



DEGREE PROJECT IN OPTIMIZATION AND SYSTEMS THEORY,  
SECOND CYCLE, 30 CREDITS  
*STOCKHOLM, SWEDEN 2024*

vitesco  
TECHNOLOGIES

# Dynamic Scheduling for Manufacturing Production

**KTH Thesis Report**

Joseph Reboud

## **Authors**

Joseph Reboud <reboud@kth.se> <joseph.reboud@eleves.ec-nantes.fr>  
Degree project in Optimization and Systems Theory, Second Cycle (30 credits)  
Double Degree Programme in Engineering Physics (300 credits)  
KTH Royal Institute of Technology, year 2024

## **Place of the Project**

Toulouse, France  
Vitesco Technologies

## **Examiner**

Anders Forsgren  
Stockholm, Sweden  
KTH Royal Institute of Technology

## **Supervisors**

Anders Forsgren  
Stockholm, Sweden  
KTH Royal Institute of Technology

Le Toan Duong  
Toulouse, France  
Vitesco Technologies

Alexandre Gaffet  
Toulouse, France  
Vitesco Technologies

## Abstract

In the competitive landscape of modern manufacturing, efficient scheduling plays a crucial role in optimizing production processes. While most methods address scheduling problems in static environments, tackling dynamic scheduling in uncertain environments is still an active research area. We address in this thesis the problem of dynamic scheduling in manufacturing production. More specifically, the goal is to design a simulation environment of a production plant under dynamic uncertainties and develop an optimization procedure to come up with a performant scheduling strategy given the complexity of the environment. The results show significant improvement over standard dispatching rules which are extensively used in dynamic contexts for their simplicity and satisfactory performance. In particular, one of the proposed method exhibits good generalization properties and suitable behavior in an industrial context, paving the way to promising further research on challenging dynamic scheduling problems in manufacturing production environments.

---

## **Keywords**

Dynamic Scheduling, Simulation, Dispatching Rule, Deep Reinforcement Learning, Genetic Programming, Manufacturing Production

# Sammanfattning

I det konkurrensutsatta landskapet för modern tillverkning spelar effektiv schemaläggning en avgörande roll för att optimera produktionsprocesser. Medan de flesta metoder tar itu med schemalägningsproblem i statiska miljöer, är det fortfarande ett aktivt forskningsområde att ta itu med dynamisk schemaläggning i osäkra miljöer. I denna avhandling behandlar vi problemet med dynamisk schemaläggning inom tillverkningsproduktion. Mer specifikt är målet att designa en simuleringsmiljö av en produktionsanläggning under dynamiska osäkerheter och utveckla en optimeringsprocedur för att på bästa sätt komma fram till en effektiv schemalägningsstrategi med tanke på miljöns komplexitet. Resultaten visar en betydande förbättring jämfört med standardregler för sändning som används i stor utsträckning i dynamiska sammanhang för sin enkelhet och tillfredsställande prestanda. I synnerhet uppvisar en av de föreslagna metoderna goda generaliseringsegenskaper och lämpligt beteende i ett industriellt sammanhang, vilket banar väg för lovande vidare forskning om komplexa dynamiska schemalägningsproblem i tillverkande produktionsmiljöer.

## Nyckelord

Dynamisk schemaläggning, Simulering, Dispatching rule, Reinforcement Learning, Genetisk programmering, Tillverknings produktion

# Acknowledgements

I would like to thank my supervisors at Vitesco technologies: Le Toan Duong, Alexandre Gaffet and Christophe Merle for their trust, flexibility and support during the project. They provided me with good advice on technical details, potential research trajectories and report review. I would also like to thank my supervisor at KTH, Anders Forsgren for his flexibility and availability during the project.

# Acronyms

<b>JSSP</b>	Job Shop Scheduling Problem
<b>DRL</b>	Deep Reinforcement Learning
<b>PES</b>	Parallel Evolution Strategies
<b>GP</b>	Genetic Programming
<b>ANN</b>	Artificial Neural Network
<b>PDF</b>	Probability Density Function
<b>LST</b>	Line Setup Time
<b>MDP</b>	Markov Decision Process
<b>TWT</b>	Total Weighted Tardiness
<b>CPU</b>	Central Processing Units
<b>EDD</b>	Earliest Due Date
<b>WEDD</b>	Weighted Earliest Due Date
<b>OEE</b>	Overall Equipment Effectiveness

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Company presentation . . . . .	1
1.2	Problem description . . . . .	2
1.2.1	General description . . . . .	2
1.2.2	Assumptions . . . . .	2
1.3	Objectives . . . . .	3
<b>2</b>	<b>Background</b>	<b>4</b>
2.1	Problem classification . . . . .	4
2.1.1	The standard Job Shop Scheduling Problem . . . . .	4
2.1.2	Flexible Job Shop Scheduling Problem . . . . .	5
2.1.3	Dynamic Job Shop Scheduling Problem . . . . .	5
2.2	Solution classification . . . . .	6
2.2.1	Exact approach . . . . .	6
2.2.2	Meta-heuristics . . . . .	6
2.2.3	Dispatching rules . . . . .	7
2.2.4	Simulation-based algorithms . . . . .	9
2.3	Investigated methods . . . . .	9
2.3.1	Genetic Programming . . . . .	10
2.3.2	Parallel Evolution Strategies . . . . .	13
2.4	Summary . . . . .	15
<b>3</b>	<b>Modeling</b>	<b>16</b>
3.1	Simulation formulation . . . . .	16
3.1.1	Constraints . . . . .	16
3.1.2	Plant parameters . . . . .	18
3.1.3	Job types . . . . .	18

3.1.4	Products generation . . . . .	19
3.2	Scheduling model . . . . .	21
3.2.1	Model formulation . . . . .	21
3.2.2	Action definition . . . . .	21
3.2.3	Variables . . . . .	22
3.3	Objective function . . . . .	23
3.4	Training procedure . . . . .	24
3.4.1	Parallel Evolution Strategies . . . . .	24
3.4.2	Genetic Programming . . . . .	25
<b>4</b>	<b>Experiments</b>	<b>26</b>
4.1	Computational complexity . . . . .	26
4.2	Design parameters . . . . .	27
4.2.1	Simulation environment . . . . .	27
4.2.2	Parallel Evolution Strategies . . . . .	28
4.2.3	Genetic Programming . . . . .	29
4.2.4	Validation scores . . . . .	30
4.3	Reference models . . . . .	31
4.3.1	Handcrafted dispatching rule . . . . .	31
4.3.2	Weighted Earliest Due Date . . . . .	32
4.4	Results . . . . .	32
4.4.1	Testing conditions . . . . .	32
4.4.2	Test scenarios . . . . .	33
4.4.3	Analysis . . . . .	40
<b>5</b>	<b>Conclusion</b>	<b>45</b>
5.1	Summary . . . . .	45
5.2	Validity and limitations . . . . .	45
5.3	Further research . . . . .	46
5.3.1	Improve simulation environment . . . . .	46
5.3.2	Improve model training . . . . .	47
5.3.3	Other methods . . . . .	47
	<b>References</b>	<b>48</b>

# Table of notations

<b>Parameter</b>	<b>Definition</b>
$\mathcal{P}^{(j)}$	Process time distribution associated with job $j$
$\mu^{(j)}$	Expected value of $\mathcal{P}^{(j)}$
$\sigma^{(j)}$	Standard deviation of $\mathcal{P}^{(j)}$
$LST^{(j)}$	Line setup time required by job $j$
$s^{(j)}$	Machine setup time associated with job $j$
$c$	Tightness factor
$d$	Due date
$t_{now}$	Current time
$\tau$	Remaining time, defined as $\tau = d - t_{now}$
$\omega_j$	weight of product $j$
$C_j$	Completion time of product $j$
$B_V(t)$	Set of valid batches at time $t$
$L_V(t)$	Set of valid lines at time $t$
$\mathcal{A}_t$	Set of valid actions at time $t$
$a^{(t)}$	A given action's features at time $t$
$s^{(t)}$	State features at time $t$
$N$	Total number of features of interest
$T$	Time after which products stop being generated
$\delta_T$	Time step for generating products
$dt$	Time step for model's decision-making
$\theta$	Vector of parameters of an artificial neural network
$R_\theta$	Artificial neural network representation of a dispatching rule characterized by $\theta$
$R_{GP}$	Syntax tree representation of a dispatching rule
$\mathcal{J}$	Set of possible job types
$\mu_P$	Plant parameters
$\mu_{BG}$	Parameters for batch generation
$F$	Objective function (total weighted tardiness)
$\bar{F}$	Monte Carlo approximation of $F$
$n_{episodes}$	Number of Monte Carlo samples to compute $\bar{F}$
$\alpha$	Learning rate
$\beta$	Weight decay factor
$\sigma$	Standard deviation
$\epsilon$	Isotropic Gaussian noise $\sim \mathcal{N}(0, \sigma I)$
$W$	Number of workers in PES

Table 0.0.1: Table of useful notations

# Chapter 1

## Introduction

With the development of science and technology, manufacturing companies are taking a digital turn to improve their efficiency and flexibility by increasing automation, taking real-time decisions, optimizing processes, and extracting knowledge from large amounts of data [1]. As production plants environments are constantly subject to high uncertainties such as unknown characteristics of products before their arrival, varying process times on machines, machine breakdowns and human error, delivering optimal dynamic scheduling strategies stands out as a major stake for productivity enhancement [2].

### 1.1 Company presentation

Vitesco Technologies, formerly known as Continental Powertrain, is an automotive supplier for drivetrain and powertrain technologies headquartered in Regensburg, Germany since September 2021. The company has about 35,500 employees at around 50 locations worldwide, and is a leading international provider of modern drive technologies and electrification solutions for sustainable mobility. In 2023, the company achieved approximately €9.23 billion sales of which €1.3 billion from electrification components, demonstrating its significant role in the sector. Moreover, Vitesco Technologies is recognized as a preferred supplier in the electromobility sector, with a strong order backlog exceeding €30 billion in the electrification business in 2023. The project was conducted in Toulouse, France in the *Central Operations* department within the AI Team

which focuses on optimizing operational performance by developing innovative artificial intelligence solutions.

## 1.2 Problem description

### 1.2.1 General description

The plant environment considered at Vitesco Technologies is a plant performing surface-mount technology, a method in which electronic components are mounted directly onto the surface of a product called printed circuit board (PCB) illustrated in Figure 1.2.1.

The plant is composed of identical production lines performing a sequence of operations with automated machines and can process any of both sides of a PCB product with its specific process and setup times requirements. The vocabulary *job* will be used to refer to one side of a product and *job type* the set of its requirements. Currently, the management of production is done with the support of an enterprise resource planning (ERP) system which provides daily planning objectives. However, handling dynamic uncertainties for real-time scheduling purposes is challenging and remains an active research area.

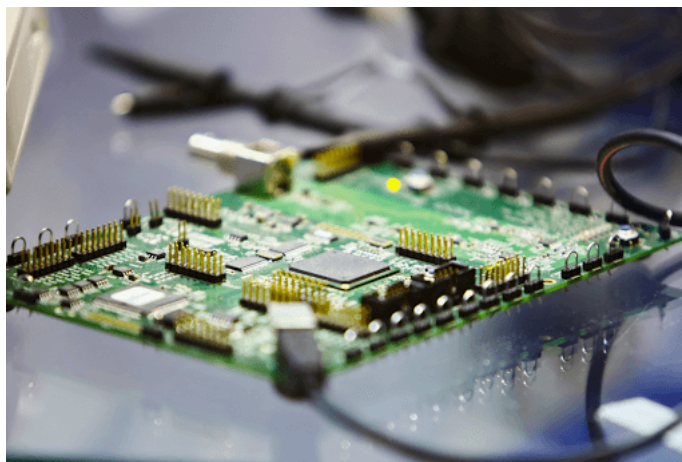


Figure 1.2.1: Example of a printed circuit board (PCB)

### 1.2.2 Assumptions

A general representation is shown in Figure 1.2.2.

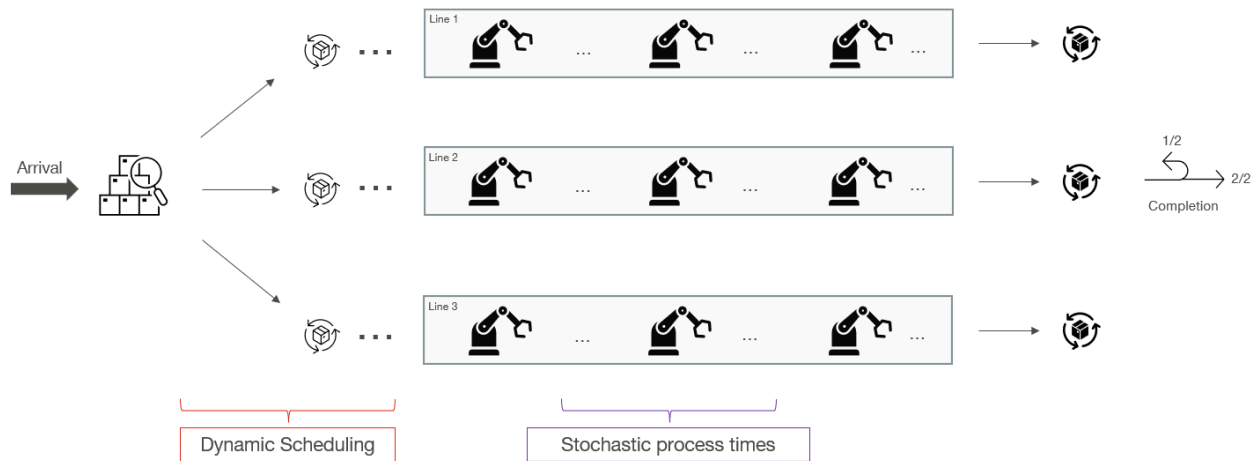


Figure 1.2.2: Overall scheduling process representation

In order to better represent the reality of production environments, non-deterministic processing times on machines are considered, as well as dynamic arrival of products in the system, the latter implying unknown characteristics of products before their arrival in the plant. In addition, no line is specific to any job type, i.e. any line can be re-parameterized to process a different job.

Thus, optimizing the scheduling policy with respect to a given performance metric is a dynamic combinatorial problem composed of two assignments for each product, and where additional uncertainties are considered on processing times and products characteristics.

### 1.3 Objectives

The purpose of the thesis is to pave the way for research on dynamic scheduling for manufacturing production with respect to the specific design and requirements at Vitesco Technologies. More specifically, this project aims to 1) build a relevant simulation environment of the production and 2) analyze, design and develop simulation-based optimization procedures inspired from current state-of-the-art and promising methods from the literature.

# Chapter 2

## Background

This chapter aims at 1) presenting the different classes of the scheduling problem in manufacturing production environments, 2) describing the different methods that have been used in the literature to tackle them and 3) delving into the methods selected to adapt to this project.

### 2.1 Problem classification

#### 2.1.1 The standard Job Shop Scheduling Problem

The Job Shop Scheduling Problem (JSSP) is a classic optimization problem in operations research and scheduling theory. In the JSSP, a set  $\{M_1, \dots, M_m\}$  of  $m$  machines represents the plant environment where each machine is only capable of processing a unique type of operation at a time. A fixed set of jobs  $\{J_1, \dots, J_n\}$  must be processed in the plant, where each job  $j$  has a due date  $d_j$  and requires a specific sequence of operations  $\{O_1^{(j)}, \dots, O_{n_j}^{(j)}\}$ , each with its own processing time and machine requirements. The goal is to find an optimal schedule which optimizes some performance criterion formulated as an objective function  $F$  to minimize. Typically, the total tardiness of completed jobs or the maximum completion time among all tasks —called makespan— are considered relevant objective functions for the problem [3]. A small example of a Job Shop schedule is illustrated in Figure 2.1.1.

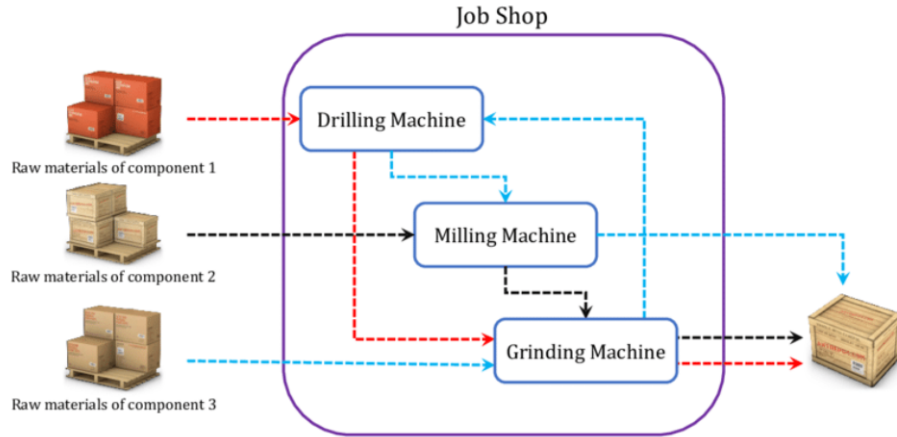


Figure 2.1.1: Example of schedule in a Job Shop with three jobs (components) and three machines. Courtesy of Cornell University, accessed 4 July, this link.

### 2.1.2 Flexible Job Shop Scheduling Problem

The Flexible JSSP is a variant of the JSSP where each operation  $O_i^{(j)}$  of a given job  $j$  can be performed on a subset  $S_i \subset \{M_1, \dots, M_m\}$  of the machines available, rather than a unique one. This design adds more complexity to the problem by enlarging the space of possible solutions.

### 2.1.3 Dynamic Job Shop Scheduling Problem

The applications of the previous problem formulations are strongly limited by the static assumptions. Indeed, the inherent dynamic environment in a production plant with continuous jobs arrival with unknown characteristics before arrival, non-constant process times on machines and machine breakdown likelihood has a high impact on the system's state, hence making optimal scheduling more difficult to perform in realistic applications [4]. The Dynamic JSSP is thus a variant of JSSP which aims at better representing the real world constraints in that regard by adding these unpredictable dynamic events in the system. However, addressing its challenges requires sophisticated scheduling strategies capable of adapting to changing conditions and efficiently allocating resources. Because of significant uncertainties present in the system, the formulation as a written mathematical problem is difficult in practice. Therefore, it is reasonable to model the problem with a simulation environment [5].

## 2.2 Solution classification

### 2.2.1 Exact approach

The JSSP previously presented can be formulated as a Mixed-Integer Linear Programming (MILP) optimization problem with binary constraints to keep track of the order of jobs and not breach physical constraints [6]. However, this combinatorial optimization problem is known to be NP-hard in realistic and non-trivial setups [3] [6], i.e. not solvable with exact optimization approaches in polynomial time. Therefore, the JSSP and its more complex variants must be addressed with more time and computationally efficient methods.

### 2.2.2 Meta-heuristics

Meta-heuristics are algorithms which can effectively approximate solutions to complex optimization problems and have been extensively used to tackle the JSSP and Flexible JSSP [7][8]. This type of algorithms is typically population-based (e.g. genetic algorithm) and works as follows: a population of feasible solutions is initialized as genetic representations [9] and then evolved through evolutionary-inspired operators (selection, crossovers, mutations), producing at each iteration a new generation whose members are feasible solutions and may have better objective values [10]. At the end of the iteration process, the individual with the best objective value encountered so far is given as the solution. Figure 2.2.1 illustrates the overall evolutionary framework (taken from [11]). Although the theoretical convergence of meta-heuristics in general is unknown, it has been established for single objective combinatorial optimization problems with finite search domains [12].

In the context of JSSP and Flexible JSSP, all the characteristics of the system are known in advance: the number of jobs is fixed, they have deterministic process time requirements and there is no dynamic variation in the system. This deterministic design is favorable for meta-heuristics, which can effectively represent feasible solutions as genetic representations, and explore in the space of all feasible solutions via the evolutionary process succinctly described. While this optimization process can take some time depending on the complexity and the size of the problem, meta-heuristics usually provide satisfactory solutions for such deterministic problems [8].

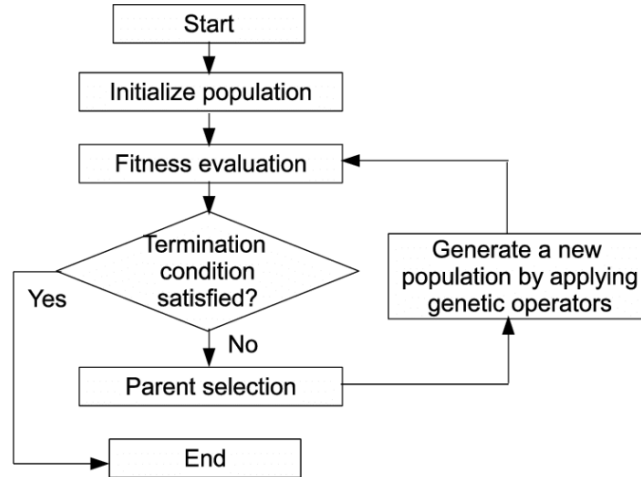


Figure 2.2.1: General evolutionary framework [11]

Some drawbacks of meta-heuristics are the representation of feasible solutions which heavily relies on the problem design, along with the time it requires to run and converge to an acceptable solution. These limitations make meta-heuristics unsuitable in complex dynamical contexts where the state of the system is constantly changing [4].

### 2.2.3 Dispatching rules

#### Definition

A dispatching rule is defined as a function:

$$R : (s^{(t)}, a^{(t)}) \rightarrow R(s^{(t)}, a^{(t)}) \in \mathbb{R}$$

which maps the vector  $s^{(t)}$  of state features and the vector  $a^{(t)}$  of a candidate action features at time  $t$  to a score of this action to be a good choice at this moment [13]. Typically, in the context of production scheduling, a candidate action  $a^{(t)}$  at time  $t$  is represented as a vector of variables of interest characterizing a pair (product, machine) where the product is considered to be assigned to the associated machine [14]. They can be e.g. the product's due date, its process time requirement, the current time  $t$ , the number of remaining operations, the state of the machine's waiting queue, etc. Similarly, the vector of state features  $s^{(t)}$  can represent e.g. statistics on other products' characteristics.

Thus, when a dispatching rule is defined, the scheduling process is done, at each time step,

by computing the scores of all possible actions and by convention perform the one with the highest score. It is straightforward to realize that finding the best dispatching rule to a given scheduling problem is not trivial and requires effort to establish, as presented later in this report. The action selection process of a given dispatching rule is illustrated in Figure 2.2.2.

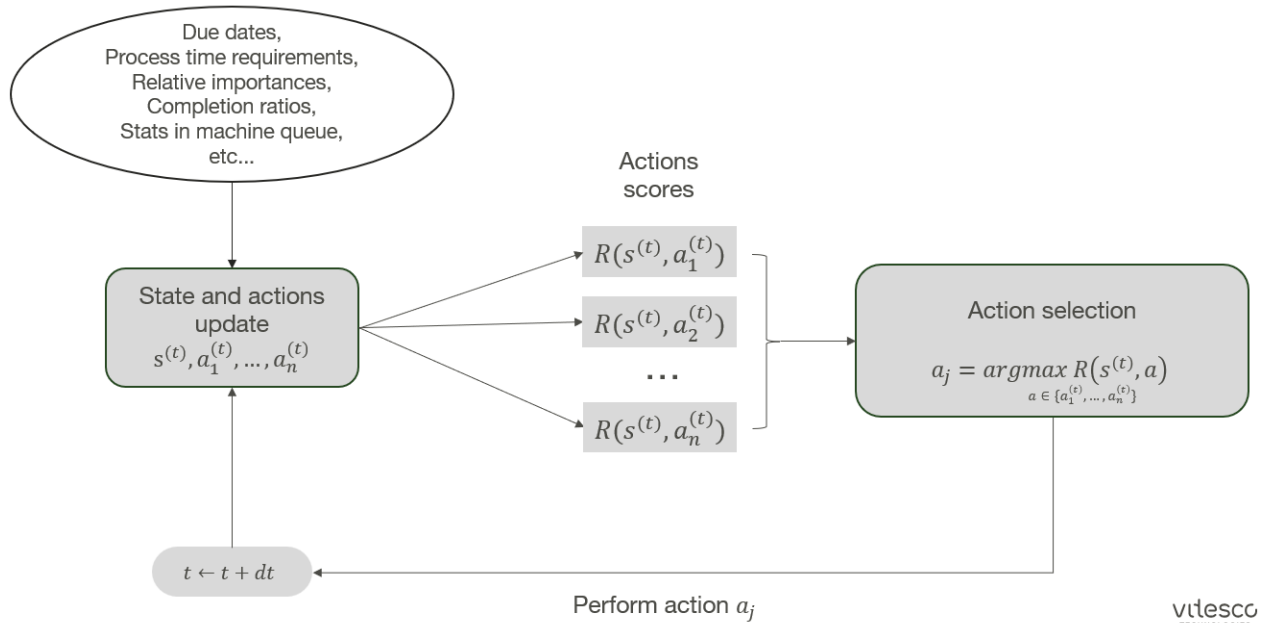


Figure 2.2.2: Action selection process of a given dispatching rule  $R$

### Standard dispatching rules

A simple yet widely used method in industry is to follow predefined handcrafted dispatching rules called standard dispatching rules. They are designed to handle comprehensible priorities between jobs [15]. For example, the First In First Out (FIFO) dispatching rule sends jobs to a machine in the order of their arrival in the machine waiting queue, and the Earliest Due Date (EDD) dispatching rule always prioritizes the most urgent job in machine queues. More precisely, with the previously introduced notations:  $R_{EDD}(s^{(t)}, a_j^{(t)}) = -d_j$  where  $d_j$  is the due date of the job  $j$  considered to be sent. These rules are easy to implement, to understand and are suitable in a dynamic environment [13]. However, it is straightforward to see that these rules are not optimal since they disregard other features that might be important in the scheduling process (e.g. relative importance, process time distributions, number of remaining operations, etc.).

### 2.2.4 Simulation-based algorithms

Motivated by the difficulty to mathematically formulate complex environments with several sources of uncertainties, effort has been made to build end-to-end scheduling agents to directly provide a scheduling decision based on the state of the system at any time step, therefore requiring a simulation environment of the system [5]. The main idea is to use the simulation environment to handle all the constraints and dynamic events occurrence, and use it either to design a reward function on the actions which helps improve the scheduling agents actions via reinforcement learning techniques [16][4], or as a 'black box' function which evaluates the overall performance of any dispatching rule and can therefore be used to establish an optimization procedure over the set of dispatching rules [4][14][17].

To the best of our knowledge, two such methods have been used in the Dynamic JSSP context: Genetic Programming (GP) and Deep Reinforcement Learning (DRL). Both methods aim at finding the best dispatching rule by interacting in the simulation environment. More specifically, GP represents dispatching rules  $R_{GP}(s, a)$  as combinations of basic mathematical operations ( $+$ ,  $-$ ,  $*$ ,  $/$ ,  $min$ ,  $max$ ) of the state and actions features, and uses a population-based evolutionary method to evolve these rules and improve their fitness [14]. When it comes to DRL, an artificial neural network is used as a representation of  $R_{DRL}(s, a)$  and its parameters are optimized to find the best dispatching rule possible [4].

Genetic programming has become popular in the Dynamic JSSP context [17], and reinforcement learning has recently shown comparable even better performance [4]. In addition, some recent work has been done on the Dynamic Flexible JSSP using other deep learning models to select the most appropriate standard dispatching rule at each decision step [18]. Yet, dynamic scheduling remains a challenging and active area of research.

## 2.3 Investigated methods

Since the major goal is to tackle a dynamic scheduling problem similar to the Dynamic JSSP in an industrial context with several sources of uncertainties, our method will focus on simulation-based algorithms. In particular, Genetic Programming (GP) and DRL via Parallel Evolution Strategies (PES) will be assessed. Both methods design the scheduling model as a Markov Decision Process (MDP), i.e. a discrete-time stochastic control process, which at each decision step takes input information from the state of the system and valid actions and then

performs an action in the simulation—like a dispatching rule. This design enables handling varying number of possible actions during the process. These methods are described in more details in the following sub sections.

### 2.3.1 Genetic Programming

In the context of Dynamic JSSP, the goal of GP is to come up with an 'optimal' dispatching rule (see 2.2.3) whose overall performance is evaluated in the simulation via a predefined objective function  $F$  called *fitness function*. The most common representation of these rules is via a syntax tree, where internal nodes are mathematical operations and terminal nodes are variables of interest [19]. Such a tree is illustrated in Figure 2.3.1, where the terminal nodes are highlighted in green and internal nodes in red.

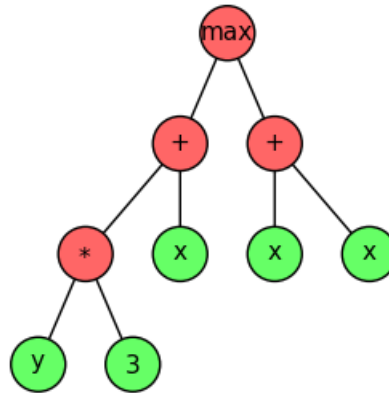


Figure 2.3.1: Genetic Programming representation of  $R(x, y) = \max(3*y + x, x + x)$ . Courtesy of DEAP documentation, accessed 5 July, this link.

The syntax tree structure makes it natural to apply evolutionary operators (e.g. mutation, crossover) to modify them and come up with new dispatching rules which remain similar to the parent function. Examples of applying the crossover and mutation operators are illustrated in Figure 2.3.2 and Figure 2.3.3.

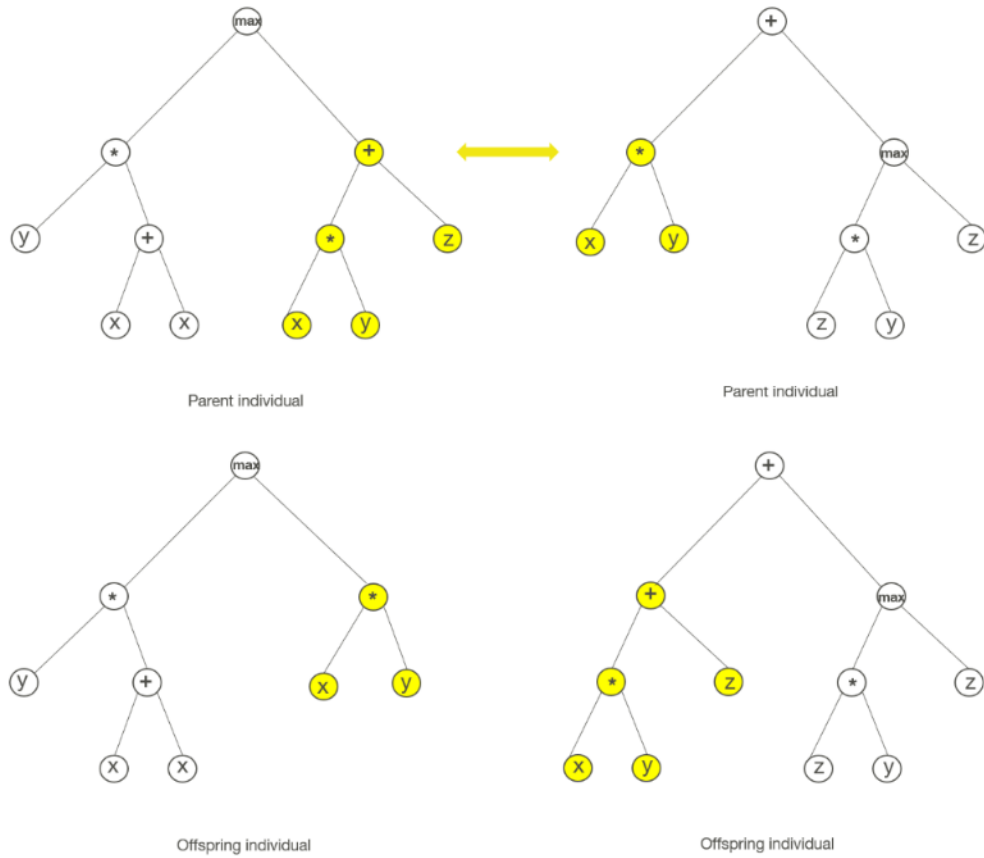


Figure 2.3.2: Crossover operation between two syntax tree representations

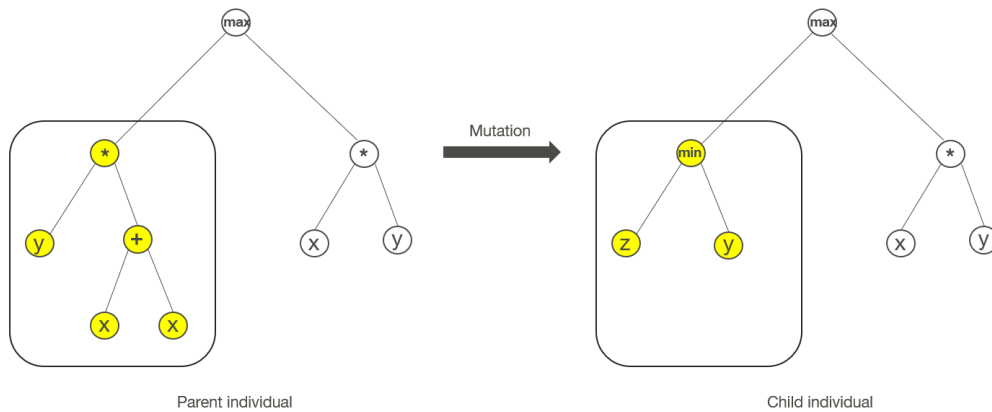


Figure 2.3.3: Mutation operation on a syntax tree representation

The idea of GP is to start randomly with a population of dispatching rules (cf 2.2.3) represented as syntax trees and then iteratively: evaluate their fitness via  $F$ , select the best individuals, perform crossover operators between them with some probability  $p_{crossover}$ ,

mutate some of them with probability  $p_{mut}$  and update the population. Doing so introduces new individuals combining branches from the best parent individuals while undergoing variability due to mutations. At the end of the iterative process, the best individual encountered so far is selected and provided as the best solution dispatching rule to the optimization problem at hand. We recall the general evolutionary framework similar to meta-heuristics in Figure 2.2.1. If the initial population size is important enough, all evolutionary operators together ensure a wide exploration in the solution space while maintaining objective value improvements at each iteration.

Although the search procedures of Genetic Programming and a meta-heuristic like Genetic Algorithm (GA) both apply genetic operators, the main difference in tackling the scheduling problem resides in the solutions representation: GA represents the global scheduling strategy -which heavily relies on the problem design- whereas GP represents a dispatching rule to apply. This difference makes GP not only much more suited to complex dynamic environments, but also computationally more efficient since it only requires an 'offline' training procedure and can then be used quickly 'online' during inference [17].

## 2.3.2 Parallel Evolution Strategies

### Preliminary: Artificial Neural Network

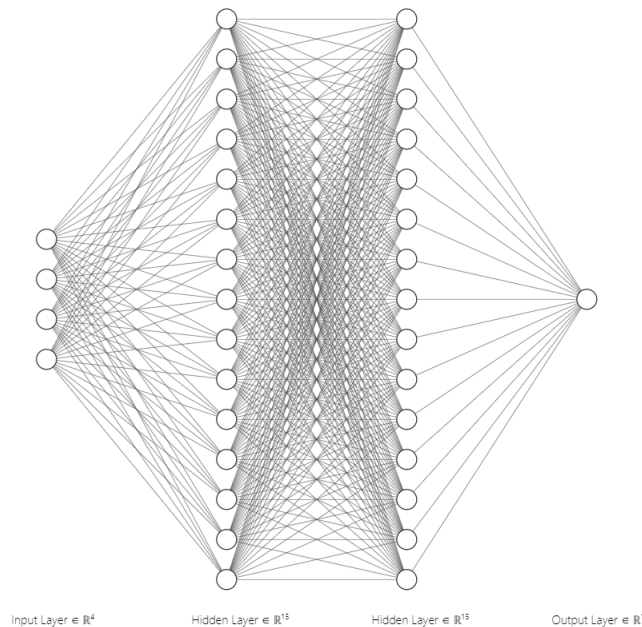


Figure 2.3.4: Dense artificial neural network representation. Courtesy of NN-SVG site, accessed 5 July, [this link](#).

Keeping the same idea of representing dispatching rules, Artificial Neural Network (ANN) stands for an interesting alternative to GP. Indeed, by computing highly parameterized functions, an ANN is a simple yet powerful way to approximate arbitrary complex functions provided that both the architecture and the network parameters are tuned for the desired behavior [20]. Figure 2.3.4 illustrates an example of a basic dense ANN. Each edge is a connection between two neurons and is associated a scalar parameter called a weight.

When inputs  $x_1, \dots, x_n \in \mathbb{R}$  are given to the network, each neuron  $j$  of the second layer multiplies the inputs by their associated weight  $w_1^{(j)}, \dots, w_n^{(j)}$ , sums them up and adds an additional parameter  $b^{(j)}$  called bias. Then this sum is passed through a predefined nonlinear function  $f$  called activation function, which produces the output of the neuron in the second layer, illustrated in Figure 2.3.5. This process is repeated for all neurons in the forward direction until the final layer is reached and produces the final outputs. Thus, an artificial neural network is a nonlinear and highly parameterized function representation which is easy to implement as a computer program. We denote  $\theta$  the vector of all the ANN's parameters (i.e. weights and biases) which can be subject to optimization.

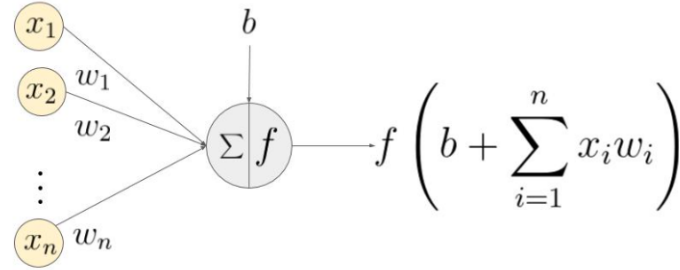


Figure 2.3.5: Computation of a single neuron output. Courtesy of LearnOpenCV site, accessed 5 July, [this link](#).

### Optimization procedure

Deep Reinforcement Learning (DRL) can be defined as a type of machine learning that combines ANN with reinforcement learning principles to enable agents to make decisions and learn from their actions. Such agents are trained to maximize rewards through trial and error, usually in a simulation environment, and often resulting in behavior or policies that perform well in dynamic and challenging environments [21]. However, one of the major challenges for a DRL strategy to tackle the Dynamic Flexible JSSP is the difficulty to design the reward function, since an action might have delayed and intractable consequences in the system [4]. A way of training an ANN policy via Parallel Evolution Strategies (PES) is described here: a method which remains effective with very sparse reward schemes and demonstrated state-of-the-art performance on reinforcement learning benchmarks problems [22]. This part is dedicated to showing the optimization algorithm procedure and the intuition on how it works. The technical details and mathematical justifications are given in [22].

Let  $\theta$  denote the vector of parameters to optimize over, and  $F$  the objective function to maximize which is only accessible via computing  $F(\theta)$ . Since no guarantees hold on  $F$ , it could be non-smooth hence regular gradient approximation methods are not suited for this type of problem. Rather than reasoning on  $\theta$  itself, the idea of PES is to find a distribution  $\mathcal{D}$  which maximizes  $\mathbb{E}_{\theta \sim \mathcal{D}}[F(\theta)]$ . Typically,  $\mathcal{D}$  is represented as a multivariate isotropic Gaussian distribution  $\mathcal{D}_\mu = \mathcal{N}(\mu, \sigma I)$  fully characterized by its mean  $\mu$  (which becomes the new optimization variable) and where  $\sigma$  is a fixed hyper-parameter. By adding noise in the space of parameters via  $\mathcal{D}$ , the method smooths the problem and keeps it relevant with regard to the initial objective [22]. Moreover, the choice of the Gaussian distribution is mathematically

justified, since it simplifies the expression of the gradient of the new objective function at hand:

$$\nabla_{\mu} \mathbb{E}_{\theta \sim \mathcal{D}_{\mu}} [F(\theta)] = \frac{1}{\sigma} \mathbb{E}_{\epsilon \sim \mathcal{N}(0, I)} [F(\mu + \sigma \epsilon) \epsilon] \quad (2.1)$$

which can be approximated with samples. From this, a basic gradient ascent algorithm can be applied to optimize over  $\mu$ , which is assimilated as  $\theta$  in Algorithm 2.3.6 [22].

---

**Algorithm 1** Evolution Strategies
 

---

```

1: Input: Learning rate  $\alpha$ , noise standard deviation  $\sigma$ , initial policy parameters  $\theta_0$ 
2: for  $t = 0, 1, 2, \dots$  do
3:   Sample  $\epsilon_1, \dots, \epsilon_n \sim \mathcal{N}(0, I)$ 
4:   Compute returns  $F_i = F(\theta_t + \sigma \epsilon_i)$  for  $i = 1, \dots, n$ 
5:   Set  $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^n F_i \epsilon_i$ 
6: end for

```

---

Figure 2.3.6: Evolution Strategies algorithm. Courtesy of OpenAI, accessed 5 July

Since the evaluations of the elements in  $\{F_i\}_{i=1, \dots, n}$  at each iteration are independent from each other, they can be computed in parallel on different workers to boost computational and time efficiency [22].

While this method can significantly improve the chances to find good solutions, it does not guarantee global optimization of non-convex problems. However, it can enhance the ability to find better local optima. In the context of our work,  $\theta$  will be the vector of parameters of an ANN representing a dispatching rule, and  $F(\theta)$  will characterize the overall performance of this rule in the simulation.

## 2.4 Summary

Suitability	Exact approaches	Meta-heuristics	Standard dispatching rules	GP - PES
JSSP	Yes/No	Yes	No	Superfluous
Flexible JSSP	No	Yes	No	Yes
Dynamic JSSP	No	No	Yes	Yes

Table 2.4.1: Summary of suitability of each method for each problem.

# Chapter 3

## Modeling

We implemented a simulation environment summarized as a Python class:

$$\text{SimEnv}(R, \mu_P, \mathcal{J}, \mu_{\text{BG}})$$

which takes in input a dispatching rule  $R$ , the plant parameters  $\mu_P$ , the set of possible job types  $\mathcal{J}$  and the parameters  $\mu_{\text{BG}}$  handling the generation of products. The class handles all the dynamic interactions between machines, lines and products—scheduled by  $R$ —with respect to the given parameters that are described in this section. It can therefore be run to assess the behavior of the dispatching rule  $R$ , evaluate its performance quantitatively, and also provide log data which can later be used for visualization purposes if needed.

`SimEnv` was implemented as a subclass of the discrete-event simulation library *Salabim* in Python [23]. We do not go into the technical details of its implementation since it does not require relevant mathematical background.

### 3.1 Simulation formulation

#### 3.1.1 Constraints

The purpose of the simulation environment is to handle necessary constraints whether they are general (physics, time-related, stochastic behaviors) or specific to the company (capacities of machines, products characteristics, etc.). These constraints are specified in the following:

1. **Products:** **a)** Any product is composed of two jobs, each requiring an entire line for processing. The second job can be processed only if the first one has been completed, and the product is completed when both its jobs are completed. **b)** New batches of several products arrive in the system at a constant time interval and each batch contains a unique product type. The characteristics of a given product type are: the weight  $\omega$  representing its relative importance to other product types, its due date  $d$ , and the process time distributions  $\mathcal{P}^{(j_1)}$ ,  $\mathcal{P}^{(j_2)}$ , machine setup times  $s^{(j_1)}$ ,  $s^{(j_2)}$  and line setup times  $LST^{(j_1)}$ ,  $LST^{(j_2)}$  required by both its jobs  $j_1$  and  $j_2$ .
2. **Time related:** **a)** A machine processes a job  $j$  with a processing time sampled from the distribution  $\mathcal{P}^{(j)}$  and then sets up during a time  $s^{(j)}$  before processing the next jobs. **b)** If a job  $j$  is assigned to a line which is not currently processing the same job type, then there is a line setup time  $LST^{(j)}$  to wait before starting processing  $j$ .
3. **Physical constraints:** **a)** All production lines have the same order of operations performed by machines with constant capacities. **b)** Once assigned to a line, a job always follows the order of operations of this line—from first to last machine. **c)** When a machine is busy, i.e. processing or setting up, the jobs coming from the previous machine wait and accumulate in a buffer of infinite size. **d)** Any machine of a line except the first one processes jobs in the order of their arrival in its associated buffer.

Some specific assumptions are made here. In particular:

1. Line Setup Time (LST) represents the time it takes to re-parameterize a line for a different job type. For simplicity, it is modeled as a delay before starting processing the new job.
2. The buffers between machines have infinite size for the purpose of scheduling analysis, i.e. assess whether optimal scheduling strategies stem accumulation between machines by themselves or not.
3. For a given job  $j$ , the same process time distribution  $\mathcal{P}^{(j)}$  and setup time  $s^{(j)}$  are used on all machines that encounter  $j$ . The idea is to keep a constant flow within lines while still introducing variability on process times.
4. New products arrive in the system at a constant time rate to stand for the daily objectives provided by the planning software. That is, the software regularly provides what

new product types must be processed, and the dispatching rule is in charge of their scheduling.

5. The mean and variance of  $\mathcal{P}^{(j)}$  are considered to be known for any job type since they can be approximated with the company's data.

### 3.1.2 Plant parameters

The physical constraints previously mentioned are handled in the simulation environment as long as the plant parameters  $\mu_P = (n_L, n_M, L_c)$  are given, which consist of the number  $n_L$  of lines in the plant and the number  $n_M$  of machines in each line, along with the list of their associated capacities  $L_c = \{c_1, \dots, c_M\}$ . These parameters are fixed for a given plant design.

### 3.1.3 Job types

As previously described in part 3.1.1, a job  $j$  is characterized by the processing time distribution  $\mathcal{P}^{(j)}$ , machines setup times  $s^{(j)}$ , and line setup time  $LST^{(j)}$  it requires in the system. The tuple  $(\mathcal{P}^{(j)}, s^{(j)}, LST^{(j)})$  will be referred to as the *job type*.

#### Process time distributions

While  $s^{(j)}$  and  $LST^{(j)}$  are considered constant for a given job type, we model  $\mathcal{P}^{(j)}$  as a random variable for the sake of realistic modeling. More precisely, we design normal right-skewed distributions approximately fitting the data from the company's plant. The idea is that the regular processing time of an operation on a given job has a lower bound which is usually not reached due to unpredictable events (voluntary interruption from a human operator, machine breakdown, etc.) which entail significant variability towards higher values. Such a Probability Density Function (PDF) is characterized by three integer parameters—*a*, *loc*, and *scale*—respectively responsible for the left slope, the localisation on the x-axis, and the variance. Several examples are illustrated in Figure 3.1.1.

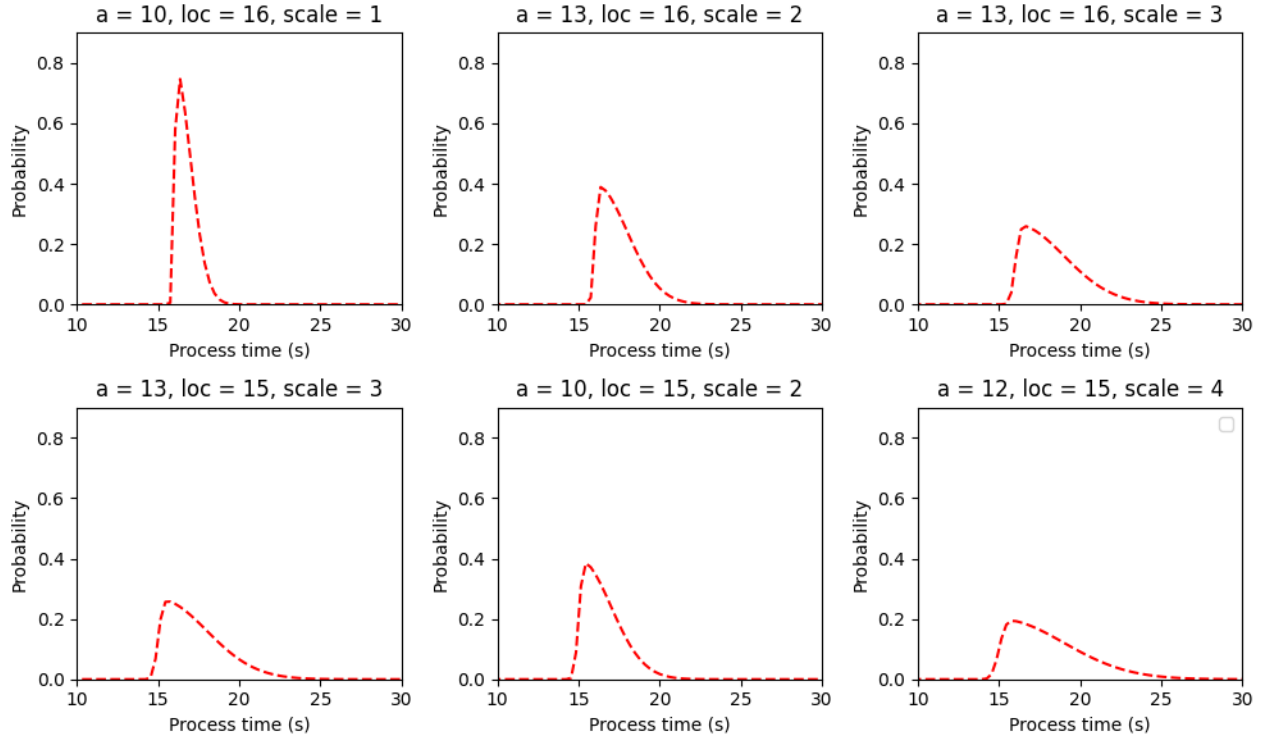


Figure 3.1.1: Examples of generated process time PDFs

### Set of possible job types $\mathcal{J}$

We introduce  $\mathcal{J}$  the set of  $N_{\mathcal{J}}$  possible job types to introduce in the system:

$$\mathcal{J} = \{(\mathcal{P}^{(j)}, s^{(j)}, LST^{(j)})\}_{j=1, \dots, N_{\mathcal{J}}} \quad (3.1)$$

which is created by using different relevant sets of parameters ( $a$ ,  $loc$ , and  $scale$ ) for the process time distributions  $\{\mathcal{P}^{(j)}\}_{j=1, \dots, N_{\mathcal{J}}}$  as well as relevant range of values for the setup times  $\{s^{(j)}\}_{j=1, \dots, N_{\mathcal{J}}}$  and line setup times  $\{LST^{(j)}\}_{j=1, \dots, N_{\mathcal{J}}}$ . This set will be useful when generating products as described next.

### 3.1.4 Products generation

#### Due dates

We choose to control the generation of due dates with a single parameter  $c > 0$  called tightness factor. Any product composed of two jobs  $j_1, j_2$  arriving in the system is assigned a due date

generated as follows:

$$d = t_{arrival} + c \sum_{j \in \{j_1, j_2\}} [n_M \mu^{(j)} + LST^{(j)}] \quad (3.2)$$

where  $t_{arrival}$  is the arrival time,  $\mu^{(j)} = \mathbb{E}[\mathcal{P}^{(j)}]$  is the expected value of the process time distribution associated with job  $j$ ,  $LST^{(j)}$  is the line setup time associated with job  $j$ ,  $n_M$  is the number of machines in a line and  $c$  is the tightness factor. The quantity  $Q = \sum_{j \in \{j_1, j_2\}} [n_M \mu^{(j)} + LST^{(j)}]$  approximates how much time a product is expected to spend in the system if it never waits to be scheduled nor accumulates in any buffer. Therefore,  $c$  characterizes how urgent new products arrive in the system: if  $c = 1$ , then all new products have no slack time hence accumulate delay whenever they wait to be scheduled. By contrast, a larger tightness factor e.g.  $c = 3$  entails larger slack times and more flexibility during scheduling.

### Batch generation

In a manufacturing production context, products are usually grouped in batches for deliveries, stocks and even sometimes for processing. Hence we decide to model batches in the simulation from which products can be taken and scheduled independently. We therefore define a batch as a set of products sharing identical characteristics: i.e. same due date, same weight and same job types. A batch is generated by randomly selecting two job types in  $\mathcal{J}$ , randomly assigning the same weight to all its products as well as the same due date following the previous description.

We control the generation of batches with the parameters  $\mu_{BG} = (T, \delta_T, c, n_b, L_{sizes}, L_w)$  where  $T$  is the time after which we stop generating products,  $\delta_T$  is the time step between each generation,  $c$  is the tightness factor,  $n_b$  is the number of different batches generated at each time step,  $L_{sizes}$  is the list of possible batch sizes and  $L_w$  is the list of possible weights to assign.

## 3.2 Scheduling model

### 3.2.1 Model formulation

The model is a dispatching rule representation:

$$R = \begin{cases} R_\theta & \text{in the PES case} \\ R_{GP} & \text{in the GP case} \end{cases} \quad (3.3)$$

where  $R_\theta$  is a dense ANN parameterized by  $\theta$  (cf 2.3.2) and  $R_{GP}$  is a syntax tree (cf 2.3.1). It is defined in either case as:

$$R : \begin{cases} \mathbb{R}^N & \rightarrow \mathbb{R} \\ x = (s, a) & \rightarrow R(x) \end{cases} \quad (3.4)$$

where  $x = (s, a)$  is the concatenation of the  $N$  state and action features defined in part 3.2.3, and  $R(x) = R(s, a)$  is the score of action  $a$  to be a good action in the state context characterized by  $s$ .

### 3.2.2 Action definition

In order to clearly define the action a model can choose during scheduling, we introduce the notion of *valid line* and *valid batch*. A *valid line* is a line which first machine is either idle or in processing with no subsequent products waiting for it, and a *valid batch* is a non-empty batch of products with identical characteristics that all require the same job type to process. We denote  $L_V(t)$  and  $B_V(t)$  respectively the sets of valid lines at time  $t$  and valid batches at time  $t$ . Therefore, we define the set of available actions  $\mathcal{A}_t$  as the set of valid pairs at time  $t$ :

$$\mathcal{A}_t = B_V(t) \times L_V(t) = \{(b, l)\}_{b \in B_V(t), l \in L_V(t)} \quad (3.5)$$

When an action  $a^{(t)} = (b, l) \in \mathcal{A}_t$  is chosen, a fixed amount of products is sent from  $b$  to  $l$ . When it occurs, the jobs sent wait for the first machine of  $l$  to be idle if it is currently busy, otherwise the machine starts processing an amount of these jobs corresponding to its capacity and repeats this process until all waiting jobs have been processed. Meanwhile, the machine is considered busy. If no action is available at time  $t$  (i.e.  $\mathcal{A}_t = \emptyset$ ) then the model will simply wait until an action is available.

### 3.2.3 Variables

The action definition previously introduced is useful for an action to actually occur in the simulation environment once a choice has been made at a given time. However, these possible actions must be somehow characterized by real-valued variables to be comprehensible by a model like GP or PES. We define here such variables which are split into state and action features and will be used by the models in a similar fashion as dispatching rules (cf 2.2.3).

#### State features $s^{(t)}$

We introduce the sets  $W^{(t)} = \{w_j\}_{j \in jobs}$ ,  $\mu^{(t)} = \{\mu_j\}_{j \in jobs}$ ,  $V^{(t)} = \{\sigma_j^2\}_{j \in jobs}$  respectively the sets of weights, process time means and process time variances of all waiting jobs at time  $t$ . The state vector  $s^{(t)}$  is then composed of the main statistics on these sets:

- $\max(W^{(t)})$ ,  $\min(W^{(t)})$ ,  $\text{average}(W^{(t)})$ ,  $\text{med}(W^{(t)})$
- $\max(\mu^{(t)})$ ,  $\min(\mu^{(t)})$ ,  $\text{average}(\mu^{(t)})$ ,  $\text{med}(\mu^{(t)})$
- $\max(V^{(t)})$ ,  $\min(V^{(t)})$ ,  $\text{average}(V^{(t)})$ ,  $\text{med}(V^{(t)})$

#### Action features $a^{(t)}$

We characterize an action  $a^{(t)} = (b, l) \in \mathcal{A}_t$  by a vector of the following features at time  $t$ :

- $|b|$  the number of products in batch  $b$ .
- $w$  the weight associated to the products in batch  $b$ .
- $\tau = d - t_{now}$  the remaining time before the products in  $b$  become late.  $\tau < 0$  when the products are already late.
- $\mu_{current}$  and  $\sigma_{current}^2$  the process time mean and variance of the current job type to process in batch  $b$ .
- $\mu_{next}$  and  $\sigma_{next}^2$  the process time mean and variance of the next job type to process (both equal to 0 if the current job type is the last one).
- $LST_{current}$  the line setup time of line  $l$  associated with the current job to process in  $b$ . Note that  $LST_{current} = 0$  if the line  $l$  is already processing the same job type.

- $LST_{next}$  the line setup time associated with the next job in  $b$ . Note that  $LST_{next} = 0$  if the current job is the product's last one.
- $T_{max}^{buffer}$  the expected time it takes to process the longest buffer in line  $l$ .

### 3.3 Objective function

A dispatching rule  $R$  is evaluated via an objective function  $F$  called Total Weighted Tardiness (TWT) which we aim to minimize. The value  $F(R)$  is computed at the end of an episode, which is defined as an entire simulation run that ends when all generated products have been completed.

$$F(R) = \begin{cases} 0 & \text{if } \mathcal{T} = \emptyset \\ \sum_{j \in \mathcal{T}} \tilde{w}_j (C_j - d_j) & \text{if } \mathcal{T} \neq \emptyset \end{cases} \quad (3.6)$$

where  $\mathcal{T} = \{j \in products; C_j - d_j > 0\}$  is the set of tardy products at the end of an episode and  $\tilde{w}_j = \frac{w_j}{\sum_{i \in products} w_i}$ ,  $C_j$  and  $d_j$  are respectively the normalized weight, completion date and due date of product  $j$ . The scheduling and evaluation process is illustrated in Figure 3.3.1.

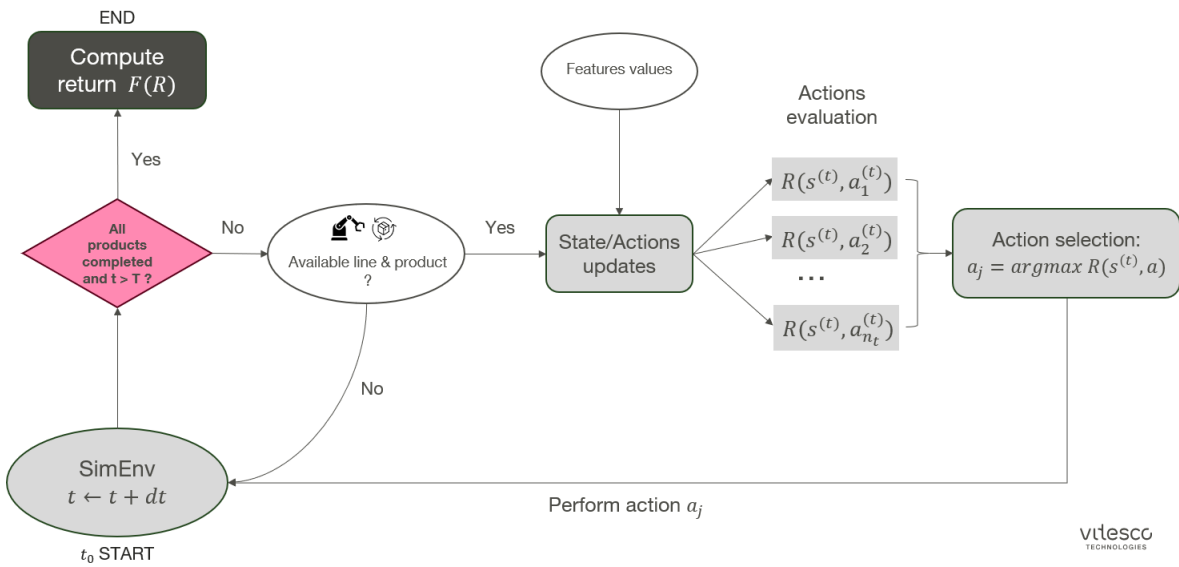


Figure 3.3.1: Scheduling process during one episode using a fixed dispatching rule  $R$  with a generation time  $T$

In practice, due to stochastic events in the system,  $F(R)$  is a random variable. Therefore, we seek to mitigate variance by evaluating the performance of a given scheduling agent with a Monte Carlo sampling on  $n_{episodes}$  episodes. Thus, the optimization problem at hand is:

$$\min_R \bar{F}(R) = \frac{1}{n_{episodes}} \sum_{i=1}^{n_{episodes}} F(R)_i \quad (3.7)$$

### 3.4 Training procedure

#### 3.4.1 Parallel Evolution Strategies

Since  $\bar{F}(R_\theta)$  is unknown and sparse (sampled from the simulation and only computed at the end of an episode), we apply PES described in the background section 2.3.2. The training algorithm 2.3.6 is summarized in Figure 3.4.1.

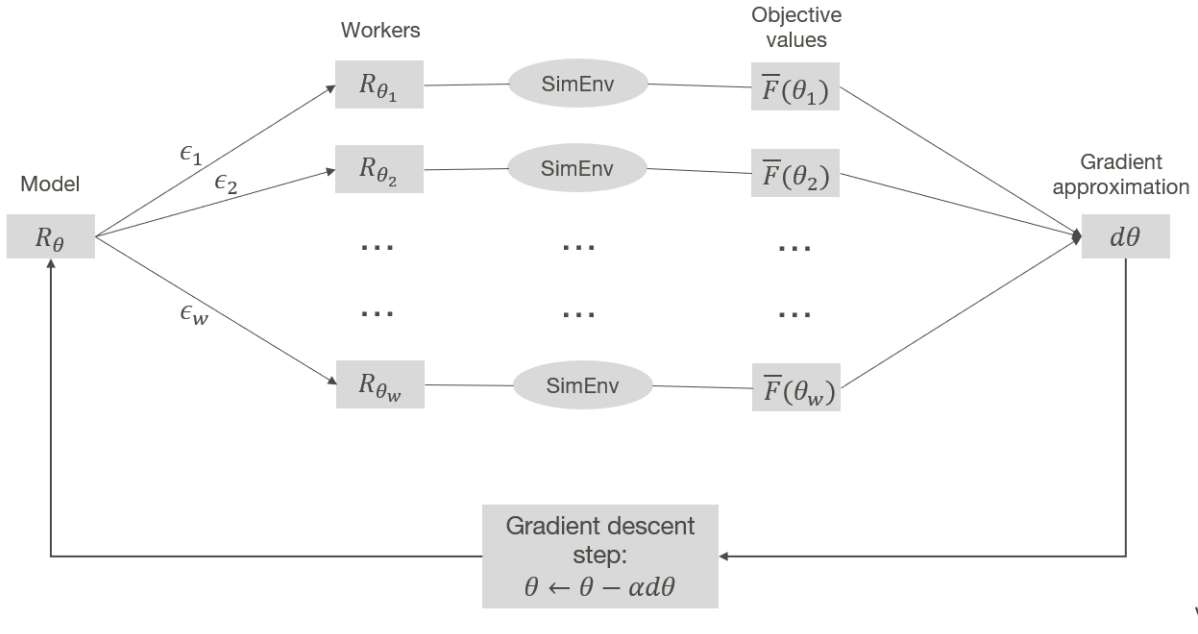


Figure 3.4.1: Illustration of PES training procedure with  $W$  workers

Some additional improvements are made during training, following indications suggested in [22]. To reduce variance, we apply mirror sampling [24], i.e. we always evaluate a pair of perturbation  $\epsilon, -\epsilon$ . We also find it useful to apply fitness shaping [25], a monotonically increasing function with the rank of the returns  $\{\bar{F}(\theta_i)\}_{i=1, \dots, W}$  which makes it possible to scale down the returns of workers that can take arbitrary high values and distort the gradient

approximation during training. Moreover, we apply weight decay [26] to the weights of the network: in addition to the parameters update, we apply  $\theta_{weights} \leftarrow \theta_{weights} - \alpha\beta\theta_{weights}$  where  $\beta > 0$  is the weight decay factor and  $\alpha > 0$  is the learning rate. This prevents the parameters from growing very large compared to the perturbations.

### 3.4.2 Genetic Programming

The GP framework is the same as described in part 2.3.1 where the fitness evaluation of  $R_{GP}$  is done by  $\bar{F}$  (cf 3.3). In particular, we initialize a population of syntax trees representing dispatching rules. A selection operation is then performed with  $\bar{F}$  to select the best individuals in the population which are then evolved using crossover and mutation operations. Therefore, new individuals are created and replace the worst ones in the population. This process is repeated until a stopping criteria is met (typically the maximum number of iteration) and the best individual encountered during the whole process is provided as the final solution.

# Chapter 4

## Experiments

In this chapter, we present the choices made on the parameters of the simulation environment along with the design of both PES and GP training. The results on test scenarios are then presented and compared with two reference models. The behavior of PES and GP are finally more closely analyzed by investigating how they handle remaining times, weights, line setup times and accumulation inside lines.

### 4.1 Computational complexity

Initially, models were trained on a simulation environment design that reflects the company's real production context with regard to the flow of products. However, each iteration in the training process was quite long and required important memory usage. For example, with a design of 10 lines composed of 10 machines, a processing flow of 30000 products/day during 28 days, using  $W = 32$  workers for PES training with  $n_{episodes} = 5$  to compute  $\bar{F}$ , one training iteration takes approximately 18 minutes and uses 80 Go of RAM. This problem is mainly due to the time scale used for processing times: the simulation environment tracks large amounts of products whose states change every second over long periods of time. Indeed, in *Salabim*, handling large amounts of components and interactions can be computationally costly by keeping track of every components and features. In addition, since the scheduling rule waits when no valid action is available, products may accumulate in the system when the generation rate is high and exacerbate the amount of tracking.

One way to overcome this problem would be improving the simulation environment by

optimizing the code. However, this would not require relevant mathematical background and would be time-consuming with no clear guarantee on significant improvements of the models' training time-efficiency. Rather, the simulation design was directly simplified. In particular, processing times  $\mathcal{P}^{(j)}$  were scaled up, along with setup times  $s^{(j)}$  and line setup times  $LST^{(j)}$ : everything becomes as if the product unit (in the simulation) represented a rack of 40 products (in real life) which stick together all along. While this choice departs from the reality of production plants, there is no loss of generality when it comes to the scheduling problem. Similarly, without loss of generality, the number of machines  $n_M$  within a line is reduced, the generation time  $T$  and the generation time rate  $\delta_T$  are scaled down, and accumulation in the system is mitigated by enforcing an approximated constant flow during training. Also, small values for  $n_{episodes}$  and the number of workers  $\mathcal{W}$  are used since the complexity scales linearly with  $n_{episodes}$  and a limited amount of virtual CPUs is available for distributed training.

These changes make it possible to scale down the time for one iteration of the training process by six: approximately 3 min by iteration for GP and PES with the following designs. As a result, the total training time for 200 iterations has been reduced to around 10 hours.

## 4.2 Design parameters

### 4.2.1 Simulation environment

Following the previous justifications about the environment simplification, Table 4.2.1 presents the simulation design which was used to train the models. In particular, a small tightness factor is used to have urgent products in the system and force the models to prioritize in an overwhelmed environment.

In addition, the set  $\mathcal{J}$  of different possible job types is created using  $a \in \{10, 11, 12\}$ ,  $loc \in \{10, 11, 12, 13\}$ , and  $scale \in \{1, 2, 3\}$  (cf. 3.1.3). Hence there are  $N_{\mathcal{J}} = 36$  different possible job types.

<b>Parameter</b>	<b>Value</b>	<b>Description</b>
$n_L$	10	Number of lines
$n_M$	2	Number of machines
$c_1, c_2$	1	Capacities of machines
$c$	1	Tightness factor
$dt$	0.1	Time step for scheduling (h)
$\delta_T$	1	Generation time step (h)
$T$	24	Time of generation (h)
$n_b$	{8, 9, 10, 11}	Possible number of batches created at each generation
$L_{sizes}$	{2, 3}	Possible batch sizes (number of products per batch)
$L_w$	{1, 2, 4}	Possible weights
$LST_{min}$	5	Minimum line setup time (min)
$LST_{max}$	15	Maximum line setup time (min)
$s_{min}$	1	Minimum machine setup time (min)
$s_{max}$	2	Maximum machine setup time (min)

Table 4.2.1: Table of simulation parameters

## 4.2.2 Parallel Evolution Strategies

### Model architecture

The model’s architecture used here is a dense ANN with four layers: input layer of size  $N = 22$  (total number of features of interest), two hidden layers of size 128, and an output layer of size 1. The activation functions used are tanh on both hidden layers. The implementation was done using PyTorch, a Python library for artificial neural networks and deep learning [27].

### Training design

The hyper-parameters used for training with PES are presented in Table 4.2.2.

Parameter	Value	Description
$\theta_0$	-	Random initial parameters handled directly in PyTorch
$\mathcal{W}$	32	Number of workers
$\alpha$	0.05	Learning rate
$\sigma$	0.05	Standard deviation
$\beta$	$5 \times 10^{-4}$	Weight decay
$I$	180	Number of iterations
$n_{episodes}$	5	Number of returns to evaluate $\bar{F}(R)$

Table 4.2.2: Table of PES training hyper-parameters

### 4.2.3 Genetic Programming

The parameters used for GP are inspired from [17]. The basic operations that can be instantiated for internal nodes are:  $max$ ,  $min$ ,  $abs$ ,  $+$ ,  $-$ ,  $*$ ,  $div$  where  $div$  is the protected division operation:

$$div(a, b) = \begin{cases} a/b & \text{if } b \neq 0 \\ 1 & \text{otherwise} \end{cases} \quad (4.1)$$

and where  $abs$  is computed on the unique downstream node and the rest is computed between the two unique downstream nodes. For terminal nodes, the possible values are the  $N = 22$  features of interest introduced in 3.2.3. The other parameters are presented in Table 4.2.3. The details on what these parameters and methods represent are described in paper [28].

Parameter	Value
Minimum depth of a tree	1
Maximum depth of a tree	8
Initial population size	32
Tournament size	5
One-point crossover probability	0.85
Uniform mutation probability	0.1
Number of episodes to evaluate $\bar{F}$	5
Number of iterations ( $I$ )	180
Initialization method	Ramped Half-and-Half

Table 4.2.3: Table of GP training design

#### 4.2.4 Validation scores

At each iteration of the training processes, validation scores are extracted from the current PES rule and current best GP rule via  $\bar{F}$  using  $n_{episodes} = 200$  under identical random/dynamic conditions (the random seeds in the simulation are controlled). These scores are more interesting than the ones directly used during training for two reasons: 1) the Monte-Carlo scores are more robust since they are computed on significantly more episodes ( $n_{episodes} = 200$ ) than during training ( $n_{episodes} = 5$ ), and 2) both models are evaluated on the same random conditions, which is not necessarily the case during training. The validation curves are illustrated in Figure 4.2.1.

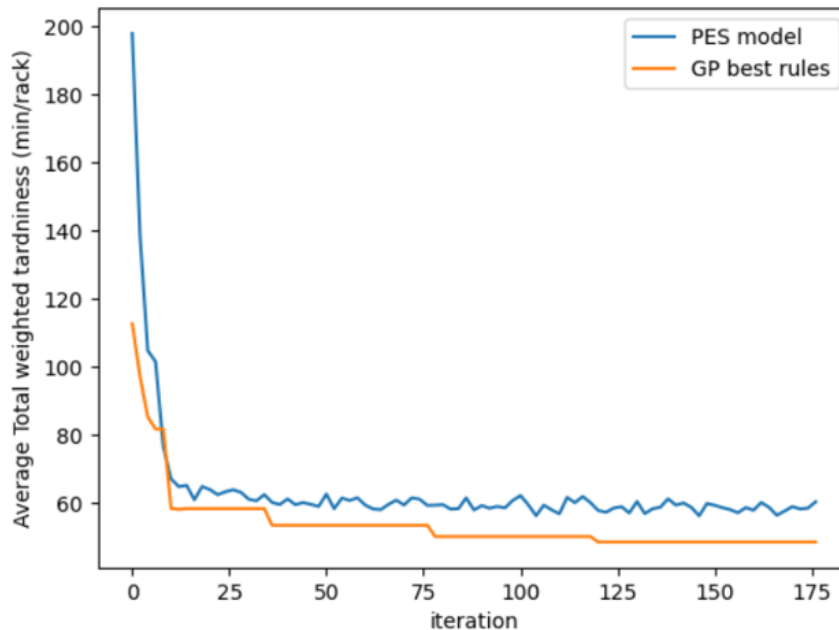


Figure 4.2.1: Validation scores for PES and GP

Since each iteration is evaluated under the same dynamic conditions, it is expected to observe the flat step regions during GP training. Indeed, it reflects the best rule from a generation to another to remain the same for some time before a better individual is found during the evolution process. For PES, this behavior is not observed since the parameters keep changing from an iteration to another.

The training process appears to be effective in achieving optimal dispatching rules that result in lower objective values in dynamic environments. In particular, GP seems better. Both models at iteration 180 are kept for results analysis.

### 4.3 Reference models

We describe in this section two reference dispatching rules that will be used for comparison with GP and PES.

#### 4.3.1 Handcrafted dispatching rule

The first reference model implemented is a handcrafted dispatching rule inspired from the literature in the context of the Dynamic Flexible JSSP [29]. More precisely, using the same feature definitions as in part 3.2.3, the scheduling choice at a time  $t$  is done as follows:

If no valid batch is available or no valid line is available:

1.  $t \leftarrow t + dt$

Else:

1. For any valid batch  $b$ , introduce the quantity  $Q(b) = n_M(\mu_{current} + \mu_{next}) + LST_{current} + LST_{next}$  which approximates the expected minimum remaining time of each product contained in  $b$  in the system.
2. Among all valid batches, if one of them is already late, then select the batch  $b$  which maximizes the quantity:

$$C_{min}(b) = w|b|[t_{now} - d + Q(b)]$$

It is a measure of the minimum cost associated to the batch.

3. Otherwise, select the batch  $b$  which minimizes the weighted slack time:

$$WST(b) = \frac{d - t_{now} - Q(b)}{w|b|}$$

4. If it exists, select the valid line  $l$  which is already processing the same job type as  $b$ , otherwise select the valid line  $l$  which is processing the job  $j$  with minimum  $LST^{(j)}$ .
5. Perform action  $a = (b, l)$
6.  $t \leftarrow t + dt$

This customized dispatching rule tries to minimize the total weighted tardiness via comprehensible rules which are designed to mitigate the negative impact on every decision step.

### 4.3.2 Weighted Earliest Due Date

Similarly, a simple rule close to the standard Earliest Due Date (EDD) is implemented, called Weighted Earliest Due Date (WEDD). The scheduling choice at time  $t$  is done as follows:

If no valid batch is available or no valid line is available:

1.  $t \leftarrow t + dt$

Else:

1. Among all valid batches, if one of them is already late, then select the batch  $b$  which maximizes the quantity  $w(t_{now} - d)$ .
2. Otherwise, select the batch  $b$  which minimizes the weighted due date:  $\frac{d}{w}$
3. If it exists, select the valid line  $l$  which is already processing the same job type as  $b$ , otherwise select the valid line  $l$  which is processing the job  $j$  with minimum  $LST^{(j)}$ .
4. Perform action  $a = (b, l)$
5.  $t \leftarrow t + dt$

This dispatching rule is a simplification of the previous one, only focusing on the weighted due date of products.

## 4.4 Results

### 4.4.1 Testing conditions

Test evaluations are performed on GP, PES and both reference models previously described. Each model  $R$  is evaluated 500 times by computing  $F(R)$  on 500 different episodes. In order to analyze the generalization properties of the models, they are evaluated under different scenarios than the one used during training. However, the random/dynamic conditions of a specific scenario are kept identical between each model for the sake of comparison.

To ensure consistency, the models is evaluated using identical control parameters to those used during training (cf 4.2.1). In each scenario, only one parameter from Table 4.4.1 is modified.

<b>Design Parameter</b>	<b>Test Values</b>
Generation time step $\delta_T$ (in minutes)	{120, 60, 45}
Line setup times ranges (in minutes)	{[1, 3], [5, 15], [15, 45]}
Tightness factor $c$	{3, 2, 1}
Possible weights $L_w$	{{1}, {1, 3, 9}, {1, 4, 16}}

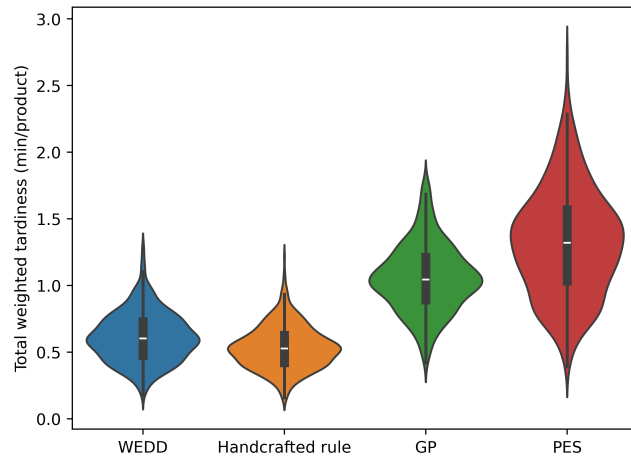
Table 4.4.1: Table of test parameters

## 4.4.2 Test scenarios

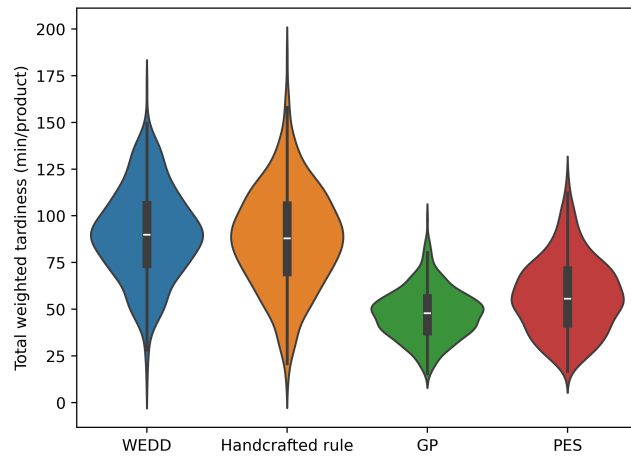
The test distributions for each of the modified control parameters previously mentioned are displayed in Figures 4.4.1, 4.4.4, 4.4.2 and 4.4.3. These figures are violin plots: they display the distributions of counts of the objective values obtained when running the simulation several times (500) for each model. The complexity of the scheduling problem increases from top to bottom in each figure. In every case, approximately the same amount of products is generated and completed (500), except in the first case (250, 500, 600 respectively from top to bottom).

### Test A: Generation time steps

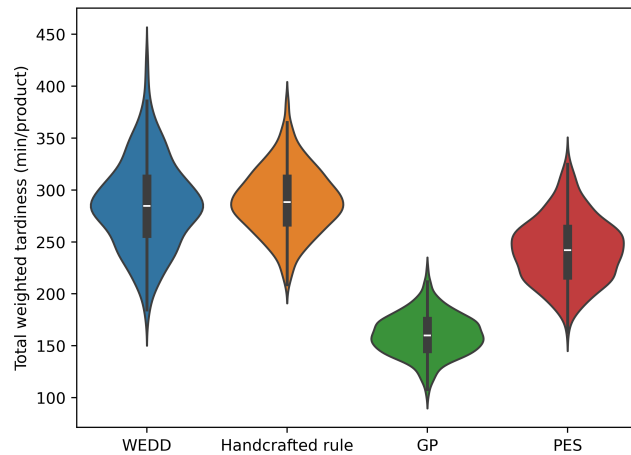
$\delta_T$  is the time step at which new batches of products are generated in the system.



(a)  $\delta_T = 120$  min



(b)  $\delta_T = 60$  min



(c)  $\delta_T = 48$  min

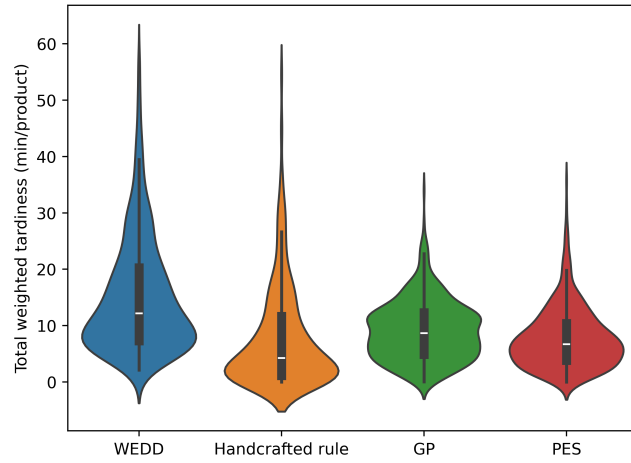
Figure 4.4.1: Generation time steps

In Test 4.4.2, the reference models perform slightly better on both the upper and middle tests.

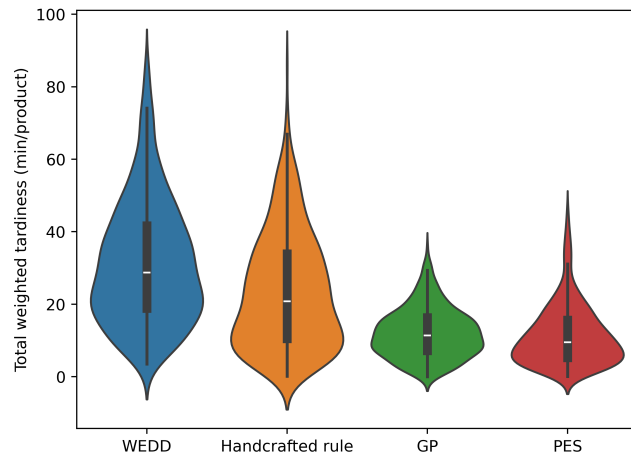
It is because the system is not stressed in these scenarios: the rate of product generation is small hence the total number of products generated is small and manageable ( $\approx 250$ ). In this case, products are easily assigned to lines and there are no important scheduling choices to make between them. This is the reason why the objective values are quite low and the models perform approximately the same (no high variability in objective values).

### **Test B: Tightness factor**

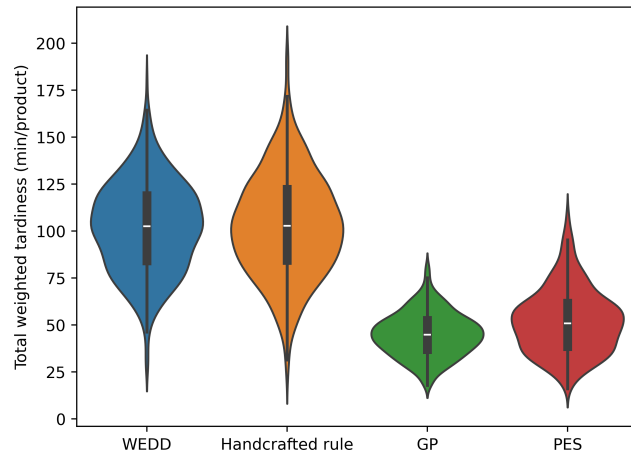
The tightness factor  $c$  characterizes the amplitude of the due dates in the system: the higher  $c$  is, the larger are the due dates of generated products.



(a)  $c = 3$



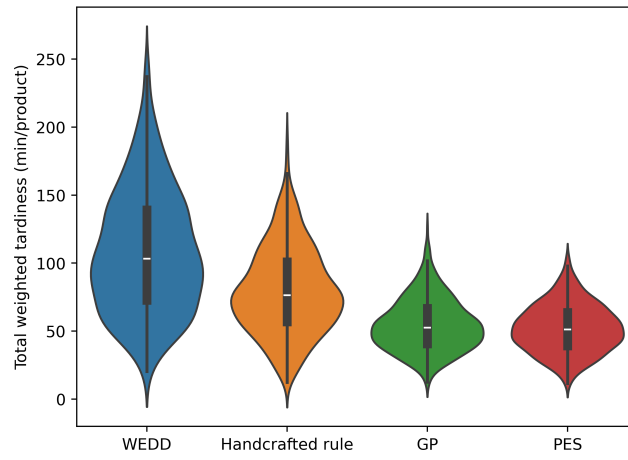
(b)  $c = 2$



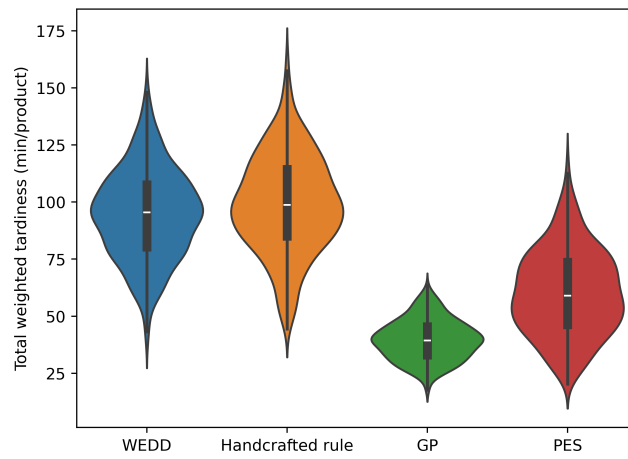
(c)  $c = 1$

Figure 4.4.2: Tightness factors

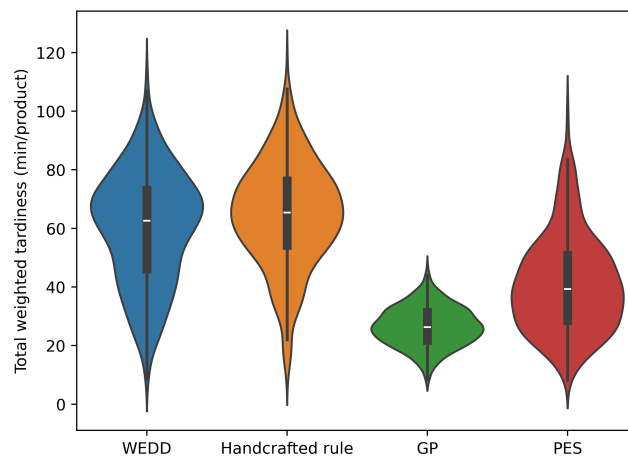
Test C: Weights



(a)  $L_w = \{1\}$



(b)  $L_w = \{1, 3, 9\}$



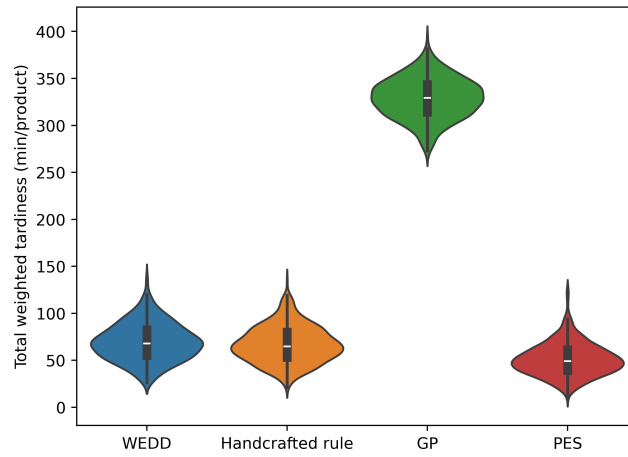
(c)  $L_w = \{1, 4, 16\}$

Figure 4.4.3: Weights

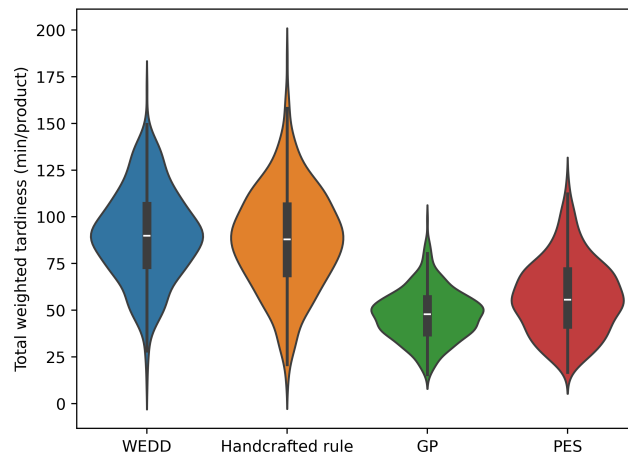
The list of weights  $L_w$  contains the possible weights assigned to products when they are generated.

#### **Test D: Line setup times**

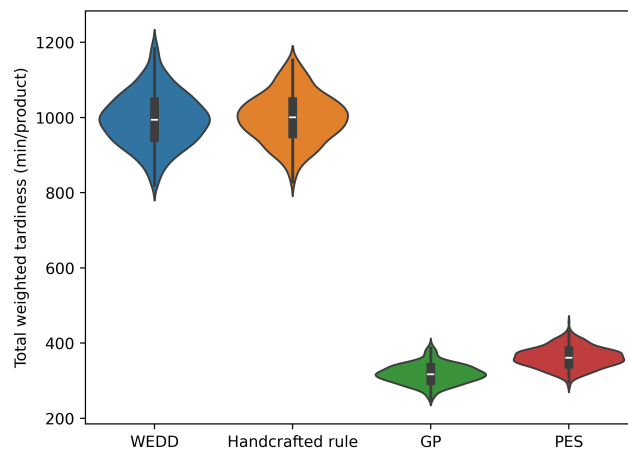
Line setup times are the time to wait before processing a product on a line which is currently processing a product of different type.



(a)  $LST \in [1, 3]$  min



(b)  $LST \in [5, 15]$  min



(c)  $LST \in [15, 45]$  min

Figure 4.4.4: Line setup times

Except in Test A (4.4.1) which has already been commented, it is straightforward to see that

both GP and PES perform significantly better than the reference models in all the other medium and complex scenarios (middle and bottom plots of each figure). In particular, GP has tighter evaluation distributions with better average values in every complex case and almost on all middle case scenarios, which would seem promising. However, GP performs poorly on Test B (Figure 4.4.4a) where all other models perform much better. This drop of performance is surprising and deserves more detailed analysis.

### 4.4.3 Analysis

#### Assessment of special case 4.4.4a

It is particularly surprising that GP performs poorly on case 4.4.4a with  $LST \in [1, 3]$  since low line setup times entail a faster production. The scheduling problem should therefore be easier than more complex scenarios where GP performs well.

Table 4.4.2 presents the Overall Equipment Effectiveness (OEE) and cumulative line setup time for each line after one episode during GP scheduling. In our case, the OEE of a given line is fully defined by its availability [30], calculated as the ratio between the total processing time of the busiest machine on the line and the total time of the production run for that line.

<b>Production Line</b>	<b>OEE (%)</b>	<b>Cumulative LST (h)</b>
Line 1	87	5.02
Line 2	78	4.21
Line 3	66	3.70
Line 4	64	3.28
Line 5	51	2.60
Line 6	45	2.40
Line 7	40	2.33
Line 8	36	2.43
Line 9	33	2.05
Line 10	31	2.03

Table 4.4.2: OEE and cumulative LST for each line after one episode during GP scheduling.

It is clear that the usage of lines is uneven which intuitively does not exhibit the right scheduling behavior. This phenomenon could not appear in a context where the generation of products would be high since the scheduling model is always forced to make a scheduling choice at every small time step  $dt$  when at least one action is available. Therefore, in this

case, GP often has a choice for scheduling between several lines, and it appears that it often chooses the same lines regardless of others that are almost unused. As a result, some of the lines are overused: products accumulate more in the buffers and therefore take more time to be completed. This behavior also forces some lines to re-parameterize more to new products, hence increasingly requiring line setup times and reducing the global production flow. These consequences are unfavorable for minimizing the total weighted tardiness.

By contrast, PES schedules products more evenly between lines—under the same random conditions— and produces smaller cumulative line setup times, as shown in Table 4.4.3.

<b>Production Line</b>	<b>OEE (%)</b>	<b>Cumulative LST (h)</b>
Line 1	67	2.52
Line 2	64	2.58
Line 3	70	2.43
Line 4	78	2.48
Line 5	76	2.48
Line 6	69	2.48
Line 7	68	2.10
Line 8	69	2.53
Line 9	85	2.40
Line 10	75	2.23

Table 4.4.3: OEE and cumulative LST for each line after one episode during PES scheduling.

This simple yet revealing case sheds lights on GP rule’s lack of comprehension of both products accumulation inside lines and the importance of line setup time values when they are small. By contrast, PES rule seems more robust to these features.

#### **Assessment of regular case 4.4.4b**

We assess GP and PES rules’ behavior on a design identical to the one used during training, where both models perform better than the reference rules and GP is slightly better than PES (cf Figure 4.4.4b). Both models handle line setup times similarly: the total cumulative line setup up times used in the system are similar, as shown in Table 4.4.4.

However, GP and PES rules manage remaining times differently. Figure 4.4.5 illustrates the statistics on remaining times of current available products over time during an episode and for both scheduling models. The same random/dynamic conditions are used in both cases.

	<b>GP</b>	<b>PES</b>
Total cumulative LST (h)	123	124

Table 4.4.4: Total cumulative line setup time for GP and PES after one episode.

In both figures, the sticks along the x-axis represent each time new batches are generated. A negative remaining time  $\tau = d - t_{now} < 0$  characterizes a currently late product.

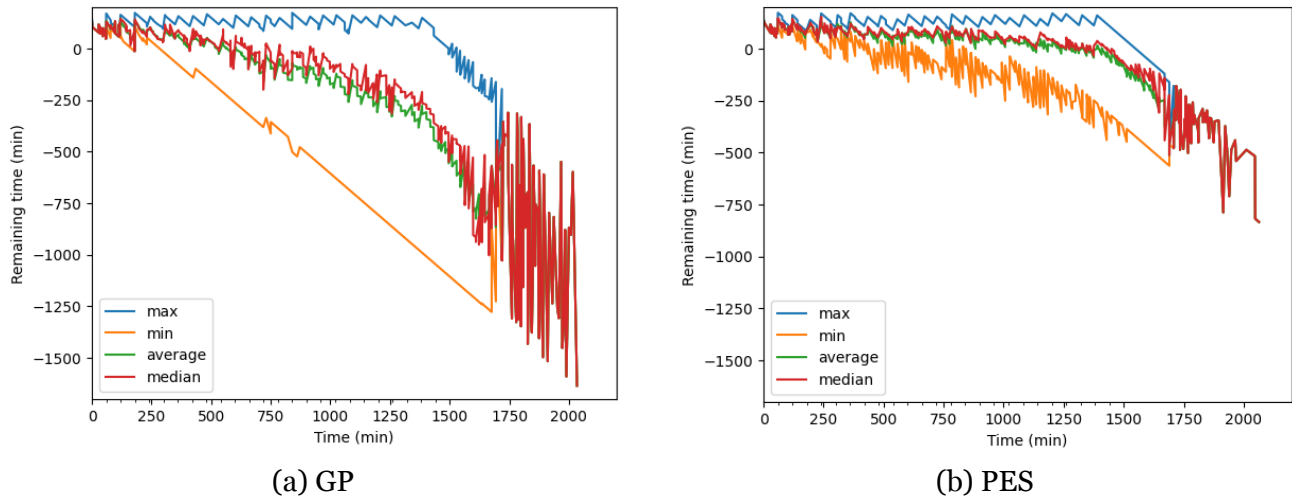


Figure 4.4.5: Statistics on remaining times of current available products over time during one episode

In sub-figure 4.4.5a, a clear almost linear decrease of the minimum remaining time occurs during the GP scheduling episode. Since the average and the median are also importantly decreasing over time, it suggests that GP is ignoring several products for a long period of time regardless of their current tardiness. These overlooked products become importantly late and are only assigned near the end of the episode when few products remain. By contrast, sub-figure 4.4.5b suggests that PES tries hard to keep current delays at bay by actively scheduling low remaining time products.

GP and PES rules also manage weights differently. Figure 4.4.6 illustrates the statistics on weights in a similar way as remaining times. It is straightforward to see that GP schedules products such that waiting products keep low weights (sub-figure 4.4.6a), whereas PES does not necessarily (sub-figure 4.4.6b).

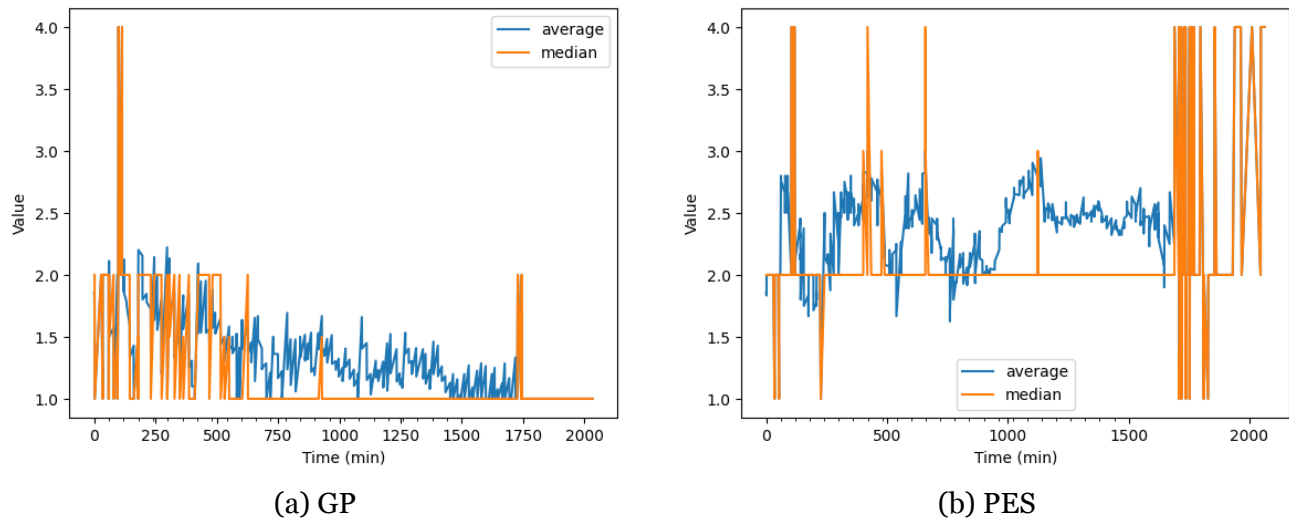


Figure 4.4.6: Statistics on weights of current available products over time during one episode

### Summary

Thus, both GP and PES training procedures in the simulation environment provide dispatching rules outperforming standard handcrafted dispatching rules in all medium and complex test scenarios, and with comparable performance on simple scenarios at the exception of one case for GP. This particular case demonstrates some kind of over-fitting: that is, GP learns during training to schedule products efficiently on a quite complex environment, yet this strategy is not suited for a particular simpler case. Moreover, by analyzing the evolution of some features of interest during scheduling, it appears that an important part of GP’s scheduling strategy is to prioritize products with high weights at the expense of late products. Even though this strategy pays off regarding the objective values produced in most of the scenarios, it feels like some kind of “cheating”: some products are deliberately overlooked for long periods of time—because of their low weights—which produces high delays. In an industrial context, this behavior can be interpreted as ignoring small clients’ orders to focus on more important ones: it might be interesting to schedule products like this in some cases, but it can also be unsuitable when excessive delays are critical from some clients’ perspective.

On the other hand, PES does not suffer from such a variability in performance from a scenario to another. It also tends to prioritize products with low remaining time over those with high weights, while understanding the importance of line setup times and accumulation of products within lines. For these reasons, PES appears more robust than GP, with

better generalization properties and an appropriate strategy with regard to the industrial context.

# Chapter 5

## Conclusion

### 5.1 Summary

Thus, in this thesis, the problem of dynamic scheduling for manufacturing production in a flexible context is addressed. First, a simulation environment is implemented, handling all the physic and dynamic interactions between products, lines and machines in the system. This simulation is controlled by many parameters which provide flexibility on the characteristics one seeks to introduce in the environment. Then, we designed two scheduling models along with their training procedures inspired from recent state-of-the-art methods: Genetic Programming and Parallel Evolution Strategies. By extensively interacting in the simulation environment, these models can learn to improve their behavior under complex dynamic variability. Although GP reaches the best performance on complex scenarios, it exhibits tricky scheduling strategies in industrial contexts where its suitability is closely related to the business strategy, and lacks understanding in more simple scenarios where it might drop in performance. By contrast, PES have better generalization properties and outperforms standard dispatching rules, while exhibiting different yet suitable behavior in an industrial context as well.

### 5.2 Validity and limitations

Even though these methods seem promising for dynamic scheduling, the highlighted generalization properties are only acceptable within the scope of our work and within

the simulation environment itself: establishing generalization properties of a simulation-based method on real-world applications requires much more effort and research work. Moreover, for the sake of modeling, several assumptions were made not only to build a satisfactory simulation environment in a relatively short period of time, but also to simplify the computational complexity of the training procedures. These assumptions discarded some real-world effects and widened the gap between the simulation and real-world applications.

## **5.3 Further research**

This work aimed at paving the way to developing simulation-based learning procedures to address dynamic scheduling in manufacturing production environments. We believe that it provides the first steps in that direction for Vitesco Technologies. Many improvements and further research are possible to enhance the potential of PES and GP and produce solutions closer to the reality of production plants. We introduce potential paths forward.

### **5.3.1 Improve simulation environment**

The fidelity of the simulation to real-world constraints and scenarios can be improved. In particular:

- Modeling line setup times e.g. by enforcing the line to finish all current products being processed before processing another one from a different type.
- Adding the rate of defective products for each machine.
- Introduce machine breakdowns to complexify the dynamic variability.
- The scheduling model can be modified such that it can choose to wait even if some actions are available. Doing so adds flexibility to the scheduling behavior.
- Process and setup times can be better fitted to the company's data, e.g. using supervised machine learning techniques.
- Buffers between machines can have finite sizes, which would force the model to forecast potential bottlenecks.

- The set of possible job types can be defined more precisely by investigating the real characteristics of products in the company's plant.
- Transportation times might be introduced in order to model operators.
- Products might be taken out of a line if necessary and sent to another one to be resumed at a similar operation.

### 5.3.2 Improve model training

- The PES model can be fine-tuned by investigating the best hyper-parameters to use in order to have a more efficient and potentially more global training procedure. Since the model training is quite long, one may find useful to investigate Bayesian optimization, either with "off-the-shelf" Python libraries such as *bayes-optim*, or with more advanced libraries like *ax* in PyTorch and *BoTorch*.
- The implementation of PES training procedure can be optimized, particularly with regard to the distributed parallel computing: it should be possible to run distributed actors on several virtual CPUs with smaller bandwidth when exchanging information to compute the gradient approximation. We refer to the article from OpenAI [22] for further details.
- For GP, hyper-parameters can also be fine-tuned and one may find interesting to introduce an attention mechanism during training to focus on impactful branches [17].
- GP can also be improved by changing the objective function or adding specific constraints in the simulation to enforce a better behavior dealing with line setup times and accumulation within lines.

### 5.3.3 Other methods

- Instead of using a unique model handling both product selection and line assignment, one might build two different models, one for each task, and investigate training both while synchronizing their choices via multi-agent reinforcement learning procedures [31].
- The single-objective scheduling problem can be changed to a multi-objective one. This might require several major changes in the way the problem was formulated.

# Bibliography

- [1] PwC, *Industry 4.0 - opportunities and challenges of the industrial internet*, <https://www.pwc.nl/en/assets/documents/pwc-industrie-4-0.pdf>, [Online; accessed 8-July-2024], 2014.
- [2] L. Varela, G. Putnik, C. Alves, N. Lopes, and M. Cruz-Cunha, “A systematic review of manufacturing scheduling for the industry 4.0,” in Jan. 2022, pp. 237–249, ISBN: 978-3-031-14316-8. DOI: 10.1007/978-3-031-14317-5\_20.
- [3] C. Cozzolino, B. Christiansen, E. Vallejos, M. Sorce, and J. Visk, *Job shop scheduling*, [https://optimization.cbe.cornell.edu/index.php?title=Job\\_shop\\_scheduling](https://optimization.cbe.cornell.edu/index.php?title=Job_shop_scheduling), [Online; accessed 4-July-2024], 2021.
- [4] X. Liang, W. Song, and P. Wei, “Dynamic job shop scheduling via deep reinforcement learning,” in *2023 IEEE 35th International Conference on Tools with Artificial Intelligence (ICTAI)*, 2023, pp. 369–376. DOI: 10.1109/ICTAI59109.2023.00060.
- [5] L. Tiacci and A. Rossi, “A discrete event simulator to implement deep reinforcement learning for the dynamic flexible job shop scheduling problem,” *Simulation Modelling Practice and Theory*, vol. 134, p. 102948, 2024, ISSN: 1569-190X. DOI: <https://doi.org/10.1016/j.simpat.2024.102948>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1569190X24000625>.
- [6] C. Özgüven, L. Özbakır, and Y. Yavuz, “Mathematical models for job-shop scheduling problems with routing and process plan flexibility,” *Applied Mathematical Modelling*, vol. 34, no. 6, pp. 1539–1548, 2010, ISSN: 0307-904X. DOI: <https://doi.org/10.1016/j.apm.2009.09.002>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0307904X09002819>.

- [7] C. Cebi, E. Atac, and O. K. Sahingoz, "Job shop scheduling problem and solution algorithms: A review," pp. 1–7, 2020. DOI: 10.1109/ICCCNT49239.2020.9225581.
- [8] J. Xie, L. Gao, K. Peng, X. Li, and H. Li, "Review on flexible job shop scheduling," *IET Collaborative Intelligent Manufacturing*, vol. 1, no. 3, pp. 67–77, 2019. DOI: <https://doi.org/10.1049/iet-cim.2018.0009>. eprint: <https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-cim.2018.0009>. [Online]. Available: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-cim.2018.0009>.
- [9] J. C. Tay and D. Wibowo, "An effective chromosome representation for evolving flexible job shop schedules," vol. 3103, Jun. 2004, pp. 210–221, ISBN: 978-3-540-22343-6. DOI: 10.1007/978-3-540-24855-2\_19.
- [10] B. Tagtekin, M. U. Öztürk, and M. K. Sezer, "A case study: Using genetic algorithm for job scheduling problem," *CoRR*, vol. abs/2106.04854, 2021. arXiv: 2106.04854. [Online]. Available: <https://arxiv.org/abs/2106.04854>.
- [11] V. Dabhi and S. Chaudhary, "Developing postfix-gp framework for symbolic regression problems," Jul. 2015. DOI: 10.1109/ACCT.2015.114.
- [12] X.-S. Yang, *Metaheuristic optimization: Algorithm analysis and open problems*, 2012. arXiv: 1212.0220 [math.OA]. [Online]. Available: <https://arxiv.org/abs/1212.0220>.
- [13] T. Raghu and C. Rajendran, "An efficient dynamic dispatching rule for scheduling in a job shop," *International Journal of Production Economics*, vol. 32, no. 3, pp. 301–313, 1993, ISSN: 0925-5273. DOI: [https://doi.org/10.1016/0925-5273\(93\)90044-L](https://doi.org/10.1016/0925-5273(93)90044-L). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S092552739390044L>.
- [14] C. Ferreira, G. Figueira, and P. Amorim, "Effective and interpretable dispatching rules for dynamic job shops via guided empirical learning," *CoRR*, vol. abs/2109.03323, 2021. arXiv: 2109.03323. [Online]. Available: <https://arxiv.org/abs/2109.03323>.
- [15] O. Holthaus and C. Rajendran, "Efficient dispatching rules for scheduling in a job shop," *International Journal of Production Economics*, vol. 48, no. 1, pp. 87–105, 1997, ISSN: 0925-5273. DOI: [https://doi.org/10.1016/S0925-5273\(96\)00068-](https://doi.org/10.1016/S0925-5273(96)00068-)

0. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925527396000680>.
- [16] M. Zhang, Y. Lu, Y. Hu, N. Amaitik, and Y. Xu, “Dynamic scheduling method for job-shop manufacturing systems by deep reinforcement learning with proximal policy optimization,” *Sustainability*, vol. 14, p. 5177, Apr. 2022. DOI: 10.3390/su14095177.
- [17] S. Salama, T. Kaihara, N. Fujii, and D. Kokuryo, “A novel feature selection for evolving compact dispatching rules using genetic programming for dynamic job shop scheduling,” *International Journal of Production Research*, vol. 60, pp. 4025–4048, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:250562624>.
- [18] W. Mouelhi-Chibani and H. Pierreval, “Training a neural network to select dispatching rules in real time,” *Computers & Industrial Engineering*, vol. 58, no. 2, pp. 249–256, 2010, Scheduling in Healthcare and Industrial Systems, ISSN: 0360-8352. DOI: <https://doi.org/10.1016/j.cie.2009.03.008>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360835209000953>.
- [19] DEAP contributors, *Deap documentation*, <https://deap.readthedocs.io/en/master/>, [Online; accessed 5-July-2024], 2009-2023.
- [20] A. Kratsios, “The universal approximation property,” *Annals of Mathematics and Artificial Intelligence*, vol. 89, no. 5, pp. 435–469, 2021, ISSN: 1573-7470. DOI: 10.1007/s10472-020-09723-1. [Online]. Available: <https://doi.org/10.1007/s10472-020-09723-1>.
- [21] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, “An introduction to deep reinforcement learning,” *CoRR*, vol. abs/1811.12560, 2018. arXiv: 1811.12560. [Online]. Available: <http://arxiv.org/abs/1811.12560>.
- [22] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, “Evolution strategies as a scalable alternative to reinforcement learning,” 2017. arXiv: 1703.03864 [stat.ML]. [Online]. Available: <https://arxiv.org/abs/1703.03864>.
- [23] R. van der Ham, *Salabim*, <https://salabim.org/manual/index.html>, [Online; accessed 8-July-2024], 2016.

- [24] D. Brockhoff, A. Auger, N. Hansen, D. V. Arnold, and T. Hohm, “Mirrored Sampling and Sequential Selection for Evolution Strategies,” in *PPSN*, ser. Parallel Problem Solving from Nature (PPSN XI), Warsaw, Poland, Sep. 2010, pp. 11–21. [Online]. Available: <https://inria.hal.science/inria-00530202>.
- [25] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber, “Natural evolution strategies,” in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 2008, pp. 3381–3387. DOI: 10.1109/CEC.2008.4631255.
- [26] M. Andriushchenko, F. D’Angelo, A. Varre, and N. Flammarion, *Why do we need weight decay in modern deep learning?* 2023. arXiv: 2310.04415 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2310.04415>.
- [27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” *CoRR*, vol. abs/1912.01703, 2019. arXiv: 1912.01703. [Online]. Available: <http://arxiv.org/abs/1912.01703>.
- [28] W. B. Langdon, R. I. McKay, and L. Spector, “Genetic programming,” in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds. Boston, MA: Springer US, 2010, pp. 185–225, ISBN: 978-1-4419-1665-5. DOI: 10.1007/978-1-4419-1665-5\_7. [Online]. Available: [https://doi.org/10.1007/978-1-4419-1665-5\\_7](https://doi.org/10.1007/978-1-4419-1665-5_7).
- [29] Z. Wu, H. Fan, Y. Sun, and M. Peng, “Efficient multi-objective optimization on dynamic flexible job shop scheduling using deep reinforcement learning approach,” *Processes*, vol. 11, no. 7, 2023, ISSN: 2227-9717. DOI: 10.3390/pr11072018. [Online]. Available: <https://www.mdpi.com/2227-9717/11/7/2018>.
- [30] IBM, *What is oee?* <https://www.ibm.com/topics/oee>, [Online; accessed 8-July-2024].
- [31] J.-D. Zhang, Z. He, W.-H. Chan, and C.-Y. Chow, “Deepmag: Deep reinforcement learning with multi-agent graphs for flexible job shop scheduling,” *Knowledge-Based Systems*, vol. 259, p. 110083, 2023, ISSN: 0950-7051. DOI: <https://doi.org/10.1016/j.knosys.2022.110083>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950705122011790>.