overall, the impact of Industry 4.0 on production scheduling in JSSP can be significant, as it allows companies to make more informed and efficient decisions about how to allocate tasks

and resources. This can help increase productivity and competitiveness, as well as reduce costs and improve customer satisfaction.

## 2.2 - Reinforcement Learning

In this subchapter, will be explored the rich body of literature surrounding reinforcement learning techniques.

### 2.2.1 - Machine Learning

The fact that JSSP is an NP-hard problem makes it difficult to solve using conventional optimizers [11]; as a result, typical optimizers are unable to find an optimal solution in a reasonable amount of time[12]. Consequently, scheduling strategies are utilised rather than complete optimizers.

As previously demonstrated, Dispatching rules (DRs) have attracted considerable interest as a result, although they are difficult to manually construct. It is challenging to obtain a rule with good performance since DRs tend to be myotic [13]. Consequently, Machine Learning is employed to achieve these DRs, overcoming the obstacles associated with manually developing one.

Basically, Machine Learning aims to learn based on previous data and make predictions or decisions for the future [14].

Enumerating some of its applications:

- Analyse product images on a production line to automatically classify them;
- Detect tumours thought brain scans;
- Summarize long documents automatically.

In accordance with their learning methods, Machine Learning systems may be divided into three categories:

- **Supervised Learning:** An external supervising Agent provides a sequence of samples (inputs) with the correct response (outputs), and then, based on training, the implemented algorithm generalises the correct answer to another set of inputs [15].
- **Unsupervised Learning:** The developed algorithm attempts to find similarities between inputs and classify them accordingly. Clustering is one of the most well-known procedures [15].
- **Reinforcement Learning:** Between Supervised Learning and Unsupervised Learning. The algorithm is informed about the response's quality, but not how to improve it. So, it is vital for the Agent to research and test alternatives until he or she learns how to generate a higher quality answer [15]

In this dissertation, it is pertinent to analyse, essentially, the Reinforcement Learning field.

## 2.2.2 - Reinforcement Learning Concepts

One of RL's enduring issues is learning how to manage Agents directly from high-level sensory input, such as vision and voice.

Contrary to other kinds of Machine Learning, such as Supervised Learning, the Agent is not explicitly instructed on what actions to perform[16]. Therefore, at the conclusion of the learning phase, the Agent will need to determine which of his previously deliberated acts resulted in bigger rewards. Intriguingly, the present activities will effect the relevant reward as well as future rewards.

The two most significant aspects of RL are "trial-and-error search" and "delayed reward".

Unlike Supervised Learning, which is a way of learning based on examples provided taking into account the knowledge of an external supervisor, this is not applicable to an interactive learning[16]. This is because, for most interactive issues, it is hard to acquire samples of the required behaviour that are both right and representative of all situations in which the Agent must operate.

- **Agent:** The Agent is the entity responsible for making decisions, and it is referred to as the "learner" and "decision maker". In particular, the Agent and the environment interact in discrete time intervals ( $t = 0, 1, 2, \dots$ ). As shown in figure 2.1, the Agent gets, for each  $t$ , a representation of the environment's state,  $s_t$ , such that  $s_t \in S$  represents the set of all conceivable states. The Agent receives a reaction in the form of a reward  $r_{t+1} \in R$  and a new state of the environment,  $s_{t+1}$ , which serves as feedback for the subsequent decision  $a_{t+1}$  [17].
- **Environment:** The environment is responsible for informing the Agent about the current condition and the reward earned for the action conducted earlier. It also provides a list of all potential states to the Agent [16].
- **Action:** The action is the consequence of the Agent's decision. The purpose is to identify the optimal solution, which equates to selecting the action that will yield the greatest reward given that each action generates varying reward values [16].
- **Reward:** The reward is a form of feedback that allows the Agent to evaluate the results of his actions in the previous stage. It is essential to reiterate that the purpose is to achieve the highest possible accumulation of rewards, bearing in mind that obtaining a large reward in one condition does not always indicate that the eventual accumulation of rewards would be the best. This is because, despite the fact that a certain reward in a given condition is the biggest, it may lead to a less-than-ideal scenario in the future and negatively impact subsequent rewards [18].

- **Policy  $\pi$ :** It is a mapping approach that enables the Agent to determine the next action to do in order to accrue a substantial reward over time. The RL explains how the Agent might modify its policy, taking its experience into consideration.

In addition to policy, value function, and model, The Agent can also be classed. A policy,  $\pi$ , is a mapping of states  $s \in S$  and actions  $a \in A(s)$  for the probability  $\pi(s, a)$  of performing an action at the time of a state  $s$ .  $V^\pi(s)$  continues to represent the worth of the state  $s$  according to a policy  $\pi$  [17].



Figure 2.1 - Agent-Environment interaction diagram, adapted from [17]

Therefore, the RL lets the Agent to determine what action to take, taking his own experience into consideration. So that he may subsequently carry out his action, the Agent will have to review his prior judgements and determine if he actually received a favourable payoff. In this way, a new paradigm emerges in which the Agent, in addition to exploring his existing knowledge from past scenarios, must also explore new, never-before-made options to determine if receives a greater benefit. Because in both circumstances, the Agent will fail (get a lower reward) and the answer lies within the Agent's critical capacity, he must do several tests in order to locate the solution that yields a final value matching to the greatest reward [17].

### 2.2.3 - Deep Reinforcement Learning

Deep reinforcement learning (DRL) is an extension of reinforcement learning (RL) that incorporates deep neural networks with traditional RL algorithms. DRL introduces deep learning techniques to manage high-dimensional input spaces, such as images or raw sensor data, by leveraging neural networks for representation learning, whereas RL concentrates on learning optimal actions in a given environment[19].

The incorporation of deep neural networks into DRL offers a number of significant advantages over conventional RL approaches:

- **Representation Learning:** Deep neural networks can autonomously learn valuable features from unprocessed input data, allowing DRL algorithms to directly process complex observations. This eliminates the requirement for manual feature engineering, making DRL more adaptable and applicable to a wider variety of problems [20].

- **Function Approximation:** As function approximators, DRL uses deep neural networks to estimate action-value functions or policies. These networks are capable of capturing intricate patterns and nonlinear relationships, enabling DRL agents to model complex behaviours and make accurate predictions [20].
- **End-to-End Learning:** By combining representation learning and function approximation, DRL agents can generate actions directly from unprocessed sensory input, facilitating end-to-end learning. This simplifies the training pipeline and reduces reliance on components designed by specialists [20].

In the realm of DRL, numerous algorithms have been proposed to tackle the challenges of high-dimensional state spaces, complex action spaces, and sample efficiency. Some notable algorithms include Deep Q-Networks (DQN), Trust Region Policy Optimization (TRPO), and Asynchronous Advantage Actor-Critic (A3C). These algorithms have demonstrated impressive results in different tasks and have contributed to the success of DRL. Among the wide array of DRL algorithms, Proximal Policy Optimization (PPO) has gained significant attention and emerged as one of the most widely used and effective algorithms.

#### 2.2.4 - Proximal Policy Optimisation

PPO is an on-policy algorithm that maximizes the expected cumulative reward by optimizing a policy function. It employs a trust region strategy to ensure that policy revisions are relatively conservative, thereby preventing radical changes that could destabilise the learning process [21].

The key characteristics of PPO include:

- **Policy Optimisation:** PPO optimizes the policy function explicitly in order to maximize the expected reward. It accomplishes this by accumulating data iteratively through interactions with the environment and using this information to update the policy parameters [22].
- **Proximal Policy Optimization:** PPO employs a substitute objective function with a proximity constraint. This constraint prevents policy updates from deviating too significantly from the original policy, thereby assuring more stable and reliable updates [22].
- **Multiple Epochs:** On the collected data, PPO performs multiple iterations of policy updates. It repeatedly samples mini-batch sizes from the data and executes multiple optimization steps, which serves to further stabilize the learning procedure and enhance sample efficiency [21].
- **Clipped Surrogate Objective:** PPO includes a clipping mechanism in the substitute objective function to limit the extent of the update. This clipping



prevents substantial policy modifications that could result in unstable training or calamitous forgetting of previously learned behaviours [21].

PPO is a popular and effective algorithm in the field of DRL as a result of its stability, simplicity, and strong empirical performance. It achieves a balance between sample efficacy and policy optimisation, making it applicable to a broad array of RL problems.

## 2.3 - Simulation

Modelling is a method for solving real-world context issues. The majority of the time, we cannot afford to conduct experiments and tests on actual products in order to find the optimal answer, as these objects are generally expensive and even uncommon. Thus, simulation assumes a fundamental role in this context [23].

In addition to the objective of modelling, this strategy has other advantages, such as [24]:

- Simulation models enable us to analyse systems and identify solutions where analytical models fall short;
- It permits the testing of new policies, operational processes, decision rules, and information flows without modifying the actual system;
- Discover the bottlenecks.

Unfortunately, despite all of these advancements, it might be difficult to understand the final results at times. The fact that the simulation takes extensive training is one of the simulation's negative aspects.

### 2.3.1 - Simulation Modelling

This section covers the fundamental components of a simulation system so that its composition may be better comprehended [25]

- **System:** A set of things that interact in order to fulfil the specified goals.
- **Model:** A system model is an abstract representation of a system that describes its state, entities, processes, events, and activities, among others.
- **System State:** A set of variables required to describe the system.
- **Entity:** A set of variables required to describe the system.
- **Attribute:** An entity property.
- **Event:** An entity property.
- **Activity:** The amount of time a task requires to be completed, which is known before it begins.
- **Delay:** The extra time interval, whose endpoint is solely known.
- **Clock:** A variable that reflects the duration of simulation.

A method is a framework for representing real-world systems in simulation models. Agent-Based, System Dynamics, and Discrete-Event Modelling are the three classifications applicable to simulation modelling approaches. The application of each of these techniques relies on the system to be created and its goals [26].

### 2.3.2 - Simulation Modelling Paradigms

- **Agent-based:** It belongs to the category of computer models used to simulate the behaviours and interactions of autonomous Agents (individual or collective). Despite a lack of understanding of the system's behaviour and the inability to describe the process flow, the primary aim is to verify and analyse the impact of the Agents on the system as a whole [27].
- **System Dynamics:** This model is a perspective and a collection of conceptual tools that enable the comprehension of the structure and dynamics of complex systems. This model is also a rigorous modelling technique that permits the construction of formal simulations of complex systems and their use in the formulation of more effective policies and organisations [27].
- **Discrete-Event Modelling:** It makes it possible to represent a system as a discrete series of occurrences in time. Each event occurs at a predetermined time and alters the state of the system. This system demands that modelling be seen as a process consisting of a series of Agent-performed activities [27].

The paradigm used in this dissertation was the Discrete-Event Modelling.

### 2.3.3 - Construction Steps of a Simulation Model

- **Problem Formulation:** The problem must be clearly stated so that there is no room for ambiguity. There are still instances in which an entire or partial reformulation of the problem is required [24].
- **Establishment of Objectives and General Project Plan:** The objectives will be the questions to be answered through the simulation. After deciding which simulation method is the most suitable, it is necessary to list a series of alternatives to the simulation and find ways to evaluate the effectiveness of these same solutions. Besides the objectives defined for the end of each state, it is also important to mention in the plan the number of people involved, as well as the associated costs and the estimated time for each phase of the project [24].
- **Model Conceptualisation:** It is not feasible to establish a priori the instructions that would result in a good model, but there are some viewpoints that must be adhered to in order for the model to be successful. To get good outcomes, it is vital to define a simple model and then make the required modifications step by step [24].

- **Data Collection:** The collecting of data is closely related to the creation of the model, since the acquired data will serve as model input. While data collection consumes significant amounts of time, and as this is a crucial element of the model's development, it is crucial to begin it as soon as feasible [24].
- **Model Translation:** During this phase, the model is translated into a simulation language using specialised software. In the specific case of this dissertation, the software used is the FlexSim program [24].
- **Verification and Validation:** This stage allows verification and debugging tests to be run to determine whether the software is ready for the simulation model. By comparing the model to the existing system's behaviour, it is beneficial to use this feedback to enhance the model. The procedure is done repeatedly until a good outcome is achieved [24].
- **Experimental Design:** The previously defined options from the "Establishment of Goals and General Project Plan" phase that must be simulated must be identified [24].
- **Production and Analysis of results:** Following several testing of the model with various data, the resulting analyses are utilised to estimate the performance of the simulated system [24].
- **Documentation:** It is important to submit two sorts of data following the simulation: programme and progress. In the event that others utilise the programme in the future, the programme documentation describes its operation and behaviour, as well as other basic characteristics. This is crucial for the model's legitimacy and certification in terms of the progress report [24].
- **Implementation:** The implementation's success will depend on each of the preceding phases [24].



# Chapter 3

## Problem and Simulation Base Model

This chapter defines the essential information for testing a RL technique in a JSSP instance by describing the problem and providing a full analysis of it. This chapter also defines the simulation elements common to the collection of to-be-executed implementations.

### 3.1 - Job Shop Scheduling Problem

Operations scheduling is the process of planning and organising the activities and resources required to complete the production of goods or services. This includes determining the sequence of operations, assigning adequate resources to each operation, and estimating the time and cost required for each step.

In a factory, operations scheduling usually involves dividing the production process into different modules or process into different modules or departments, each of which is responsible for execution of a specific set of operations on the product. For example, a factory producing automobiles may have a module for welding, another for painting, and another for assembly. However, car factories organise these modules into a production line in order to maximise line the profitability of the production line.

Once the product has undergone each of the necessary operations in a particular module, it is then transported to the next module for further processing. In the scenario introduced above, transportation introduces an additional logistical challenge, as the plant has to ensure that the product is moved efficiently and on time to avoid delays in the overall to avoid delays in the overall production process.

In this case, the manufacturing process is divided into 3 non-contiguous factory modules: department A, the initial module of the factory; department B and department C.

The first department consists of 2 machines in parallel (M1 and M2). Machine 1 (M1) uses 3 different tools (F1, F2 and F3). Machine 2 (M2) uses 2 different tools (F4 and F5).

The department B is composed of Machine 3 (M3), using 2 tools (F6 and F7).

Finally, the department C packages the products produced so that they can be transported directly to the shops for sale, according to the orders placed by each retailer. Machine 4 (M4) is used for packaging.

As the various modules are not in the same place it is necessary to transport the product to the module where it will have to operate next. For this there are 3 transport services: operator A, B and C. There is only one operator to directly connect two modules, so it will

always be necessary to consider the return time of the operator before it can make a new journey. The structure of the plant is summarized in the diagram represented in Figure 3.1.

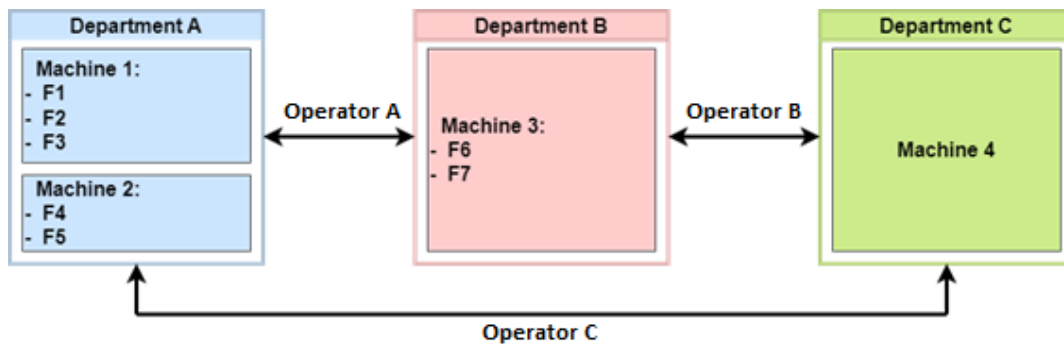


Figure 3.1 - Diagram Representing the Factory Structure

The model used in FlexSim that represents the diagram of the previous figure, is shown in the following figure.

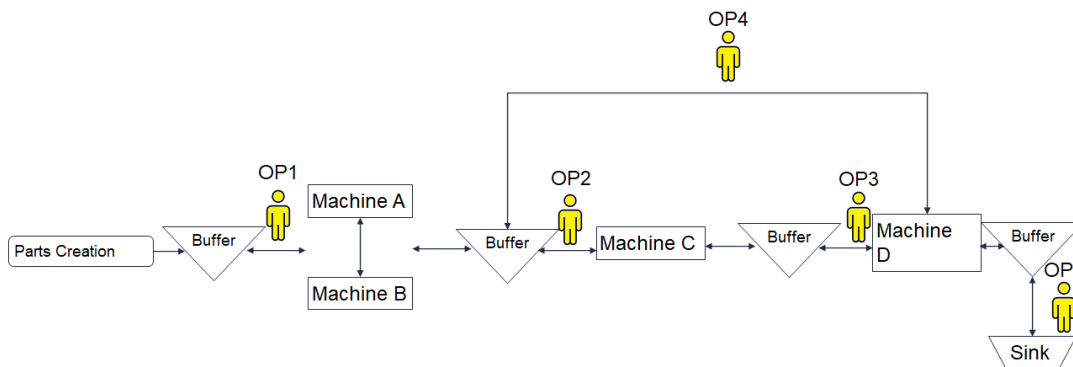


Figure 3.1 - Block diagram of the problem

### 3.1.1 - Problem Instance

The factory produces essentially 4 products for sale to retailers. The sequence of operations required to produce each of these products can be summarised by:

- **Product 1: M1F1 - M2F4 - M3F6 - M4;**
- **Product 2: M1F2 - M2F4 - M3F7 - M4;**
- **Product 3: M1F3 - M2F5 - M4;**
- **Product 4: M1F3 - M2F5 - M3F7 - M4;**

Firstly, the operation times and the setup costs for each machine necessary to produce each of the 4 types of products were defined.

Table 3.1 - Duration (in minutes) of the operation to be carried out on each machine, according to the type of product

| Type of product | Machine 1 | Machine 2 | Machine 3 | Machine 4 |
|-----------------|-----------|-----------|-----------|-----------|
| 1               | 10        | 3         | 9         | 5         |
| 2               | 8         | 3         | 7         | 5         |
| 3               | 7         | 4         | 5         | -         |
| 4               | 7         | 4         | 7         | 5         |

Then, tool changeover results in costs and machine downtimes with fixed durations. Thus, tool changeover brings with it an interval in which the machine may not be performing useful work.

Table 3.2 - Duration (in minutes) of the total changeover to be carried out on all machines, according to the type of product

| Type of product | 1   | 2   | 3   | 4   |
|-----------------|-----|-----|-----|-----|
| 1               | -   | 3   | 4,5 | 6,5 |
| 2               | 3   | -   | 2,5 | 2,5 |
| 3               | 4,5 | 2,5 | -   | -   |
| 4               | 6,5 | 2,5 | -   | -   |

Finally, each operator has fixed machines with which they can move.

Table 3.3 - Parts Transportation (based on destination machine)

| Machine 1  | Machine 2  | Machine 3  | Machine 4                 | Machine 5  |
|------------|------------|------------|---------------------------|------------|
| Operator 1 | Operator 1 | Operator 2 | Operator 3/<br>Operator 4 | Operator 5 |

It was considered that only one order of 3 pieces of each product type was made, that is 12 pieces in total.

The objective function used will be the minimization of the makespan.

### 3.2 - Discrete-Event Simulation Model - FlexSim

Following the steps outlined in section 2.8.2 to construct a simulation model, it is possible to simulate the behaviour of the investigated system using the discrete event simulation category of the FlexSim software utility.

This software programme enables for a highly accurate and realistic simulation of the factory's behaviour. Thus, it is possible to simulate the release timings of a component as well as the processing of each machine, allowing us to draw accurate conclusions concerning the system's behaviour.

### 3.2.1 - Entitites

These objects represent the parts that run through each one of the machines. As shown in figure 3.3, by selecting an object, you can access its part type (type) and order posting number (Sequence).

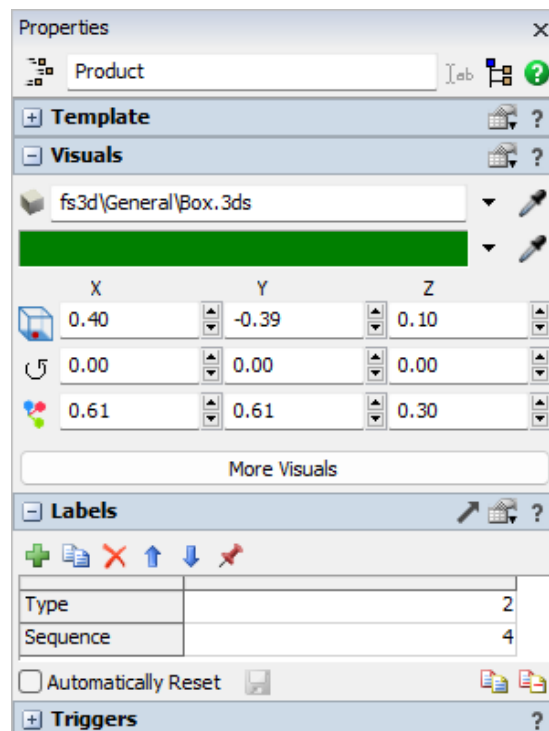


Figure 3.3 - Properties of an entity in FlexSim



### 3.2.2 - Resources

The FlexSim Library provides a vast assortment of resources, as shown in figure 3.4 below. In this JSS problem, the resources will specifically represent the machines.

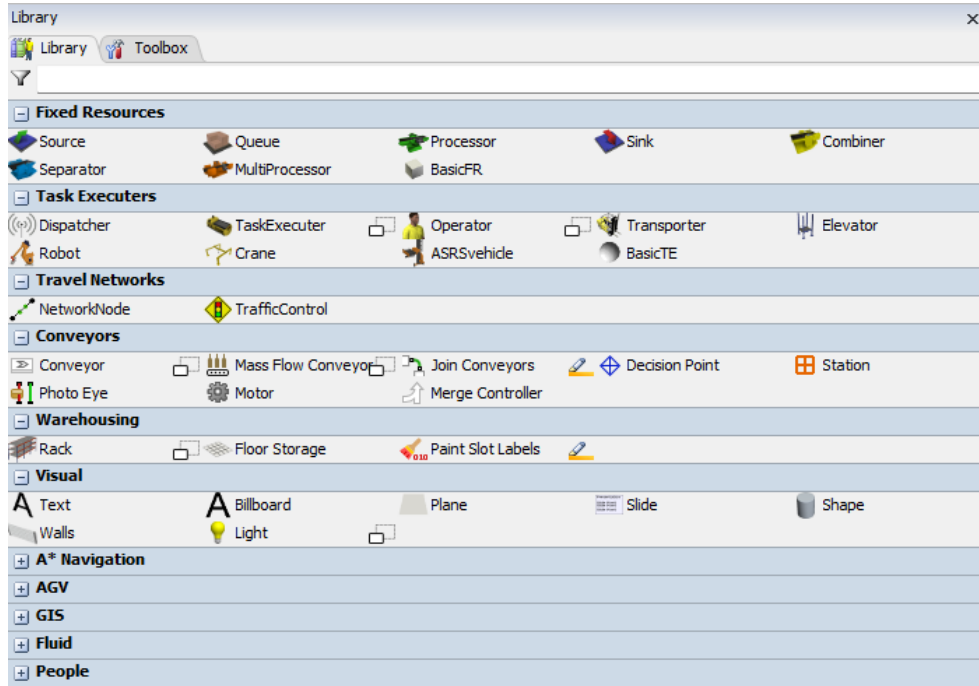


Figure 3.4 - Library in FlexSim

### 3.2.3 - Machines

The machines are reflected by Fixed Resources and it is important to mention that they will be represented by native Processor resources, illustrated in figure 3.5. It is also possible to obtain concrete information about the inputs and outputs of the parts in each machine.

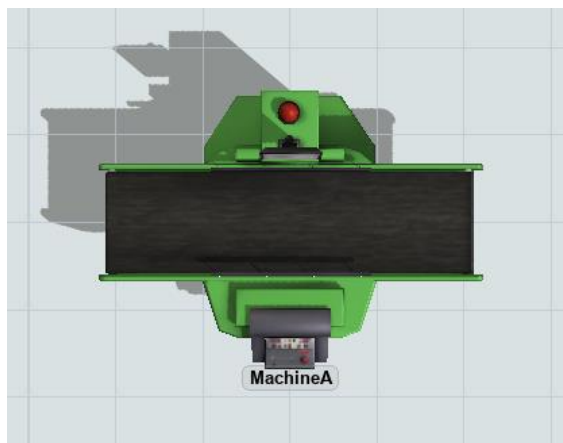


Figure 3.5 - Graphical representation of a Machine in FlexSim

### 3.2.4 - Transport

Transport times were not considered for the simulations to simplify them.

### 3.2.5 - Load

All components enter the system via the Source; however, in this instance, this resource has been supplanted with a Queue, which permits the same behaviour. The primary advantage of employing a Queue object rather than a Source is the ability to observe the accumulation of components throughout the simulation.

### 3.2.6 - Unload

Sink is the location of the final discharge of items.

### 3.2.7 - Process Flow

3D model and Process Flow are the two most important instruments for creating simulation models. Regarding Process Flow, it overlaps the 3D model under all conditions.

However, cooperation between the two is essential, although in this instance, the transfer of entities and resources will be defined in the Process flow.

In general, all transport systems consist of two major components: the parts creation (figure 3.6), and the transportation and parts processing (figure 3.7). The phases of creation and processing are shared by all models discussed in this dissertation [16].

- **Parts Creation**

This block causes the parts to be created according to the sequence established before starting the whole process.

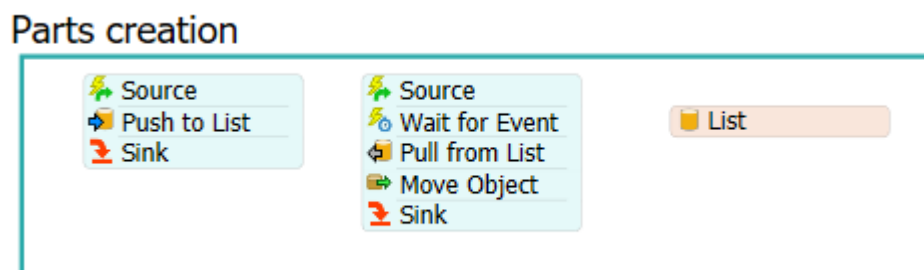


Figure 3.6 - Representation of the "Parts Creation" Block

- **Transportation and Parts Processing**

### Transportation and Parts processing

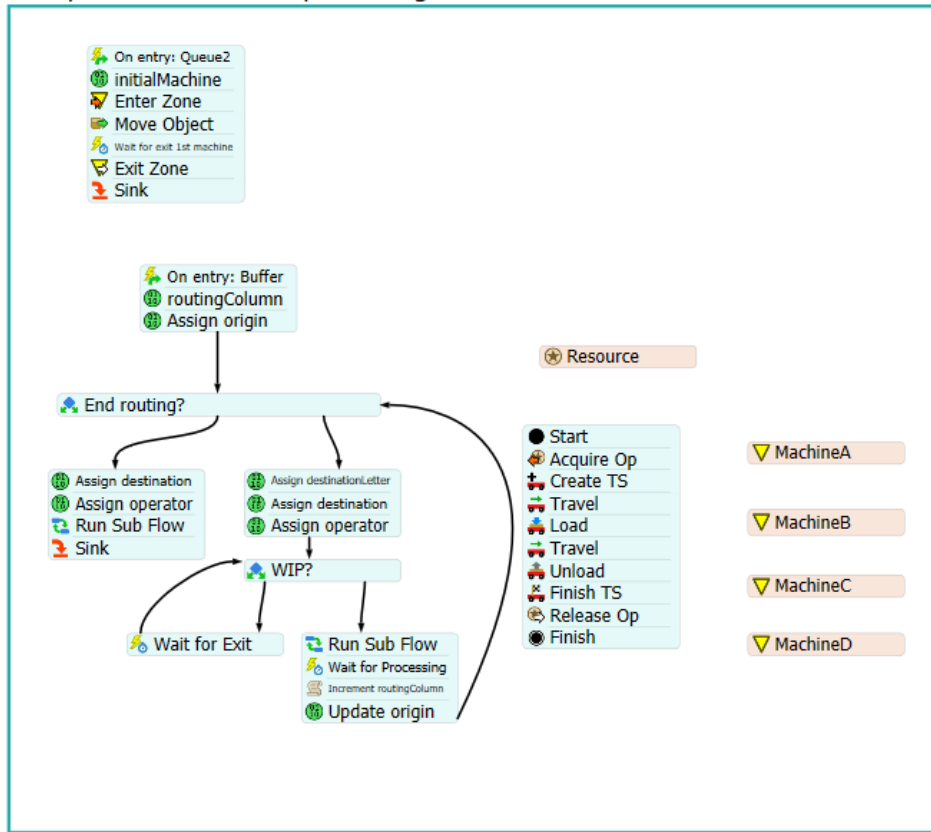


Figure 3.7 - Representation of the "Transportation and Parts Processing" Block

Depending on the type of the current part, the block permits accessing the Routings Table (figure 3.8) and identifying the correct line in order to follow the machine sequence for the type of part in question. However, it must be noted that the part will be just moved from the current machine to the next one if there are no parts in the respective machine, because its maximum capacity is unitary. These restrictions are defined by the creation of zones, focused on each one of the existing machines. Processing times and changeover times are also defined in tables, which are read by the model machines depending on the current part type.

| Routings |           |           |           |           |
|----------|-----------|-----------|-----------|-----------|
|          | Machine 1 | Machine 2 | Machine 3 | Machine 4 |
| 1        | A         | B         | C         | D         |
| 2        | A         | B         | C         | D         |
| 3        | A         | B         | D         | -         |
| 4        | A         | B         | C         | D         |

Figure 3.8 - Routings Table

### 3.2.8 - 3D Model

In the next figure, it is possible to see the 3D model of the problem under analysis. By visualizing the 3D model in FlexSim, users can observe the dynamics of the simulated system,

the flow of materials, the interactions between elements, and perform analyses to optimize system performance. 3D simulation provides a more realistic and interactive representation of the system, allowing users to identify bottlenecks, test different scenarios, and make informed decisions to improve efficiency and productivity.

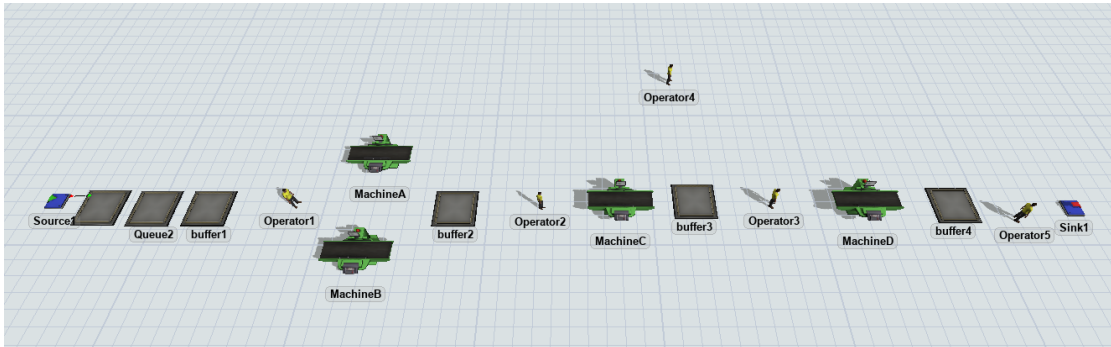


Figure 3.9 - Model 3D in FlexSim

# Chapter 4

## Adopted Methodology

This chapter provides insight into the implementation of the RL model in this problem, but also into the implementation of OptQuest.

### 4.1 - Contextualization of the RL implementation

The key difference between this implementation and the one presented in Section 4.4 is the addition of a third element (Agent), which is responsible for decision making. The Agent is educated with Reinforcement Learning algorithms to define, like the previous system, the sequence of parts entering the simulation.

In addition to this capability, this Agent may also close and reset the FlexSim simulation application.

As illustrated in figure 4.1, the Server is merely an intermediary between this third entity and the Flexsim simulation environment. Similarly, to the simulation environment, which was previously defined as a client, the Agent will also be a client, while the Server will serve as the "go-between".

In other terms, the Agent is the programme that contains the code that executes certain functionalities on the training environment (FlexSim) [16].

In this dissertation, the Server is already included in FlexSim, which is a new module of this simulation programme.

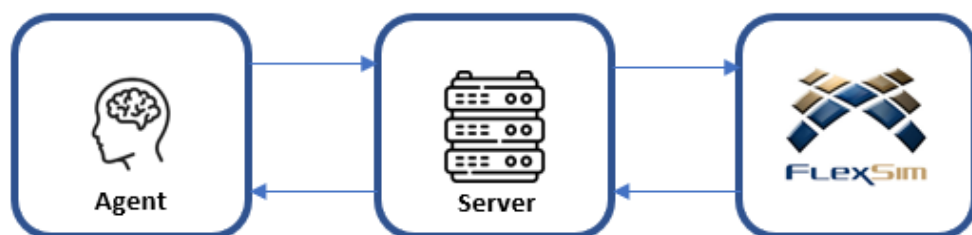


Figure 4.1 - Agent-Server-FlexSim relationships, adapted from [16]

In accordance with this logic, there are 3 possible forms of interaction between the Agent and the simulation environment:

- **Reset:** Its use enables the Agent to request a reset of the FlexSim;
- **Close:** Allows the FlexSim to be closed;

- **Step:** Grants the transmission of an action, representing the total production of the 12 parts. The action decision is modified and enhanced by RL algorithms of the StableBaselines framework training a neural network. This framework is an improved, more stable variant of the OpenAI Baselines algorithms [16].

The only distinction between the available RL algorithms is the strategy used to redefine the policy defining the action to be taken for a given system state. The Proximal Policy Optimization (PPO) algorithm is the current state-of-the-art in the domain of this dissertation; therefore, it was deemed most suitable for the present problem. However, it is possible to use other algorithms in the future with only minor modifications to the Agent's source code.

This Agent comprises a neural network that receives simulation environment observations as input data and outputs the representative action of the Production Sequence.

The sections 4.2 and 4.3 present the problems regarding Reinforcement Learning treated in this dissertation.

Regarding the initial query regarding the Agent-Server-Flexsim relationships, it is necessary to establish communication between the three entities, example code written in Python was used, using the *OpenAI Gym toolkit* and the *Stable-Baselines3* implementations of reinforcement learning algorithms, provided by FlexSim itself.

The 3 python scripts used were:

- **Flexsim\_env.py:** *FlexSimEnv* is a subclass of *gym.Env* that implements its customised environment interface. This class contains methods and properties that use sockets to launch and communicate with FlexSim.
- **Flexsim\_training.py:** A single instance of the `main()` method demonstrating the use of the *FlexSimEnv* class to train a *stable\_baselines3* reinforcement learning algorithm, save the trained model, and evaluate the trained model through a direct connection to the FlexSim environment.
- **Flexsim\_inference.py:** A *FlexSimInferenceServer* class that implements methods for handling HTTP requests and is a subclass of *BaseHTTPRequestHandler*. The `main()` method demonstrates importing the trained model that was preserved during training and hosting an HTTP server that can utilise the trained model to provide an action in response to a system observation. This is a basic demonstration example that is not recommended for use in production.

## 4.2 - Neural Network General Architecture

Originating from the study of the human nervous system, neural networks are distinguished by their high complexity and nonlinearity. The human brain's remarkable adaptability stems

from its ability to aggregate and organise its neurons in order to accomplish quite complex tasks [28]

The primary objective of the PPO algorithm is to ensure that the new policy is not radically different from the previous one. The version in use is an implementation of *Stable Baselines*, and in addition to supporting vectorized environments, it permits discrete, box, multidiscrete, or multibinary types for observations and resulting actions. In this specific case, the observation assumes the discrete type (0-27) and the action admits the multidiscrete type (12). The next figure 4.2 represents the network architecture of input and output layers, a single layer perceptron architecture, being the base of the NN of this dissertation.

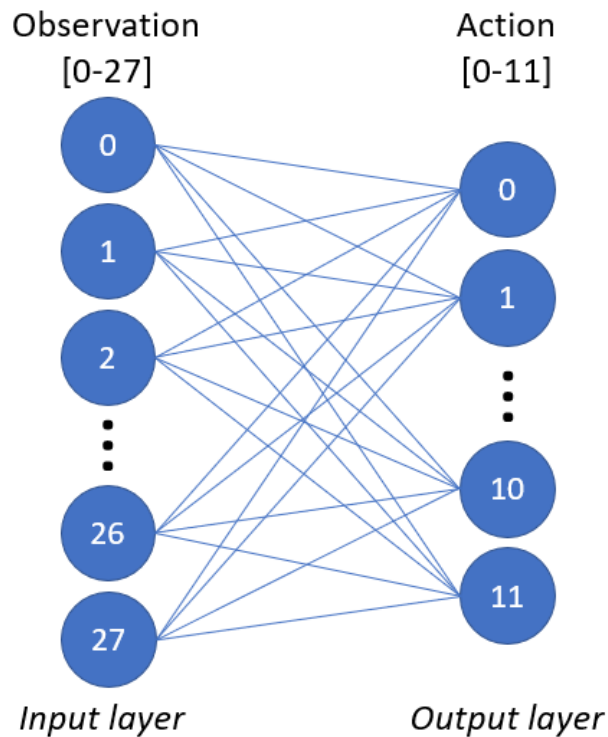


Figure 4.2 - SLP Architecture

Within the observation, the first 16 inputs [0-15] indicate the processing time, in seconds, of each machine, according to the type of part (table 3.1). The last 12 [16-27] correspond to the total changeover to be carried out on all machines, according to the type of product (table 3.3). The state is updated when the production of the 12 pieces is finished, which corresponds to an episode of the simulation. Each output neuron represents a piece and its value corresponds to a number in the interval [-1, 1] obtained by the RL agent. These are ordered in descending order, leading to the production sequence of the next iteration. The next figure 4.3 represents an example.

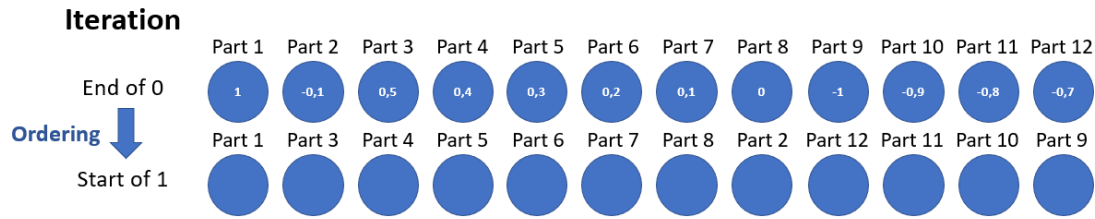


Figure 4.3 - Action Sorting Example

### 4.3 - Makespan Minimization using RL

To verify the precise operation of the Reinforcement Learning algorithm PPO, the initial goal was to minimize the makespan.

The total number of time steps associated with the Agent's learning process was defined as 250,000, which is believed to be sufficient for achieving the optimal solution.

The FlexSim will send the value corresponding to the makespan as a reward to the Server, but with a negative signal (-makespan), once the 12 items have been produced and entered in the sink, or once a step has been completed. This specificity is justified by the fact that the PPO algorithm's objective is always to maximise something. In this instance, the objective is to maximise (-) makespan, or to minimise makespan. The "done" parameter is also sent as a unit in this scenario, as the 12 elements have already been inputted into the Sink and the simulation is complete.

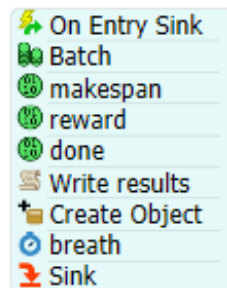


Figure 4.4 - Representation of the block that assigns values to makespan, reward and done

In the following subsections are presented the reward function, a penalty method for the reward and a normalization.

- **Reward Function without Penalty**

$$Reward = - Makespan \quad (4.1)$$

The negative signal is explained in the previous section (4.3).

- **Reward Function with Penalty**

$$Reward = - Makespan - totalPenChangeover \quad (4.2)$$



The penalty “totalPenChangeover” corresponds to the total time used in changeovers by the machines to produce the 12 pieces. This was intended to benefit iterations with shorter changeover times, thus subtraction was used, in order to decrease the reward.

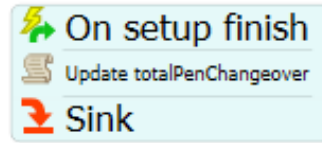


Figure 4.5 - Representation of the block that update the "totalPenChangeover"

- **Normalization**

The primary objective of normalisation is to determine if, by modifying the values of the observations to values between [0, 1], the algorithm will be able to converge on a suitable solution more quickly and improve its learning capacity. The expression used was:

$$Observation_{normalized} = \frac{Observation - Observation_{min}}{Observation_{max} - Observation_{min}} \quad (4.3)$$

- **FlexSim RL Module**

In the next figure it is possible to see the new FlexSim RL module, where all the parameters are inserted, so that the Agent implements what is intended.

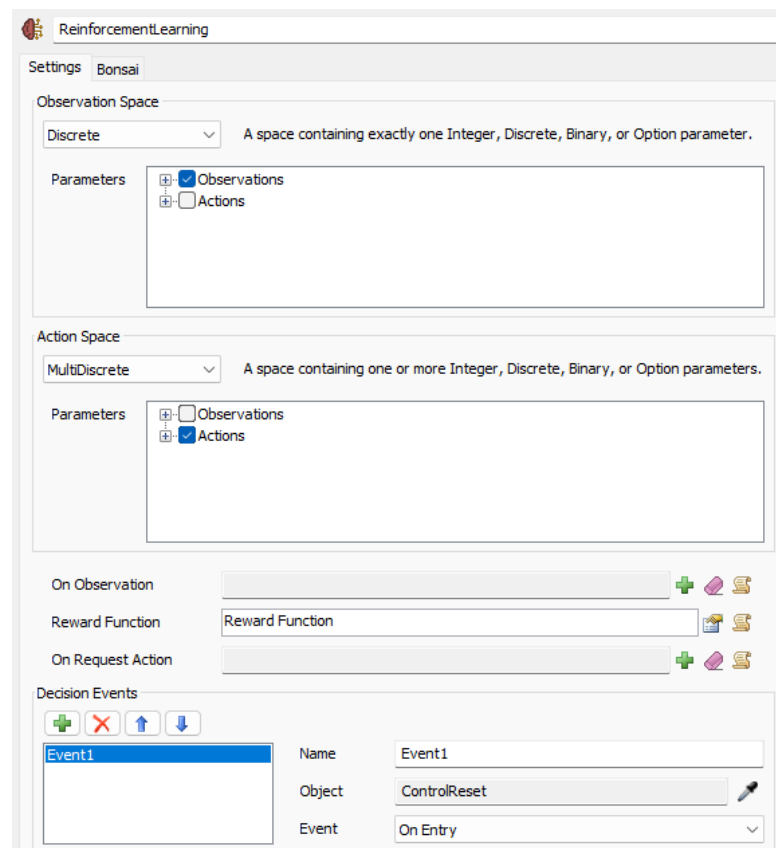


Figure 4.6 - FlexSim RL module

## 4.4 - Makespan Minimization using OptQuest

To address makespan minimization, the OptQuest feature of the FlexSim software was used.

OptQuest is an optimisation feature that integrates tabu search and local search techniques to discover optimal or near-optimal solutions in FlexSim models. This algorithm is designed to explore different combinations of values for the decision variables in the FlexSim model, to find a solution that meets the defined objective, in this case, the minimization of makespan [29].

The decision variable used was a vector called “ProductionSequence” with 12 positions, which corresponds to the sequence of pieces entering the simulation, which are numbered from 0 to 12. OptQuest varies the position of these parts within the vector, obtaining and minimizing the makespan value.

# Chapter 5

## Design of Experiments and Results Analysis

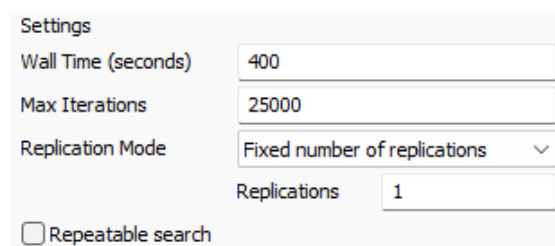
The purpose of this chapter is to analyse the results obtained from the minimisation of the makespan, with OptQuest or Reinforcement Learning techniques.

In the chapter's culminating section, important queries are posed to assess the robustness of each simulation model.

In addition, it should be noted that Python 3.11 and FlexSim (Educational Version 2022) served as the programming and simulation software development environments, respectively, for the dissertation. In this section, all simulations were run on an HP OMEN computer equipped with an Intel Core i7 processor, Quad-core up to 2.20 GHz, and 16 GB RAM.

### 5.1 - Results of makespan minimization using OptQuest

The settings used are represented in the next figure.



| Setting             | Value                        |
|---------------------|------------------------------|
| Wall Time (seconds) | 400                          |
| Max Iterations      | 25000                        |
| Replication Mode    | Fixed number of replications |
| Replications        | 1                            |
| Repeatable search   | <input type="checkbox"/>     |

Figure 5.1 - Settings of OptQuest in FlexSim

As can be seen in the lower left corner of figure 5.2, Makespan = 7013,36 seconds was the best iteration of the makespan optimization by OptQuest. This result will serve as basis for comparison with the optimization scenarios using RL.

OptQuest Success - The optimization stopped when the maximum time was reached.

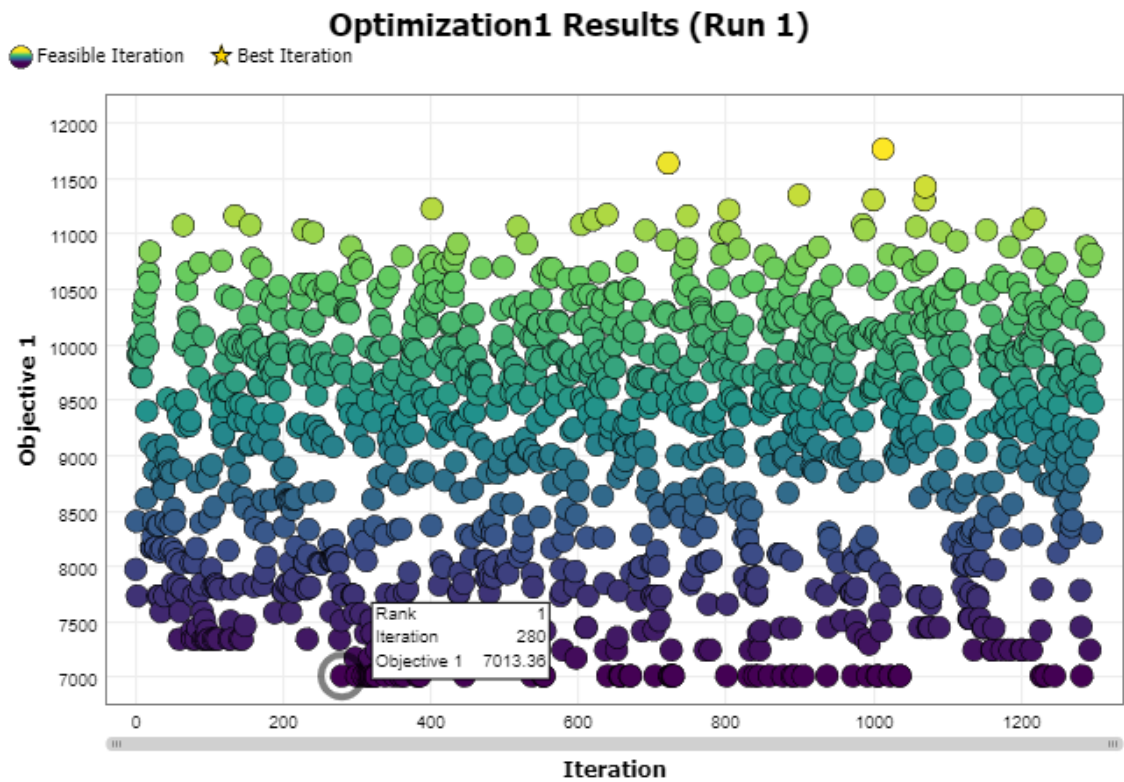


Figure 5.2 - Makespan minimization in FlexSim using OptQuest

## 5.2 - Preliminary Experiments

Some variations of the RL model were also evaluated, such as different hidden layer configurations, with/without penalty, with/without normalization, varying the step size of actions and varying the values and the types of actions, as can be seen in the following table.

Table 5.1 - List of Experiments

| Experiment | Time steps | Hidden layer | Penalty | Normalization | Step size | Action Type | Action space  |
|------------|------------|--------------|---------|---------------|-----------|-------------|---------------|
| 1          | 250k       | 2            | Yes     | Yes           | 0.1       | Discrete    | MultiDiscrete |
| 2          | 100k       | 1            | Yes     | Yes           | 0.1       | Discrete    | MultiDiscrete |
| 3          | 100k       | 2            | No      | Yes           | 0.1       | Discrete    | MultiDiscrete |
| 4          | 100k       | 2            | Yes     | No            | 0.1       | Discrete    | MultiDiscrete |
| 5          | 100k       | 2            | Yes     | Yes           | 0.01      | Discrete    | MultiDiscrete |
| 6          | 100k       | 2            | Yes     | Yes           | 0.1       | Continuous  | Box           |

## 5.2.1 - Multi Layer Perceptron Configuration

In order to choose the RL agent parameters best suited to solve this job shop problem, some tests were performed, varying some hyper parameters, as can be seen in the table 5.2.

Table 5.2 - Table of parameters tests

| Parameters | learning_rate | n_steps | batch_size | n_epochs | policy_kwargs | Reward<br>max | Makespan<br>min |
|------------|---------------|---------|------------|----------|---------------|---------------|-----------------|
| Default    | 0,003         | 2048    | 64         | 10       | none          |               |                 |
| 1          | 0,003         | 2048    | 64         | 1        | []            | -9094,6       | 7627,9          |
| 2          | 0,003         | 2048    | 64         | 1        | [8]           | -9810,8       | 7578,1          |
| 3          | 0,003         | 2048    | 64         | 1        | [16]          | -10087,8      | 7929,2          |
| 4          | 0,003         | 2048    | 64         | 1        | [8, 8]        | -9935,2       | 7647,2          |
| 5          | 0,003         | 2048    | 64         | 1        | [16, 16]      | -9810,8       | 7494,2          |
| 6          | 0,003         | 2048    | 64         | 10       | []            | -9079,9       | 7657,9          |
| 7          | 0,003         | 2048    | 64         | 10       | [8]           | -9824,9       | 7334,9          |
| 8          | 0,003         | 2048    | 64         | 10       | [16]          | -9779,2       | 7719,3          |
| 9          | 0,003         | 2048    | 64         | 10       | [8, 8]        | -9079,9       | 7474,3          |
| 10         | 0,003         | 2048    | 64         | 10       | [16, 16]      | -10461,3      | 7937,4          |
| 11         | 0,003         | 2048    | 64         | 20       | []            | -9803,6       | 7674,0          |
| 12         | 0,003         | 2048    | 64         | 20       | [8]           | -9293,9       | 7308,6          |
| 13         | 0,003         | 2048    | 64         | 20       | [16]          | -9293,9       | 7308,6          |
| 14         | 0,003         | 2048    | 64         | 20       | [8, 8]        | -9125,0       | 7797,1          |
| 15         | 0,003         | 2048    | 64         | 20       | [16, 16]      | -9787,8       | 7443,4          |

Using default hyper parameters, the best results (maximum reward and minimum makespan) are obtained when the "policy\_kwargs" parameter is [8, 8], that is, Multi Layer Perceptron (MLP).

According to the complexity of the job shop problem addressed in this dissertation, MLP has several advantages over SLP, such as[30]:

- MLPs can manage complex nonlinear data relationships, whereas SLPs are limited to basic linear relationships [31];
- High-level feature representations: MLPs can learn more complex and higher-level feature representations, which enables the identification of complex patterns in the data [31];
- Due to their stratified structure, MLPs are more versatile and adaptable to a variety of problems than SLPs, which have a limited modelling capacity [31];

- The Universal Approximation Theorem states that MLPs can approximate any continuous function with high precision, making them suitable for a vast array of modelling applications [31];
- MLPs perform better on tasks involving intricate problems, such as image recognition, natural language processing, and time series prediction [31].

These summarized benefits demonstrate why choosing an MLP over an SLP can be advantageous for this dissertation.

### 5.3 - Results of makespan minimization using RL

As described in Section 4.3, the purpose of the makespan study was to analyse the PPO algorithm's behaviour.

In this subsection, as said in the chapter 5 presentation, the reward was analysed in various experiments.

#### 5.3.1 - Experiment 1 - With 2 hidden layers, with penalty and with normalization

In table 5.2, the experiment with 2 hidden layers was the best parameter with 2048 steps. Here was used 250 000 steps to have a better analyse of the reward.

This experiment is represented by figure 5.3 and table 5.3 and is used as a default scenario. The minimum value of makespan is equal to the one using OptQuest, which shows good signs about the method under study.

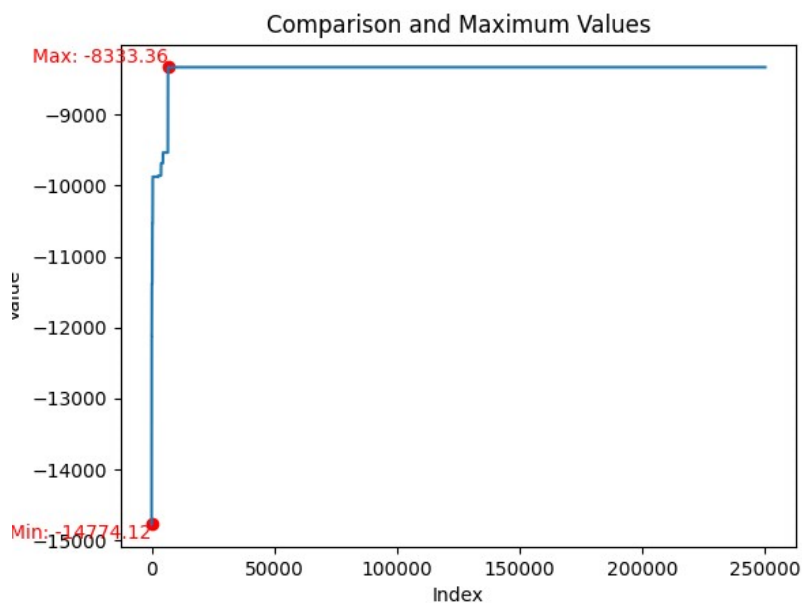


Figure 5.3 - Reward related to Experiment 1, for a total of 250k time steps

Table 5.3 - Numerical analyse of makespan referring to Experiment 1 (in seconds)

| Minimum value | Maximum Value |
|---------------|---------------|
| 7013,359      | 12010,910     |

### 5.3.2 - Experiment 2 - With 1 hidden layer, with penalty and with normalization

According to the graph in figure 5.3, the algorithm reaches the maximum quickly, so it was decided to change the 250k steps to 100k.

Comparing the values obtained in this experiment with the previous one, can be observed that they are relatively equal, so it can be concluded that for a large number of steps, it is the same to have 2 or 1 hidden layers for this problem.

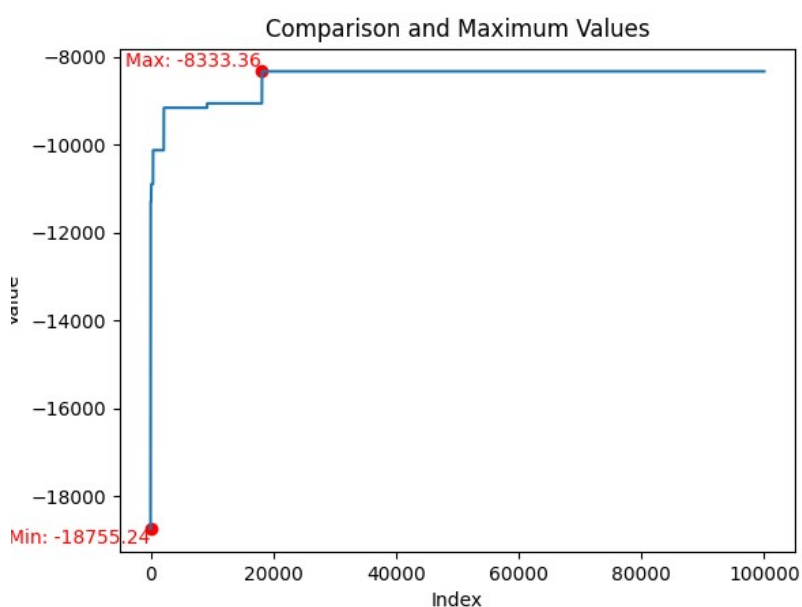


Figure 5.4 - Reward related to Experiment 2, for a total of 100k time steps

Table 5.4 - Numerical analyse of makespan referring to Experiment 2 (in seconds)

| Minimum value | Maximum Value |
|---------------|---------------|
| 7013,359      | 11959,729     |

### 5.3.3 - Experiment 3 - With 2 hidden layer, without penalty and with normalization

As shown in figure 5.5, when the penalty isn't applied, the maximum value of the reward is reached later than the first experiment, and its value is higher, as expected according to the reward function (4.2).

Despite this, the minimum makespan value is higher than in the previous experiments. In conclusion, the introduction of penalty criteria was necessary for the model to progress towards the optimal solution.

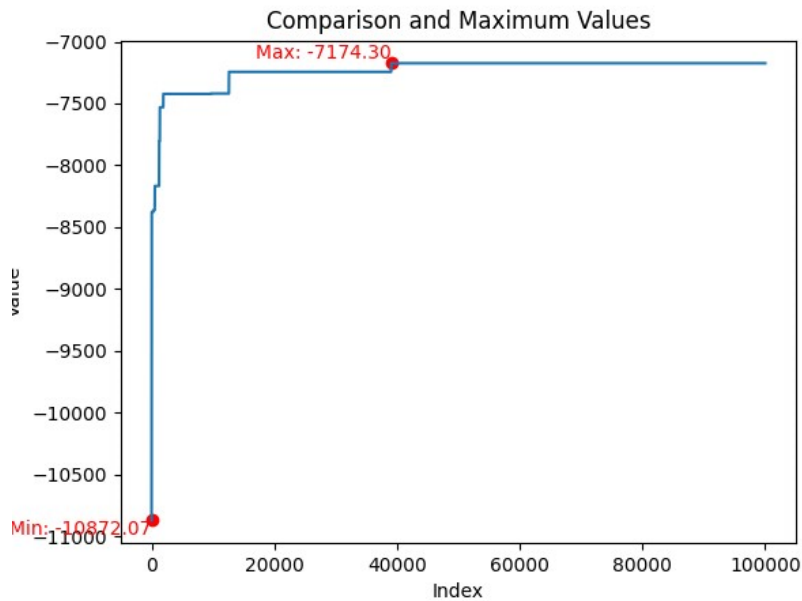


Figure 5.5 - Reward related to Experiment 3, for a total of 100k time steps

Table 5.5 - Numerical analyse of makespan referring to Experiment 3 (in seconds)

| Minimum value | Maximum Value |
|---------------|---------------|
| 7174,298      | 12005,697     |

### 5.3.4 - Experiment 4 - With 2 hidden layer, with penalty and without normalization

As illustrated in figure 5.6 and table 5.6, without normalisation the Agent reaches the best reward later, but the makespan value its the same, comparing with the first experiment. It can be concluded that, when the normalisation is applied, the Agent learns faster.



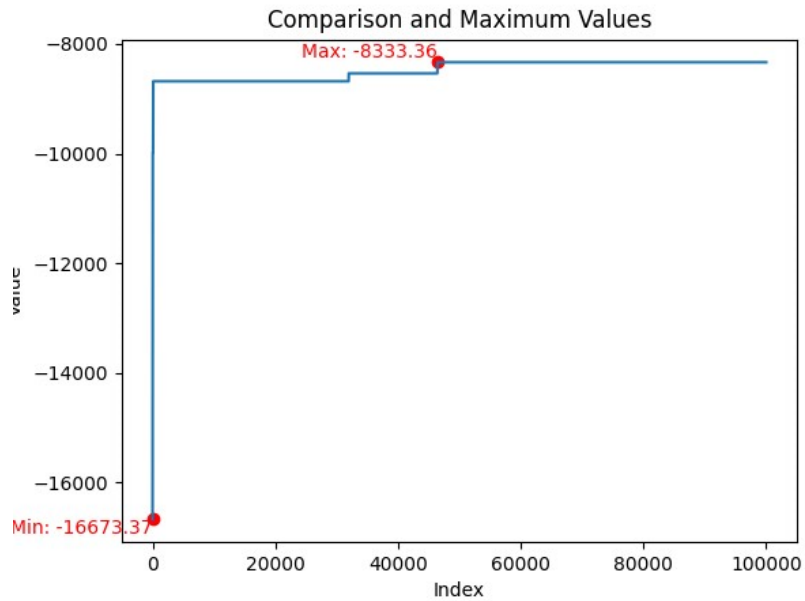


Figure 5.6 - Reward related to Experiment 4, for a total of 100k time steps

Table 5.6 - Numerical analyse of makespan referring to Experiment 4 (in seconds)

| Minimum value | Maximum Value |
|---------------|---------------|
| 7013,359      | 11950,741     |

### 5.3.5 - Experiment 5 - Actions with a different step size

In this experiment, a step size of 0.01 instead of 0.1 was used, in each action.

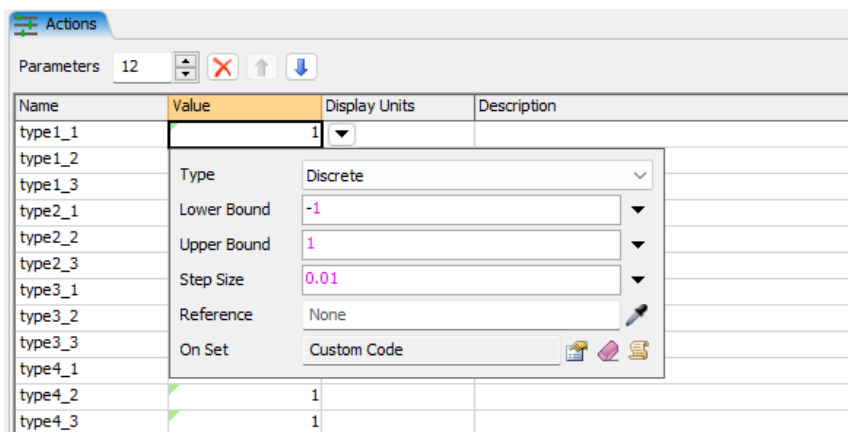


Figure 5.7 - Parameters of an actions in FlexSim

Analysing the figure 5.8 and the table 5.7, can be concluded that with a lower step size in actions, the model reaches later and worst results for the reward and the makespan.

Probably, for this experiment a larger number of time steps would be needed, in order to equal the values of experiment 1.

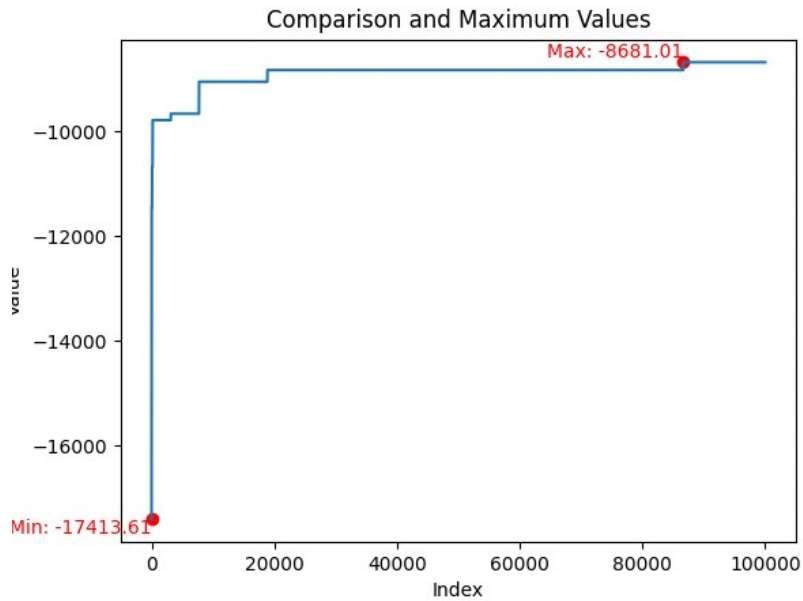


Figure 5.8 - Reward related to Experiment 5, for a total of 100k time steps

Table 5.7 - Numerical analysis of makespan referring to Experiment 5 (in seconds)

| Minimum value | Maximum Value |
|---------------|---------------|
| 7298,640      | 12010,911     |

### 5.3.6 - Experiment 6 - With different action space, and different type of actions

In this experiment, actions with a continuous type and as action space with box type was used, instead of discrete actions and MultiDiscrete action space.

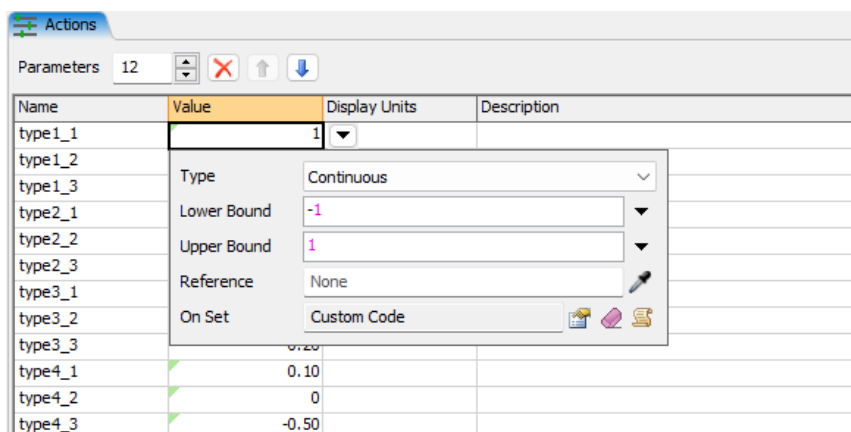


Figure 5.9 - Parameters of an action in FlexSim

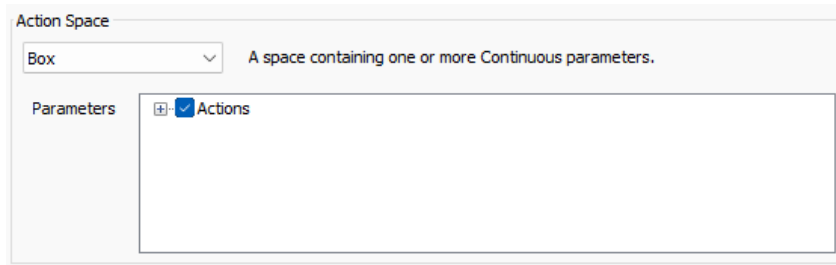


Figure 5.10 - Action Space of Reinforcement Learning in FlexSim

This experiment aims to prove that the parameters chosen for the base experiment (experiment 1) were the correct ones. What can be seen in figure 5.11 and table 5.8, where the agent learns slower and achieves worst results, both for reward and makespan.

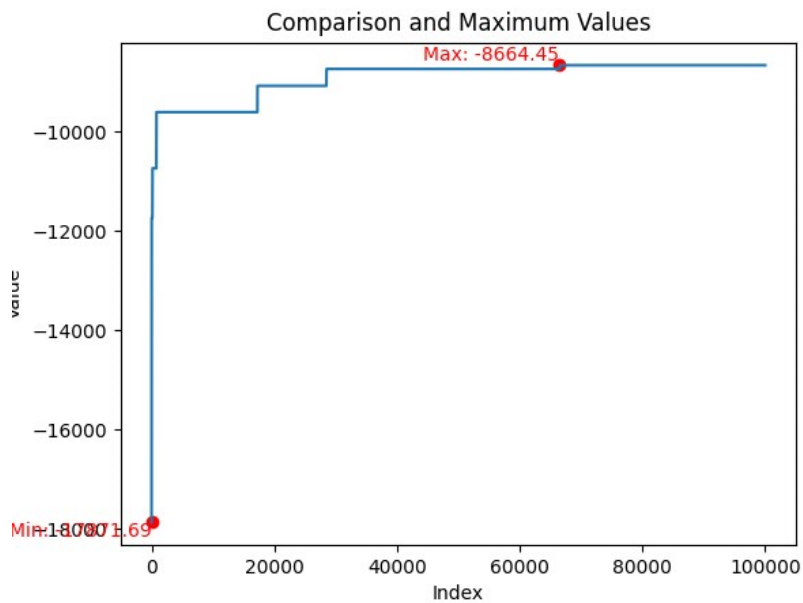


Figure 5.11 - Reward related to Experiment 6, for a total of 100k time steps

Table 1.8 - Numerical analyse of makespan referring to Experiment 6 (in seconds)

| Minimum value | Maximum Value |
|---------------|---------------|
| 7174,298      | 11899,162     |

### 5.3.7 - Results Analysis

In short, it is possible to assert that the Reinforcement Learning PPO algorithm was accurately implemented, given that the models adhere to their characteristic curves, maximising reward and minimising makespan.

The minimum makespan value using OptQuest is equal to the value of some experiments (1, 2 and 4) which reveals that probably the optimal solution of the problem was obtained.

In addition, the results also indicate that the normalisation of rewards and application of penalties enhance the model's capacity for learning.

## 5.4 - Robustness of RL Agent

After analysing the models' performance in deterministic environments, it is necessary to examine their behaviour in other situations. Particularly, it is essential to investigate the effect of the variation of observations in order to assess the adaptability of the models.

As can be seen in figure 5.12, the values of the variation limits of the observations have been changed by 20%. The other parameters of the model are the same as in Experiment 1.

| Name          | Value | Display Units | Description |
|---------------|-------|---------------|-------------|
| MachineA_1_PT | 600   |               |             |
| MachineB_1_PT |       |               |             |
| MachineC_1_PT |       |               |             |
| MachineD_1_PT |       |               |             |
| MachineA_2_PT |       |               |             |
| MachineB_2_PT |       |               |             |
| MachineC_2_PT |       |               |             |
| MachineD_2_PT |       |               |             |
| MachineA_3_PT |       |               |             |
| MachineB_3_PT |       |               |             |
| MachineC_3_PT | 0     |               |             |
| MachineD_3_PT | 300   |               |             |
| MachineA_4_PT | 420   |               |             |
| MachineB_4_PT | 240   |               |             |
| MachineC_4_PT | 420   |               |             |
| MachineD_4_PT | 300   |               |             |
| C1_2          | 180   |               |             |
| C1_3          | 270   |               |             |
| C1_4          | 390   |               |             |
| C2_1          | 180   |               |             |
| C2_3          | 150   |               |             |
| C2_4          | 150   |               |             |
| C3_1          | 270   |               |             |
| C3_2          | 150   |               |             |
| C3_4          | 0     |               |             |
| C4_1          | 390   |               |             |
| C4_2          | 150   |               |             |
| C4_3          | 0     |               |             |

Figure 5.12 - Variation of observations by 20%

As expected, the RL's Agent takes longer to reach the best reward, indicating a lower learning rate. However, the difference between the minimum makespan values is quite small, which represents an excellent indicator regarding the adaptability of the model under study.

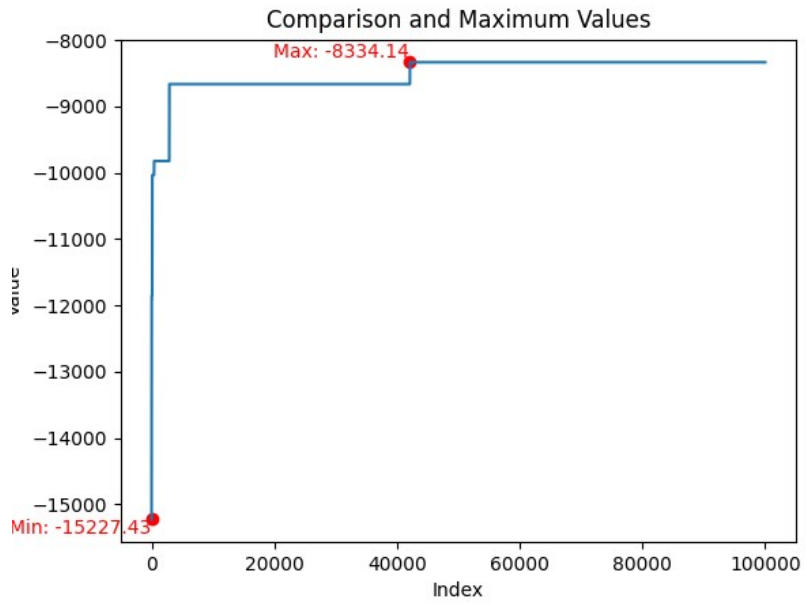


Figure 5.13 - Reward study, for a total of 100k time steps

Table 5.9 - Numerical analyse of makespan

| Minimum value | Maximum Value |
|---------------|---------------|
| 7014,137      | 11959,729     |



# Chapter 6

## Conclusions and Future Work

This chapter presents the conclusions resulting from the dissertation project at hand and allows the reader to determine whether or not all the initial objectives have been met. Lastly, future contributions to this initiative are also proposed.

### 6.1 - Conclusions

The primary objective of the project was to combine Reinforcement Learning techniques with simulation approaches in order to optimise the productivity of a job-shop scheduling problem through the use of RL and simulation. To this end, the metric used was the minimization of the makespan.

At first, the construction of the job shop problem was done in the simulation environment, using the FlexSim software, and, in order to obtain the best possible result, ordering the production sequence of 12 orders and using traditional optimization methods, such as meta-heuristics, a software feature, the OptQuest, was implemented.

In a later phase of the project, Reinforcement Learning techniques were implemented in order to demonstrate that the addition of a third entity, Agent, which is equivalent to a neural network, increases productivity values. As the network learning model, the PPO algorithm from OpenAI Baselines was utilised.

Relatively to this third entity, with regard to productivity, the Agent receives an observation, with all process and changeover times, and a reward. The state is updated when the production of the 12 pieces is finished. Subsequently, it sends the action it considers most suitable and which contains twelve neurons, where each one represents a part with a value in the interval  $[-1, 1]$ , which is ordered in order to build the vector of the production sequence. It should also be noted that the neural network considered has 1 input layer and 1 output layer, composed, respectively, by 28 and 12 neurons. After some tests, it was concluded that the introduction of 2 hidden layers of 8 neurons each, improved the results.

Concerning the RL-based model, it was demonstrated that the normalisation of rewards is a crucial factor in achieving improved results, as network learning would not have been as effective without normalisation. The same happens with the penalty on changeover times.

In order to verify and demonstrate the adaptability and robustness of the productivity model, the original production mix was altered, and the model was subsequently introduced into a stochastic environment in terms of processing and changeover times. It is possible to

assert that the model has a level of robustness that allows it to be applied in the real world due to the encouraging results.

Finally, comparing the results of the model using OptQuest and RL techniques, which were equal, it is possible to conclude that the use of these new RL algorithms in scheduling problems may be very useful in real contexts.

In conclusion, it should be noted that all the initial objectives proposed were fully respected and achieved, and that the project's results and conclusions may serve as an analysis and study tool for future work involving the application of RL techniques in job-shop environments.

## **6.2 - Future Work**

Looking forward, there are many opportunities for further research and development in this area. For example, additional work could be done to investigate the potential for applying these techniques in other manufacturing contexts, or to explore the integration of reinforcement learning with other decision-support tools. These efforts will help to fully realize the potential of intelligent and flexible production systems, and bring the world closer to the industry 4.0 vision of decentralized decision-making and self-organizing systems.

In this context, it would be interesting to develop a simpler neural network, where the step corresponds to the production of only 1 piece. The observation and the action were only one neuron each, corresponding to the type of piece previously produced, and the piece that would be produced next, respectively.

Another possible development, would be to add more parts, as well as more types of parts, so that it would be more difficult to reach an optimal solution, allowing to get a more accurate learning of the RL agent.

Finally, a further development concerning the robustness of the model using RL techniques would be interesting, in order to prove that these algorithms will have a significant importance in the future of scheduling problems.



# References

- [1] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, 'Review of job shop scheduling research and its new perspectives under Industry 4.0', *J Intell Manuf*, vol. 30, no. 4, pp. 1809-1830, Apr. 2019, doi: 10.1007/s10845-017-1350-2.
- [2] M. A. Mediavilla, F. Dietrich, and D. Palm, 'Review and analysis of artificial intelligence methods for demand forecasting in supply chain management', *Procedia CIRP*, vol. 107, pp. 1126-1131, 2022.
- [3] C. Stephanidis *et al.*, 'Integration of Internet of Things Devices in Manufacturing Workspaces: A Systematic Literature Review', in *HCI International 2021 - Late Breaking Papers: HCI Applications in Health, Transport, and Industry*, vol. 13097. Switzerland: Springer International Publishing AG, 2021.
- [4] S. Tian, T. Wang, L. Zhang, and X. Wu, 'The Internet of Things enabled manufacturing enterprise information system design and shop floor dynamic scheduling optimisation', *Enterp Inf Syst*, vol. 14, no. 9-10, pp. 1238-1263, 2020, doi: 10.1080/17517575.2019.1609703.
- [5] T. M. Choi, S. Kumar, X. Yue, and H. L. Chan, 'Disruptive Technologies and Operations Management in the Industry 4.0 Era and Beyond', *Prod Oper Manag*, vol. 31, no. 1, pp. 9-31, Jan. 2022, doi: 10.1111/POMS.13622.
- [6] M. L. Pinedo, 'Scheduling: Theory, Algorithms, and Systems, 4th Edition', 2012.
- [7] Á. Manuel and F. Pereira Da Silva, 'Using Dimensionally Aware Genetic Programming to find interpretable Dispatching Rules for the Job Shop Scheduling Problem'.
- [8] M. Đurasević and D. Jakobović, 'Automatic design of dispatching rules for static scheduling conditions', *Neural Comput Appl*, vol. 33, no. 10, pp. 5043-5068, May 2021, doi: 10.1007/S00521-020-05292-W.
- [9] H. Xiong, S. Shi, D. Ren, and J. Hu, 'A survey of job shop scheduling problem: The types and models', *Computers and Operations Research*, vol. 142. Elsevier Ltd, Jun. 01, 2022. doi: 10.1016/j.cor.2022.105731.
- [10] J. C. Beck, 'Proactive Algorithms for Job Shop Scheduling with Probabilistic Durations', 2007.
- [11] M. R. Garey, D. S. Johnson, and R. Sethi, 'COMPLEXITY OF FLOWSHOP AND JOBSHOP SCHEDULING.', *Mathematics of Operations Research*, vol. 1, no. 2, pp. 117-129, 1976, doi: 10.1287/MOOR.1.2.117.
- [12] D. H. Wolpert and W. G. Macready, 'No free lunch theorems for optimization', *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67-82, 1997, doi: 10.1109/4235.585893.

- [13] Y. Zhou, J. jun Yang, and Z. Huang, 'Automatic design of scheduling policies for dynamic flexible job shop scheduling via surrogate-assisted cooperative co-evolution genetic programming', *Int J Prod Res*, vol. 58, no. 9, pp. 2561-2580, May 2020, doi: 10.1080/00207543.2019.1620362.
- [14] Y. Li, 'Deep Reinforcement Learning: An Overview', Jan. 2017, doi: 10.48550/arxiv.1701.07274.
- [15] M. Kubat, 'An Introduction to Machine Learning', *An Introduction to Machine Learning*, pp. 1-348, Sep. 2017, doi: 10.1007/978-3-319-63913-0.
- [16] F. Alexandre, L. Maia, J. P. Tavares, and V. Basto, 'Hybrid Machine Learning/Simulation Approaches for Logistics Systems Optimization', 2020.
- [17] A. M. Andrew, 'REINFORCEMENT LEARNING: AN INTRODUCTION by Richard S. Sutton and Andrew G. Barto, Adaptive Computation and Machine Learning series, MIT Press (Bradford Book), Cambridge, Mass., 1998, xviii + 322 pp, ISBN 0-262-19398-1, (hardback, £31.95).', *Robotica*, vol. 17, no. 2, pp. 229-235, Mar. 1999, doi: 10.1017/S0263574799211174.
- [18] V. Hugo *et al.*, 'Abordagens híbridas de "machine learning"/simulação para sistemas logísticos dinâmicos', Jul. 2019, Accessed: Jun. 27, 2023. [Online]. Available: <https://repositorio-aberto.up.pt/handle/10216/121197>
- [19] V. Mnih *et al.*, 'Human-level control through deep reinforcement learning', *Nature*, vol. 518, no. 7540, pp. 529-533, Feb. 2015, doi: 10.1038/NATURE14236.
- [20] T. P. Lillicrap *et al.*, 'Continuous control with deep reinforcement learning', *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, Sep. 2015, Accessed: Jun. 21, 2023. [Online]. Available: <https://arxiv.org/abs/1509.02971v6>
- [21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. K. Openai, 'Proximal Policy Optimization Algorithms', Jul. 2017, Accessed: Jun. 21, 2023. [Online]. Available: <https://arxiv.org/abs/1707.06347v2>
- [22] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, 'Trust Region Policy Optimization', *32nd International Conference on Machine Learning, ICML 2015*, vol. 3, pp. 1889-1897, Feb. 2015, Accessed: Jun. 27, 2023. [Online]. Available: <https://arxiv.org/abs/1502.05477v5>
- [23] G. A. Wainer, 'Discrete-event modeling and simulation: A practitioner's approach', *Discrete-Event Modeling and Simulation: A Practitioner's Approach*, pp. 1-494, Jan. 2017, doi: 10.1201/9781420053371/DISCRETE-EVENT-MODELING-SIMULATION-GABRIEL-WAINER.
- [24] J. Banks, J. S. Carson II, and B. L. Nelson, *Discrete-event system simulation*, 2nd ed. Upper Saddle River, New Jersey: Prentice-Hall, 1984.

- [25] M. Munsamy and A. Telukdarie, 'Discrete event modelling for evaluation and optimisation of power utility energy demand', *Journal of Industrial Engineering and Management*, vol. 15, no. 1, pp. 124-141, Feb. 2022, doi: 10.3926/JIEM.3606.
- [26] A. K. da Silva, 'Método para avaliação e seleção de softwares de simulação de eventos discretos aplicados à análise de sistemas logísticos.', Feb. 2007, doi: 10.11606/D.3.2007.TDE-09052007-160956.
- [27] S. Mostafa, N. Chileshe, and T. Abdelhamid, 'Lean and agile integration within offsite construction using discrete event simulation A systematic literature review', *Construction Innovation*, vol. 16, no. 4, pp. 483-525, 2016, doi: 10.1108/CI-09-2014-0043/FULL/XML.
- [28] A. Cruz and P. Cortez, 'Data Mining via Redes neuronais Artificiais e Máquinas de Vectores de Suporte', *Revista de Estudos Politécnicos Polytechnical Studies Review*, vol. 12, pp. 99-118, 2009.
- [29] X. Linwei and Z. X. Li, 'Simulation and optimization of logistics collaborative operation based on flexsim', *Advances in Intelligent and Soft Computing*, vol. 125 AISC, pp. 453-457, 2012, doi: 10.1007/978-3-642-27329-2\_62/COVER.
- [30] C. M. Bishop, *Neural networks for pattern recognition*. Oxford: Oxford University Press, 1995.
- [31] S. S. Haykin, *Neural networks and learning machines*, 3rd, international ed. Upper Saddle River: Pearson, 2009.