INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA DE INTELIGÊNCIA ARTIFICIAL




Escalonamento Dinâmico da Produção de uma fábrica

RICARDO HENRIQUE MACEDO SOUSA
Julho de 2022

P.PORTO

i

Dynamic Scheduling of the Production of a Factory

Ricardo Henrique Macedo Sousa

Student nº: 1160900

Dissertation to obtain the Degree of Master in Engineer of Artificial Intelligence

Supervisor: Dr. Carlos Fernando da Silva Ramos, Main Coordinator Professor of Instituto Superior de Engenharia do Instituto Politécnico do Porto

Jury:

President

Dr. Isabel Cecília Correia da Silva Praça Gomes Pereira, Coordinator Professor of Instituto Superior de Engenharia do Instituto Politécnico do Porto

Other members

Dr. Paulo António da Silva Ávila, Coordinator Professor of Instituto Superior de Engenharia do Instituto Politécnico do Porto

Dr. Carlos Fernando da Silva Ramos, Main Coordinator Professor of Instituto Superior de Engenharia do Instituto Politécnico do Porto

Porto, June 2022

Dedicatory

In the Bachelors' Thesis I dedicated it to my grandmother, that had passed away in that year. Now, I will dedicate it to my grandfather, for the same reason and for all the support he gave me while he was alive. All the history and little curiosity about Portugal, despite being irrelevant for the development of this thesis gave me the curiosity that brought me here, to data and looking for new insights, this time discovered by me.

Rest in peace, you deserve it.

Thank you.

Resumo

A indústria 4.0 está já aí. Com o seu desenvolvimento, as fabricas estão a ficar cada vez mais ciberfísicas. Sendo assim, sistemas de escalonamento dinâmico que podem tratar altas quantidades de dados fazem cada vez mais sentido. Estes sistemas podem reduzir os custos de uma fábrica, otimizando a produção na mesma ao máximo.

Em primeiro lugar, o estado da arte dos sistemas de escalonamento foi estudado, de forma a perceber qual seria a melhor opção para implementar um. Duas possibilidades foram implementadas: um algoritmo genético (GA) e um algoritmo de colónia de abelhas artificiais (ABC). Estas soluções utilizam o makespan como função objetivo e oferecem uma grande variabilidade nas soluções devido ao seu sistema de gerar soluções iniciais para os algoritmos. Estes algoritmos foram validados e testados para o Flexible Job Shop Scheduling Problem, resultando daí a conclusão de que o ABC é melhor que o GA por 2% e que o ABC está perto de outros algoritmos de estado da arte.

Esta tese também tem como objetivo o tratamento de possíveis eventos dinâmicos que podem acontecer numa fábrica, criando heurísticas para o tratamento destes. A melhor solução implementada com estas heurísticas tem como objetivo diminuir a entropia entre o antigo e o novo planeamento de produção. Foram feitos testes comparando o algoritmo com algoritmos do estado da arte, obtendo bons resultados.

Para concluir, dois sistemas de escalonamento estático foram desenvolvidos e várias heurísticas para tratar diferentes disrupções na fábrica foram desenhadas.

Palavras-Chave: Otimização, Escalonamento, Flexible Job Shop Scheduling Problem, Algoritmo Genético, Algoritmo de Colonia de Abelhas Artificiais.

Abstract

Industry 4.0 is around the corner. As it develops, factories are becoming more and more cyber-physical. Thus, a dynamic scheduling system that can treat the ever-increasing amount of data produced every day by a factory makes more sense as it goes. These can reduce costs and resource waste, optimizing the efficiency of production to its maximum.

In a first instance, static scheduling algorithms were studied and compared, to understand which was the best. Two of them were implemented: a genetic algorithm (GA) and an artificial bee colony algorithm (ABC). These solutions provide variability in its initial populations by generating them through different rules and use the makespan as the function to evaluate the solutions. Furthermore, the algorithms were validated and tested for the Flexible Job Shop Scheduling Problem, where operations can be produced in several machines. The conclusion was that the ABC algorithm had a 2% improvement against the Genetic Algorithm.

This thesis also comprises the treatment of possible dynamic events that happen in a factory, through the development and testing of heuristics for the problem. The proposed implementation can treat two types of disruptions: job disruptions and machine disruptions.

The solution provided tries to diminish the entropy between the previous and new schedule, by keeping the schedule the same until a certain point in time. The best performing algorithm was tested under these possible disruptions, allied with the heuristics providing good results when compared to the state of the art.

To conclude, two static scheduling systems were developed and several heuristic methods to treat different disruptions were designed.

Keywords: Optimization, Scheduling, Flexible Job Shop Scheduling Problem, Genetic Algorithm, Artificial Bee Colony Algorithm.

Acknowledgements

The first person I would like to thank is to Carlos Ramos for all the support and availability every time that I needed. Also, I would like to thank my family, my parents in special, for all the emotional and monetary support and effort done for me to get here, the end of my master's degree. At last, I would like to thank the love of my life, Inês Silva, for all the support throughout the most difficult and demotivating times of the development of this thesis.

Index

Image Index

Table Index

Equation Index

Acronyms and Symbols

Acronyms List

ABC	Artificial Bee Colony
CDR	Complex Dispatching Rules
d	Due date
ERP	Enterprise Resource Planner
FJSSP	Flexible Job Shop Scheduling Problem
GA	Genetic Algorithm
IDE	Integrated Development Environment
ISEP	Instituto Superior de Engenharia do Porto
IoT	Internet Of Things
J	Job
JSSP	Job Shop Scheduling Problem
L	Lateness
M	Machine
C	Makespan
MES	Manufacturing Execution System
MEIA	Mestrado em Engenharia de Inteligência Artificial
MOFJSSP	Multi-Objective Flexible Job Shop Scheduling Problem
MOR	Most Operations Remaining
MWR	Most Work Remaining
O	Operation Time
PSO	Particle Swarm Optimization
POX	Precedence Preserving Order-Based Crossover
PPS	Precedence Preserving Shift Mutation

PT	Processing time
ST	Starting time of the operation
T	Tardiness
VNPSO	Variable Neighbourhood Search Particle Swarm Optimization
VNS	Variable Neighbourhood Search
WCT	Weighted Completion Time
W	Weight of the job

Subscripts

I	Operation
J	Job
K	Machine
max	Maximum Value

1 Introduction

This document is written in the scope of the dissertation of the MSc Programme on Artificial Intelligence (MEIA) of the *Instituto Superior De Engenharia do Porto* (ISEP). It documents the process of finding a good scheduling algorithm that can deal with dynamic events that happen in a factory. Throughout this document, the main concepts regarding the theme will be concerned. The previous work, developments and results of the study will be presented.

1.1 Motivation

The Industrial Revolutions have been turning points on how the people live their lives in their eras. The first industrial revolution began with the appearance of the first manufacturing machine. It allowed for some of the heavy work to be replaced by machinery. The production became faster and cheaper, permitting its results to reach people who could not afford it. The second one, with the usage of electricity, permitted the mass production of products, enabling once again the possibility for people that could not meet the expense of the product to be able to buy it. The third one, with the improvements on the automation of the manufacturing processes, provided the possibility to produce faster. The fourth (Industry 4.0), happening now, powered by the Internet of Things and Artificial Intelligence, enables the possibility of having cyber-physical machines with modern control systems and hyper-flexible assembly lines[1]. The fact that this project is in the scope of this last revolution is a motivating factor to develop it.

Furthermore, due to the covid-19 pandemic and the increase in demand in some industries, such as the automotive industry, some materials started lacking in the factories all over the world. The most predominant materials that have been lacking in 2021 have been silicon and semiconductors made from it[2]. This implies the need to produce better scheduling systems that can change rapidly and with ease, so that the factories are more responsive and agile.

1.2 Production Management

Production management is the application of management techniques to the manufacturing area of a factory[3][4]. Its focus is the production planning and control. Production scheduling is the area that concerns the planning of how the production orders will be organized for manufacturing on each machine.

1.2.1 Scheduling problems

The scheduling problems are challenges to solve the issue of defining which production order will be produced in each machine. Usually, these problems have resource constraints. This means that just like on a shop floor, the resources for every part of the manufacturing process are limited. Organizing the production order with these constraints, optimizing a certain objective function (for example minimizing the time to process all the production orders) is the solution to a scheduling problem. The scheduling problem that was tried to solve was the Flexible Job Shop Scheduling Problem (FJSSP).

1.2.2 Flexible Job Shop Scheduling problem

The Flexible Job Shop Scheduling Problem (FJSSP) is a generalization of the Job Shop Scheduling Problem (JSSP). The JSSP defines the challenge of, given a set of Jobs (J= {0,…,n}) (sets of manufacturing operations in a factory) and a set of Machines (M= {0,…,m}) that can process them, define when and where should each Job be processed so that an objective function is fulfilled (being the simplest minimizing the makespan of a set of Jobs). Every Job contains a finite number of operations that should be performed, possibly by different machines with different functions for it to be complete. The order of operations for each job is fixed. Solving the JSSP does not imply rearranging the operations of a Job [7]. It contains three main conditions [6]:

- It is impossible to start a new operation on a job while a previous operation is in process
- A machine can only process an operation at a time
- An operation cannot be interrupted after it has been started.

The Flexible Job Shop Scheduling Problem, besides the conditions presented above for the JSSP, also formulates that an operation can be processed in several machines with the same purpose.

1.2.3 Objective Functions

There are several common objective functions on Job Shop Scheduling Problems, concerning different ways to optimize the production of a factory. The makespan is the maximum completion time of the jobs. The objective function based on this metric would be to minimize it, as it corresponds to high utilization of the machines, reducing the production whole

production cost. The equation 1 below concerns the calculus of the completion time of a production order. The C_i represents the completion time of job J_l[7]

$$C_{max} = max(C_j) = max(ST_{jik} + PT_{jik}), (j \in J, k \in M, i \in O_{ji}) \quad (1)$$

Equation 1 – Maximum Completion time of a production order

The Total Weighted Completion Time is represented in the equation below. It adds to the makespan the weight w_j of a job related to other jobs. The objective function is to minimize this said time[7].

$$\mathrm{WCT} = {}_{n}\sum_{j=1} w_j C_j, (j \in J) \quad (2)$$

Equation 2 – Total Weighted Time Formula

Maximum Lateness corresponds to how late the latest job would be delivered to the client. The objective is to minimize the maximum lateness. Equations 3 and 4 below represent the lateness of a job and the maximum lateness of a schedule[7].

$$L_j = C_j - d_j, (j \in J) \quad (3)$$

Equation 3 – Formula for the lateness of a job

$$L_{max} = \max(L_i, \ldots, L_n) \quad (4)$$

Equation 4 – Formula for the maximum lateness of a schedule

The number of tardy jobs, and its minimization, can also be seen as an objective function. Instead of considering how late jobs would be, considering the number of late jobs can also be an important metric. Equation 5 below show this objective function formula[7].

$$T_j = \max\{0, c_j - d_j\}, (j \in J) \quad (5)$$

Equation 5 – Formula for the number of tardy jobs

It is also possible to combine these objective functions to get a multi-objective function[7].

1.3 Tools

The main tool used for this project is the Python programming language. Python is an object-oriented programming language that is optimal for prototype developments due to being fast to program in it [8]. To program in this language, Google Colab was used. This software is a web-based IDE (Integrated Development Environment) to program in python. The fact that it is easy to use was the most proponent consideration to choose this software instead of any other. Regarding libraries on python, *"pyeasyga"* was used to produce the Genetic Algorithm in python and the *Hive* framework to develop the artificial bee colony algorithm. For project management, the Trello [9] software was used. It is a software for organization of a project, having in mind agile methodologies. Although this is a single person project, the fact that a software like this is

used provides a way to control what has been done and what is still missing completion. Git will be used for project versioning, so that no data is ever lost, and time is not wasted.

1.4 Methodologies

The project methodologies are going to be presented in this subsection. Besides this introduction, two types of methodologies will be addressed: Development and Test methodologies.

For the project to be organized and planned, the Scrum management technique was used. Scrum[10] is an agile methodology for project management. It contains four different moments: dailies, planning, review, and retrospective. As the project is done only by one person, the methodology was altered to fit for a one-man project:

- Dailies: A small moment of thought of what needs to be done for the day and what can be the worst threats and problems that can rise throughout the development. The idea here is for the biggest issues to be tackled first, and for the planning of the day to work better
- Planning: The weakly planning of what was needed to be done for the project to meet the delivery deadline
- Retrospective: The insightful moment of thinking what went bad and well, to improve for the next sprint.

As there were not regular deliveries, the concept of review (a periodical delivery of a working solution, even though not complete) was not implemented in the development of the project.

1.4.1 State of The Art Study Methodology

To know what exists in the area, a state of the art was conducted. To do so, the first thing done was to find surveys on how to solve the FJSSP. After considering the possibilities, a more extensive search was done, by way to solve, on how to solve the problem of this thesis. Two types of solutions were found: Heuristic and Meta-Heuristic methods.

1.4.2 Development Methodology

The project has as one of its main focuses the development of optimization algorithms to solve the above-mentioned problem. To do so, after studying the possibilities, some algorithms were implemented. Comparison between the algorithms was performed and an interpretation of the results was done.

1.4.3 Test Methodology

The first part of the testing done was between the two proposed solutions. These were compared with each other to see which provided better results. The best solution was tested by comparison with some of the solutions presented in the state-of-the-art section of this document. Using the data of these articles (number of jobs, number of machines and overall structure of the factory modelled), a comparison and interpretation of the performance of the proposed solution was done. The Makespan and the improvement between them (presented in equation 6 will be the metrics used to compare the algorithms

$$improvement = \frac{gbestbenchmarks - gbestSolutionPresented}{gbestbenchmarks} \text{x100} \quad (6)$$

Equation 6 – Improvement of a solution

Several types of algorithms were chosen to compare the best developed solution. The approach that was decided was picking different algorithms so that the implementation was not only compared with others from the same family, but also with different solutions.

Throughout the tests and results section, there is a metric, present in several articles which is the best-known lower bound. This value represents the lowest value ever found for a given optimization problem with X number of machines, Y number of Jobs, Z number of Operations and T_{XYZ} time to process an operation from a certain job on a certain machine.

1.4.4 Visualization methodology

This subsection presents the visualization techniques that will be used in this document.

The schedule presented is provided in the format of a Gantt Chart. This provides a simple manner for readers to interpret the schedule. A Gantt Chart is a form of chart that shows a schedule. It has 3 main components:

- Job: Defined as rectangles in the chart
- Task Duration: The length of the rectangles represents how much time the
- Machines: Defined as the vertical Axis of the chart

Figure 1 shows an example of a Gantt Chart.

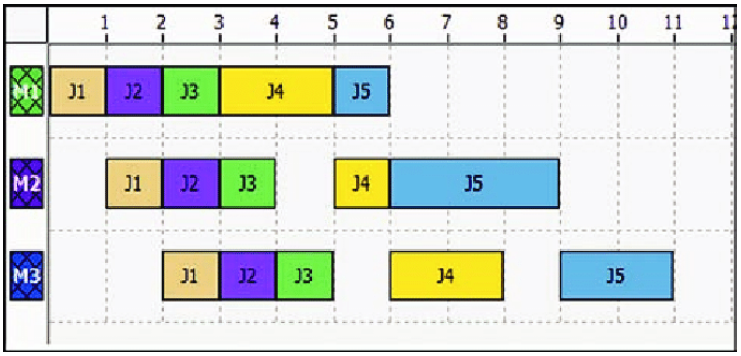

Figure 1 - Example of a Gantt Chart [11]

1.5 Document Overview

Apart from the above Introduction, section 2 represents the state of the art pertinent to the development of the problem, section 3 contains proposed algorithms and its development. Section 4 comprehends the tests of the solution. At last, section 5 comprehends the conclusions taken from the exploration of the algorithms, some concerns and future work.

2 State of the art

This section presents what has already been investigated and discovered in the field of flexible scheduling. There are two main areas of focus that are pertinent to study, besides defining scheduling and rescheduling: heuristic-based techniques and Meta-Heuristic-based techniques.

2.1 Scheduling and Rescheduling

Scheduling is the process that decides what machine should be producing for each production order at any given time. It defines how the resources on a factory should be used to optimize a certain metric, given certain constraints. These constraints are usually given by what the manufacturing processes are on a factory. There are two types of scheduling: Static and Dynamic Scheduling. The first one is built at the beginning of a timeframe and is planned to be followed no matter what happens. The second one considers the unexpected changes that can happen in a factory. A machine breakdown, a new production order with high priority, or a lack of a certain raw material would make a static schedule unfollowable. A dynamic schedule can adapt to these unexpected changes in the shop floor [12]

Dynamic Scheduling has several categories, according to the manner that it reacts to changes in production.

Complete reactive scheduling does not require a baseline schedule. The scheduling is made on the resource level based on what production order that needs to use that machine has the highest priority. It has a low computational burden, being the fastest way to find what should be the next production order to be produced on a certain machine. This is usually made with the usage of dispatching rules (rules that prioritize all the jobs that are waiting for processing on a machine)[12].

Pro-active scheduling requires a baseline schedule. It is built according to the information of previous production experiences, considering what are the usual disruptions to the process. It has small scheduling cycles where the new disruptions and experiences from iteration 1 are added to the previous experiences of iteration 2, making the software more robust as several iterations of scheduling are made [5]

Predictive-reactive scheduling is the mix of the above-mentioned categories. It requires the creation of a baseline schedule that is to be followed until there is a disrupting event in production. When it happens, a reschedule is done. The magnitude, periodicity, and techniques of that rescheduling should be set at the creation of this type of scheduling. There are several approaches regarding these three metrics[12].

Regarding the magnitude of the rescheduling, two approaches can be followed: Complete Rescheduling and Schedule Repair. The first one is the process of regenerating a full schedule from scratch. It is an approach that requires high computational power and, therefore, is not recommended for unexpected events that happen in a small timeframe until the machine stops. Nevertheless, the results provided are the most proximate to optimal as possible. The second one is the process of trying to mitigate the unexpected event through minor changes in the pre-established schedule. It requires less computational power, although it does not provide as good results as the previously mentioned approach[13].

There are three main policies concerning the periodicity of the rescheduling,. The first one, Periodic Rescheduling, implies that rescheduling is done at a very predefined time interval, regardless of how many unexpected issues occurred in the factory. The second one, Event-Driven Rescheduling, implies that rescheduling is done every time an unexpected issue appears. The third one, Hybrid Rescheduling, is a combination of the two above mentioned. The rescheduling is done periodically but certain events may trigger a new rescheduling even if the time before the next one has reached[14].

2.2 Heuristic Techniques

A heuristic is a technique that seeks good solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular solution is [15]. One of the best options is the dispatching rules. Dispatching rules are rules that prioritize all the jobs that are waiting for processing on a machine. A dispatching rule inspects the waiting jobs whenever a machine has been freed and selects the job with the highest priority. The most basic dispatching rules are the Simple Priority Rules. These rules define what is the product with the highest priority to be produced on a machine based on different factors. These can be the shortest processing time, the earliest due date, among others[16]. Nevertheless, these were found to not perform well across every performance measure[17]. To overcome this limitation, composite dispatching rules (CDR) were created. *Barman* [17], for instance, used three different Simple Priority Rules in different machines, providing better results than only one SPR for all.

On the other hand of CDR's are mixed priority rules on the same machine. *Chandrasekharan et Al.* [18] present five possible combinations with optimistic results. This approach was also shown to have better results than simple priority rules. The main challenge with using CDR's is finding the best combinations possible. To overcome this, Zhang *F et Al.* [19] and *Tay J et Al.* [20] present options for evolving dispatching rules using genetic algorithms. *Jing Huang et Al.* [21] shows a possibility to improve dispatching rules using a genetic algorithm and fuzzy logic. Paolo Priore et Al. [22] studies a way of predicting dispatching rules with machine learning techniques. *Teymourifar et Al.* [23] studied the possibility to predict the best dispatching rule for a given moment using gene expression programming. *Jun et Al.* [24]presents a way to learn dispatching rules based on random forests. It consists in three parts: schedule generation, rule learning with data transformation, and rule improvement with discretization. *Oukil et Al.* [25]proposes a way to rank dispatching rules using a Ranking algorithm.

2.3 Meta-Heuristic Techniques

Besides the above-mentioned techniques, meta-heuristic methods have also been studied as a possibility to calculate the best schedule for a factory. These techniques aim to find global or near-optimal solutions at a high computational cost [26]. They have the main disadvantage of not guaranteeing that the optimal solution will be found.

2.3.1 Particle Swarm Optimization

Particle swarm optimization (PSO), developed by *Kennedy and Eberhart* [27], is a swarm-based heuristic that iteratively attempts to enhance a candidate solution for a specified quality measure. The basis behind this algorithm is to mimic the behaviour of a flock of birds. In the algorithm, a swarm of particles flies among the candidate solutions trying to discover the optimal solution. Each particles' behaviour is affected either by the best local solution within a certain neighbourhood or by the best global solution. PSO uses the population concept and a certain performance measure to evaluate the quality of the particle. This algorithm focuses on two different phases: Exploration and Exploitation.

- Exploration – Particles on the exploration phase explore the search space extensively to give more variability to the set of solutions.
- Exploitation – Particles on the exploitation phase focus on particularly promising regions withing the search space.

These two phases should be balanced to increase the algorithm's performance for a given problem [28]

There have been many examples of pertinent studies in the scheduling area with this algorithm. *Xia et Al.* [29] present a solution combining PSO with Simulated Annealing. It uses iterative local search for every particle in the swarm to promote a better local solution. *Liu et Al.* [30] hybridize variable neighbourhood search (VNS) and PSO algorithms (VNPSO) to solve the Multi-Objective

Flexible Job Shop Scheduling Problem (MOFJSSP). The two objective functions are simultaneously minimized by a dynamic weighted sum function. The same group of investigators also studied the possibility to use Multiple Particle Swarm Optimization. They use a group of swarms to calculate the operation order and another group to find the best machine assignment for each order. *Sadrzadeh et Al.* [31] present research that introduces sequence-dependent set-up times as a new constraint to the solution of the scheduling problem with PSO. *Xu et Al.* [32]presented a solution that combines a genetic algorithm with particle swarm optimization. *Wang et Al.* [33]provides a solution for a dynamic flexible job shop scheduling problem, focusing on job arrivals. It focuses on keeping the old jobs as stable as possible and on treating new jobs efficiently. *Nouiri et Al.* [34] shows a two stage PSO solution to deal with a dynamic flexible job shop scheduling problem considering machine breakdowns. *Kamble et Al.* [35] combined PSO with Simulated Annealing for both multi-objective and only one objective possibilities. *Ding et Al.* present an improved PSO using novel encoding and decoding for the FJSSP.

2.3.2 Artificial Bee Colony

Artificial Bee Colony (ABC), proposed by *Karaboga* [36]is a metaheuristic technique based on the intelligent attitude of the honeybees within a swarm. The algorithm defines the concept of food source. It also defines the concept of three kinds of bees: employed bees, onlooker bees, and scout bees (the last two can also be called unemployed bees, as they are not assigned to any food source).

- Unemployed bees are responsible for looking for food sources. Employee bees become scouts when their food sources are exhausted. There are two types of unemployed foragers: scouts and onlookers. Scout bees are searching the environment surrounding the nest for new food sources and onlooker bees are waiting in the nest and establishing a food source through the information shared by employed foragers.
- Food source - The value of a food source depends on diverse factors, such as the proximity to the nest, the richness of the source, the concentration of its energy, and the ease of extracting this energy. Each of them is considered a solution to the optimization problem.

The algorithm also defines two types of behaviour: the recruitment to a nectar source and the abandonment of a source which are used for self-organization. The former behaviour is positive feedback. That is, an increase in the nectar amount of food source increases the number of onlookers visiting them. The latter is negative feedback.

Despite the similarities with the functioning of beehives, this algorithm is an iterative algorithm ran by a computer. It starts by associating employed bees to a randomly generated food source. On each iteration the employed bees go to a new food source in its neighbourhood and evaluate the quality of the food source. After this process, these bees share the information they have gathered with the onlooker bees, that choose for which food source they will move. The amount

of onlooker bees in a food source is usually associated with the fact that it is a quality food source. Scout bees, after being X number of iterations in the same food source without improvement move to a new food source.

The termination criteria for the algorithm are the number of iterations defined is finished or no more optimizations can be found.

Wang et Al. [37] introduce a Pareto-based ABC algorithm to solve a multi-objective FJSSP. *Li et Al.* [38]present other solution considering maintenance activities. *Carolina Azevedo*[5] suggests yet another solution for the FJSSP. *Zhang et Al.* [39] show a dynamic self-learning ABC algorithm to solve the FJSSP, concerning the insertion of new Jobs. It utilizes the Q learning algorithm for self-learning.

2.3.3 Tabu Search

Tabu Search is a method based on local search heuristic. It was proposed by *Glover*[40]. Its goal is to search for a potential solution to a problem and to check its local neighbours, expecting to find a better solution. It has three primary schemes. The first one is the use of flexible memory structures to search and evaluate performance of past moves. The second one is used to control the moves to be applied on the current search iteration. The third one is used to diversify the search, by using different types of memory:

- Short-term memory – The list of recently found solutions.
- Intermediate-term memory – The list of some of the best areas of the search-space, used to intensify search in that area, thus increasing the amount of exploitation.
- Long-term memory – Drives the search into new areas, thus increasing the amount of exploration.

A lot of researchers have studied the possibility of using this algorithm to solve scheduling problems. *Baykasoğlu* [41] presents a solution to use this method to solve the MOFJSSP. This solution is complemented with a linguistic-based scheme where its data is represented as a context-free grammar. *Jia and Hu* [42] present a novel path-relinking algorithm based on Tabu Search to solve the same problem. A Pareto-optimality approach is used to find the best solution and a mid-term memory structure. Firstly, a path relinking heuristic is defined to explore the search space efficiently. Then, a dimension-oriented refinement search is done to enhance this algorithm. At last, Tabu Search with jump tracking is used to solve the problem. *Li et Al.* [43] provides a solution based in the pareto methodology to address the MOFJSSP. It combines the makespan, critical machine workload and earliness/tardiness objective functions.

2.3.4 Evolutionary Algorithms

The idea of evolutionary algorithms was introduced by *Holland* [44]. This approach takes the idea of Darwin's evolution of the species theory and applies it to optimization problems. It uses

the concepts of population, crossover, mutation, and selection to continuously improve the solution until an optimal solution is found, given an objective function. The population is the search space of every iteration of the genetic algorithm, each individual is mapped as a solution for a problem. The crossover, mutation and selection concepts are used for matting the individuals within a population. Selection is the process of choosing each two individuals that are going to be mated. The crossover is the process of combining the individuals to get children, according to the parents' information. At last, the mutation operation provides a way for individuals to be different from their parents, by mutating chromosomes in its identity. The individuals from a population are ranked by a certain fitness value, which can be compared among them.

Huang et Al. [45] provides a solution to the given problem with this technique. *Vlašić et Al.* [46] provide a solution to solve this problem by initializing the population using dispatching rules. *Reddy et Al.* [47] provide a solution to treat real time events in the type of problem defined above. *Liu et Al.* [48] considers the machine switching off-on operation and uses a genetic algorithm to provide a solution.

2.4 Reinforcement Learning

At last, the State of the art also shows good results when using the reinforcement learning technique to find suitable results to the Flexible Job Shop Scheduling Problem and its Dynamic branch.

Reinforcement Learning is an unsupervised machine learning technique that learns by trial and error. The model starts with no prior knowledge and tries to come up with a solution. Nevertheless, the model designer must provide the rules to the model. These rules comprise the reward mechanism, which can be compared to the objective function in the previous stated meta-heuristic models, and what the structure of what it should do Besides that, it is up to the model to discover which is the best way to reach the best reward possible. Much like the meta-heuristic methods, this technique does not guarantee the best possible results for the problem, but it provides near-optimal solutions[49].

Chen et Al. [50] provide a solution for the FJSSP using a knowledge-based reinforcement, taking in consideration a multi-criteria objective function that uses the concepts of machine processing speed, setup time, idle time, and transportation between machines. It also considers the time-of-use electricity price constraint. *Lei et Al.* [51] provide a deep reinforcement learning framework using a multi-pointer graph network. *Shu et Al.*[52] provide a solution using a Deep Q Network (DQN). *Han et Al.*[53] provide other deep reinforcement learning algorithm taking advantage of Recurrent Neural Networks (RNN).

2.5 Comparison Between Algorithms

All of the Algorithms presented above provided good results in solving this problem. As the intended solution should include both scheduling and re-scheduling, it was decided that a meta-heuristic algorithm should be the go-to approach for this thesis. The near-optimal solution that they provide in a reasonable timeframe is enough, considering the time to implement and the capabilities that these algorithms have.

Between the meta-heuristic algorithms presented in Section 2.3, it was chosen to develop two of them. The reason why two should be developed is to be able to experiment with different algorithms for the actual problem that was proposed to be developed, without only believing in what others have developed on the matter. The choice of the genetic algorithm was simple. According to surveys on the matter by Senvar et Al. [53] and Chaudhry et Al. the Genetic Algorithm is the most experimented algorithm on the matter. Thus, although there is no proof that it is actually better than any of the other algorithms, it was considered as a good metric to choose this method. The choice of the Artificial Bee Colony algorithm was done because it has a lot of similarities with the Genetic Algorithm (that was already chosen due to being widely used) and, therefore, was also considered.

3 Development

To solve the Flexible Job Shop Scheduling Problem, two solutions were developed: a genetic algorithm approach and an ant bee colony approach. The reason for this choice was, besides the interest and knowledge on trying these algorithms, that these are two of the algorithms that provided better results, according to the state of the art[54][55] . For the first approach, *pyeasyga* was the library chosen. For the second problem, the Hive framework was the chosen one.

3.1 Genetic Algorithm

The basic steps of this Algorithm applied to the specific case specified in Table 1:

Table 1 – Steps of a basic Genetic Algorithm Procedure

Steps	Description
Step 1	Current Population= Initial Population, Best Solution = Best Solution of the Population and Best Objective Function Value = Objective Function Value of the Best Solution in the Population.
Step 2	Next Population= Generate Populations from Current Population using Selection and/or Reproduction and/or Mutation.
Step 3	Current Population= Population Next
Step 4	If the Best Objective Function Value of the Population is best than the Best Objective Function Value found so far then: Best Solution = Best Objective Function Value of the Population and Best Objective Function Value = Objective Function Value of the Best Solution in the Population.
Step 5	If the maximum number of populations defined is met, the Stop. Otherwise, return to Step 2.

To develop this algorithm, each solution to the problem will be considered an individual. These individuals will have the information regarding the order of jobs and machines where the jobs will be running, as well as the time taken to do each job on each machine.

The initial population is generated finding, for each job, which is the machine with less workload (minimum processing time) and adding the job to the end of the schedule for that machine. This approach is completely dependent on the order of the jobs in the raw data, something that is not desirable, as two jobs with the same time to produce will always appear in the same machines. Thus, to give more variability to the initial population individuals, a random permutation of jobs is executed after having a complete schedule to some of the individuals.

On the other hand, the order of the operations in each machine must be defined as well. To do so, the schedule must respect the sequence of operations for the same job (Ji_{+1} should always be executed after J_i has been concluded). Apart from that, three dispatching rules were used, on the organization of the operations. The code for each of the dispatching rules is below, with the explanation of the rule.

- Random Dispatching Rule – The Jobs and operations are organized randomly, respecting the sequence of operations

```
RandomDispatchingRule(Jobs){
  Cromossome c = new Cromossome();
        UnorderedTaskVector unordered = GetUnorderedTaskVector()
  OrderedVector = OrderJobs(unordered)
  While( len(OrderedVector)>0){
    Job=Random.choice(OrderedVector)
    Operation = Job.nextOperation();
    Machine m = GetRandomFeasibleMachine(Job,Operation)
    Cromossome.append(task.JobId, task.OperationI, m.MachineId)
    OrderedVector.pop(Job,Operation)
  If(c.IsValidCromossome()){
    Return c
  }
 }
```

- Most Work Remaining (MWR) – The jobs and operations are organized by the Jobs that have more workload to be done

```
MostWorkRemainingDispatchingRule(Jobs){
   Cromossome c = new Cromossome();
         UnorderedTaskVector unordered = GetUnorderedTaskVector()
   OrderedVector = OrderJobs(unordered)
   While( len(OrderedVector)){
     Job=GetMostWorkJob(OrderedVector)
     Operation = Job.nextOperation();
     Machine m = GetRandomFeasibleMachine(Job,Operation)
     Cromossome.append(task.JobId, task.OperationI, m.MachineId)
     OrderedVector.Pop(Job,Operation)
   If(c.IsValidCromossome()){
     Return c
   }
}
```

- Most number of operations remaining (MOR) – The jobs and operations are organized by the greatest number of operations remaining in the job.

```
MostNumberOfOperationsRemainingDispatchingRule(Jobs){
    Cromossome c = new Cromossome();
    UnorderedTaskVector unordered = GetUnorderedTaskVector()
    OrderedVector = OrderJobs(unordered)
    While( len(OrderedVector)){
      Job=GetMostOperationsJob(OrderedVector)
      Operation = Job.nextOperation();
      Machine m = GetRandomFeasibleMachine(Job,Operation)
      Cromossome.append(task.JobId, task.OperationI, m.MachineId)
      OrderedVector.Pop(Job,Operation)
    If(c.IsValidCromossome()){
      Return c
    }
}
```

The probability of using each of these dispatching rules can be changed at will to fine-tune the algorithm for better results.

The result of this scheduling is based on the *Kacem et Al.* [56] task sequencing list proposal, using a triplet *(I,j,k)* to define an operation where

- I is the job the operation belongs to
- j is the order of the operation in the job
- k is the machine the operation was assigned to.

The structure follows a string with length equal to the total number of operations. For the example shown in Figure 1, the structure would be (1,1,3) (1,2,1) (2,1,3) (2,2,4) (3,1,3) (1,3,2) (3,2,1) (2,3,4).

$S = (O_{11}, M_3), (O_{12}, M_1), (O_{21}, M_3), (O_{22}, M_4), (O_{31}, M_3), (O_{31}, M_3), (O_{13}, M_2), (O_{32}, M_1), (O_{23}, M_4)$

Besides the creation of the initial population, four main functions were implemented on top of the *pyeasyga* framework: Selection, Crossover, Mutation, and Fitness.

3.1.1 Selection

The selection function chooses which are the individuals that will breed for the next generation and the ones that don't. Experimentation was done with several methods of selection:

- n-Size tournament – the best of a randomly generated number of individuals is chosen for selection
- Linear ranking – the individuals are ranked according to their fitness and where the probability of choosing the individual for selection is:

$$\mathrm{pi} = \frac{2ri}{N\,(N+1)}\ ,\mathrm{i} = 1,\ldots,\mathrm{N} \quad (7)$$

Equation 7 – probability of choosing an individual for selection

, where r_i is the rank of the individual, I is the number of the individual and N is the population size

- Binary tournament – the best of two randomly selected individuals is chosen for selection.

The results of the experiments were that the binary tournament provided best results. After experimenting with the three solutions with 2000 generations, it was seen that using the n-Size tournament the algorithm started converging to the near-optimal solution found after around 700 generations, the Linear Ranking after 900 generations and the binary tournament after only 500 generations. The pseudo-code for this last type of selection is presented below.

```
BinaryTournament(population,problem){
 Selected =[]
 Total=len(population)
 while(len(Selected!=Total){
  Parents=[]
  for(i in range(0:2)){
    parent=None
    While(True){
     Parent=random.choice(population)
     if(Parent not in Parents){
      break
     }
    Parents.append(Parent)
  Chosen = CalculateMinimumMakespan(population,problem)
  Selected.append(Chosen)
 Return Selected
}
```

Moreover, the solution renews 100% of the mating pool at every iteration, repeating the process as many times as it is needed to meet the number of individuals in the population size, as it was found to have the best results.

At last, from the mating pool chosen previously, pairs of individuals are chosen randomly for reproduction.

3.1.2 Crossover and Mutation

The Crossover and mutation functions provide the logic for the Genetic Algorithm to reproduce its population, having two individuals as a basis for the crossover and one for the mutation. After the individuals have been chosen from the selection operation, two types of operators are done: Assignment Operators and Sequencing Operators.

The Assignment Operators only change the machine the operations are going to be done, without changing the sequence of those operations. The assignment crossover exchanges a random subset of operations between the two parents, without changing the order of the

operations. The assignment mutation exchanges one randomly chosen operation to a different random machine. The intelligent mutation finds the machine with most workload and picks an operation from that machine and moves it to the machine with less workload. The pseudo-code below presents the way to do the assignment operators.

```
AssignmentCrossover(parent1,parent2){
 operationToCross = random.choice(parent1)
 index = parent2.indexOf(operationToCross)
 Child1 = parent2
 Operation2ToCross = Child1[index]
 Child1[index] = operationToCross
 index2 = parent1.indexOf(operation2ToCross)
 Child2 = parent1
 Child2[index] = operation2ToCross
 Return {Child1, Child2}
}

AssignementMutation(individual,problem){
 operationToMutate=random.choice(individual)
 While(True){
  newMachine=randInt(0:problem.NumberOfMachines)
  if(newMachine!=operationToMutate.MachineNr){
    operationToMutate.SetMachine(newMachine)
 }
}

IntelligentMutation(individual,problem){
 machineToRemoveOperation=getMachineWithMostWorkload(individual,problem)
 machineToAddOperation=getMachineWithLeastWorkload(individual,problem)
 operationsForMachine =
getOperationsForMachine(machineToRemoveOperation,individual)
 operationToMoveIndex = random.choice(operationsForMachine)
 operation =
individual.indexOf(machineToRemoveOperation,operationToMoveIndex)
 individual.addOperationToMachine(operation, machineToAddOperation)
}
```

The Sequencing Operators change the order of operations in a machine. These operators must follow the sequence of operations for a given Job. The Precedence Preserving Order-based crossover (POX) suggested by *Kacem et Al.*[56] and Precedence Preserving Shift Mutation (PPS) suggested by *Lee et Al.* [57] was used to guarantee this rule.

The POX method consists in generating two children from two parents. It selects an operation from the first parent, adds, in order, all the operations from that operation's Job to the first child and completes the child with the operations from the second parent. The second parent is generated doing the symmetric to the first child. The PPS method selects an operation from a single parent chromosome and moves it into another position, taking care of the precedence constraints for that operation.

```
PreservingOrderBasedCrossover(parent1,parent2,problem){
 operation_selected = random.choice(problem.jobs)
 positions = getPositionAndOperationOfRelatedOperations(selected,parent1)
 child1 = getPositionAndOperationOfUnrelatedOperations(selected,parent2)
 for(position in positions){
  child1.insertAt(position.index,position.operation)
 }
 operation_selected2 = random.choice(problem.jobs)
 positions2 = getPositionAndOperationOfRelatedOperations(selected2,parent2)
 child2 = getPositionAndOperationOfUnrelatedOperations(selected2,parent1)
 for(position in positions2){
  child2.insertAt(position.index,position.operation)
 }
return {child1,child2}

PrecedencePreservingShiftMutation(individual,problem){
 While(True){
  Operation = random.choice(individual)
  NewIndex= random.choice(1:problem.NrOperations)
  Individual.remove(Operation)
  Individual.append(Operation)
  If(Individual.IsValid(problem)){
   Break;
  }
 }
}
```

Each of the methods can be used, with a probability that can be configured per run of the genetic algorithm.

3.1.3 Fitness

The fitness function provides a way for each individual's performance to be quantified, in order to calculate which is the individual that optimizes production the most.

This function considers the Makespan (referenced in 1.2.4) of the schedule (the maximum time taken by the machines to produce what was scheduled). The objective here is to minimize its value. The maximum machine's makespan is considered as the makespan of the whole schedule.

```
fitnessFunction(individual,problema){
 time=[]
 for(machine in individual.machines){
  time[machine]=0
  for(operation in individual.getOperationsOnMachine(machine)){
   time[machine]+=individual.getProcessedTime(operation)
  }
 }
 Return max(time)
```

3.2 Artificial Bee Colony

The basic steps of this Algorithm applied to the specific case are specified in Table 2:

Table 2 - Steps of a basic ABC Procedure

Steps	Description	Phase
Step 1	Initialize Parameters: Initialize Population of Food Sources Compute objective function and save best food source	Initialization Phase
Step 2	Crossover Mutation	Employed Bee Phase
Step 3	Compare the best food source with every food source	Onlooker Bees Phase
Step 4	Update population, best solution and generate scout bees	Scout Bees Phase
Step 5	If the stopping Criterion has been met, stop. Otherwise, return to 2.	

The Artificial Bee Colony algorithm was used to solve the Flexible Job Shop Problem as well. To do so, the *Hive* framework[58] was used. The Hive framework is a "swarm-based optimisation algorithm based on the intelligent foraging behaviour of honeybees" [58]. It implements most of the algorithm's logic, leaving the problem-specific functions open so that it can be adapted to each optimization problem. These functions, referred as phases, are the initialization phase, employed bees' phase, onlooker bees' phase and scout bees' phase.

In the specific case of the flexible job shop problem, the food source is a scheduling solution. It is structured as two vectors:

- Machine Assignment Vector – Vector that codes the assignment of operations to the machines
- Operations Sequence Vector – Vector that codes the processing sequence of operations for each job.

For example, given the four jobs and machines given in table 3 below:

Table 3 - Example of machines and operations

Operations	Machines			
	M1	M2	M3	M4
O11	4	7	6	5
O12	2	6	-	5
O21	4	5	7	-
O22	5	-	6	3
O23	-	5	4	7
O31	5	3	-	6
O32	-	-	4	-
O41	2	4	5	-
O42	-	4	2	-
O43	5	4	6	3

The first subscript of the operation means the Job number and the second the order of the operations within a job.

The food source would be structured as the figure 3 below:

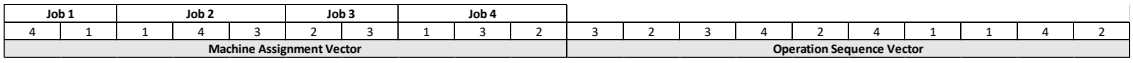

Figure 2 - Food Source Structure

This structure means that O_{11} would be scheduled to the machine 4, the O_{12} would be scheduled to the machine 1 and so on. Moreover, it also means that the order of the scheduling would be $\{O_{31}, O_{21}, O_{32}, O_{41}, O_{22}, O_{11}, O_{12}, O_{42}, O_{23}\}$.

The quality of these food sources will be quantified as the makespan of the scheduling solution, explained in section 1.2.4 of this document. Only one bee will be allocated to each food source.

3.2.1 Initialization Phase

The initialization phase is the first phase of the algorithm. It comprises the initialization of the parameters and of the food sources (scheduling solutions).

For the machine assignment vector, three dispatching rules were implemented, being one chosen for each food source, given a predefined probability for each to happen:

- Random rule - chooses a random machine for each operation

```
RandomMachineDispatchingRule(problem){
    Solution={}
    For(operation in problem.Operations){
     Machine=random.choice(problem.machines)
     Solution.Add(operation,machine)
    }
    Return Solution
}
```

- Minimum processing time rule (MPT) – The minimum processing time for each operation is found and the machine that minimizes this time has the operation assigned to it

```
MinimumProcessingTimeDispatchingRule(problem){
    Solution={}
    For(operation in problem.Operations){
     Machine=getMachineWithMinimumProcessingTime(operation.problem.machines)
     Solution.Add(operation,machine)
    }
    Return Solution
}
```

- Global minimum processing time rule – The minimum processing time for each machine is used to assign the operations to the machines.

```
  GlobalMinimumProcessingTime(problem){
    Solution={}
    For(machine in problem.machines){
machineOPerationAssignment=getMinimumProcessingTimeOperations(machine,problem
)
    Solution.Add(machineOperationAssignment,machine)
    }
    Return Solution
}
```

Regarding the operation sequence vector initialization, the same dispatching rules applied in the genetic algorithm initialization are used:

- Random Dispatching Rule – The Jobs and operations are organized randomly, respecting the sequence of operations

•

```
RandomDispatchingRule(Jobs){
    Cromossome c = new Cromossome();
           UnorderedTaskVector unordered = GetUnorderedTaskVector()
    OrderedVector = OrderJobs(unordered)
    While( len(OrderedVector)>0){
      Job=Random.choice(OrderedVector)
      Operation = Job.nextOperation();
      Machine m = GetRandomFeasibleMachine(Job,Operation)
      Cromossome.append(task.JobId, task.OperationI, m.MachineId)
      OrderedVector.pop(Job,Operation)
    If(c.IsValidCromossome()){
      Return c
    }
}
```

- Most Work Remaining (MWR) – The jobs and operations are organized by the Jobs that have more workload to be done

```
MostWorkRemainingDispatchingRule(Jobs){
    Cromossome c = new Cromossome();
           UnorderedTaskVector unordered = GetUnorderedTaskVector()
    OrderedVector = OrderJobs(unordered)
    While( len(OrderedVector)){
      Job=GetMostWorkJob(OrderedVector)
      Operation = Job.nextOperation();
      Machine m = GetRandomFeasibleMachine(Job,Operation)
      Cromossome.append(task.JobId, task.OperationI, m.MachineId)
      OrderedVector.Pop(Job,Operation)
    If(c.IsValidCromossome()){
      Return c
    }
}
```

- Most number of operations remaining (MOR) – The jobs and operations are organized by the greatest number of operations remaining in the job.

```
MostNumberOfOperationsRemainingDispatchingRule(Jobs){
    Cromossome c = new Cromossome();
    UnorderedTaskVector unordered = GetUnorderedTaskVector()
    OrderedVector = OrderJobs(unordered)
    While( len(OrderedVector)){
      Job=GetMostOperationsJob(OrderedVector)
      Operation = Job.nextOperation();
      Machine m = GetRandomFeasibleMachine(Job,Operation)
      Cromossome.append(task.JobId, task.OperationI, m.MachineId)
      OrderedVector.Pop(Job,Operation)
    If(c.IsValidCromossome()){
      Return c
    }
}
```

3.2.2 Employed Bees Phase

This phase comprises the crossover and mutation operations.

For the machine assignment vector two methods of crossover were implemented, having a probability that can be defined in the algorithm's hyper-parameters: two-point crossover and uniform crossover.

The two-point crossover chooses two positions in each of the two parents. Then, each children keeps all the values but the ones between the two points, which come from the other parent. Example is provided in the Figure 3 below.

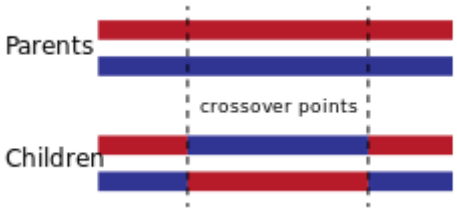

Figure 3 - Two-Point Crossover Example [59]

```
TwoPointCrossOver(Parent1,Parent2){
 Child1={}
 Child2={}
 For(i in range(1:2){
  randomNumber = randInt(len(Parent1))
  Child_1=Child_1.append(Parent1[:i],Parent2[i:])
  Child_2=Child_2.append(Parent2[:i],Parent1[i:])
 }
 Return {Child_1,Child_2}
}
```

The uniform crossover comprises the idea of picking positions from each the parents. For each of the parents, different positions will be chosen, providing those positions to be passed to the children. Figure 5 provides an example of this crossover.

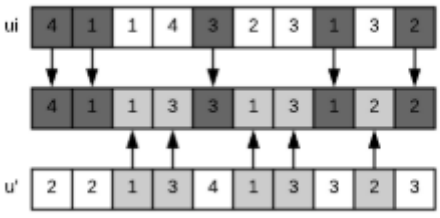

Figure 4 – Uniform Crossover Example [59]

```
UniformCrossover(Parent1,Parent2){
  Child1=Parent1
 Child2=Parent2
 For(int i in len(Parent1){
   P=randInt(0:1)
  If(P<0.5){
    Temp=Child1[i]
   Child1[i]=Child2[i]
   Child2[i]=Temp
  }
 }´
 Return {Child1,Child2}
}
```

For the operation sequence vector, the crossover method used in this implementation is the Preservingá Order-Based Crossover (POX) explained in the genetic algorithm implementation.

```
PreservingOrderBasedCrossover(parent1,parent2,problem){
 operation_selected = random.choice(problem.jobs)
 positions = getPositionAndOperationOfRelatedOperations(selected,parent1)
 child1 = getPositionAndOperationOfUnrelatedOperations(selected,parent2)
 for(position in positions){
  child1.insertAt(position.index,position.operation)
 }
 operation_selected2 = random.choice(problem.jobs)
 positions2 = getPositionAndOperationOfRelatedOperations(selected2,parent2)
 child2 = getPositionAndOperationOfUnrelatedOperations(selected2,parent1)
 for(position in positions2){
  child2.insertAt(position.index,position.operation)
 }
return {child1,child2}
```

A mutation operation for the machine assignment vector was also implemented. It is a simple random mutation where two machines are changed randomly within the vector.

```
RandomMutation(individual,problem){
 operationToMutate=random.choice(individual)
 While(True){
  newMachine=randInt(0:problem.NumberOfMachines)
  if(newMachine!=operationToMutate.MachineNr){
    operationToMutate.SetMachine(newMachine)
 }
}
```

3.2.3 Onlooker Bees Phase

The onlooker bees phase comprises the comparison of several food sources, choosing the best one. The best one, as stated before, is considered the one with the smaller makespan. Thus, new food sources will only be accepted if the makespan of the new food source is smaller than the previous best makespan result.

This phase also saves the best food source after every iteration of the process.

```
fitnessFunction(individual,problema){
 time=[]
 for(machine in individual.machines){
  time[machine]=0
  for(operation in individual.getOperationsOnMachine(machine)){
   time[machine]+=individual.getProcessedTime(operation)
  }
 }
 Return max(time)

Min(fitnessFunction(individual,problem)
```

3.2.4 Scout Bees Phase

The scout bees phase comprises the creation of new food sources, based on the initialization processes explained in subsection 4.2.1. After the creation of the food source, if the new food source has a better quality than the worst employed bee's food source quality, we replace this employed bee with the scout bee's new food source.

3.3 Dynamic Scheduling

This subsection contains the methods the algorithm must deal with dynamic scheduling. As stated before, problems happen in factories. Not always machines are available to produce, even though it was planned. depending on the type and model, machines can have several defects that make it inoperable. Moreover, clients of a factory might have issues themselves. Clients might cancel orders or part of them, making so that the schedule is outdated and that the number of products planned to produce is bigger than the demand from clients. It is possible to distinguish two types of different disruptions from this scenario: machine and job/operation disruptions.

On the other hand, there should be a concern with the timing of the disruptions. While with machine disruptions it is comprehensible that they happen while the schedule is already put in practice, with job or operation disruptions, there might be more variability with timings. A client might cancel/change an order before it is already being produced or it can do it while it is already in production.

In order to solve these problems that a static scheduling system cannot solve, heuristics were created for the initialization phase, in order to transform the static scheduling system in a dynamic one, using the event driven rescheduling technique.

3.3.1 Early Job/Operation Disruption

This subsection presents the heuristics for the early job or operation cancelation. Although jobs and operations are different entities (a job is a set of different operations), these disruptions can be dealt with using the same technique.

As the disruption is an early disruption, a complete generation reschedule can be done. This means that the schedule can be completely changed without consequences for the state of the production. Thus, when this type of disruption happens, we can remove or add the operations/jobs from the machine assignment and operation order vectors and let the static scheduler run with the new data. Figure 5 shows the workflow of this heuristic.

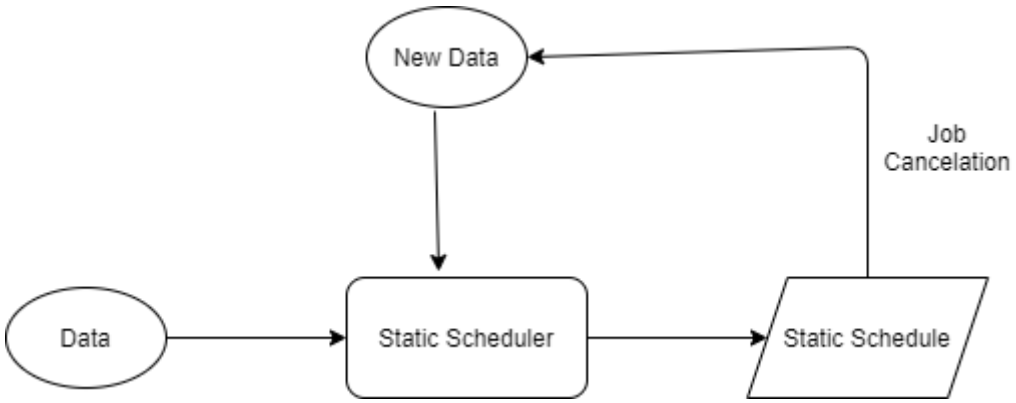

Figure 5 – Early Job Cancelation Heuristic

This will provide a possibly completely different schedule allowing for the best possible optimization as there are no ties with the state of production.

3.3.2 Early Machine Disruption

The possible early addition/breakdown of a machine is the disruption studied in this subsection. Once again, we are dealing with an early disruption, so it is possible to use a complete generation rescheduling technique. Thus, it is possible to remove the machine from the problem data and run the static algorithm again for the new problem. Figure 6 represents the workflow of this heuristic.

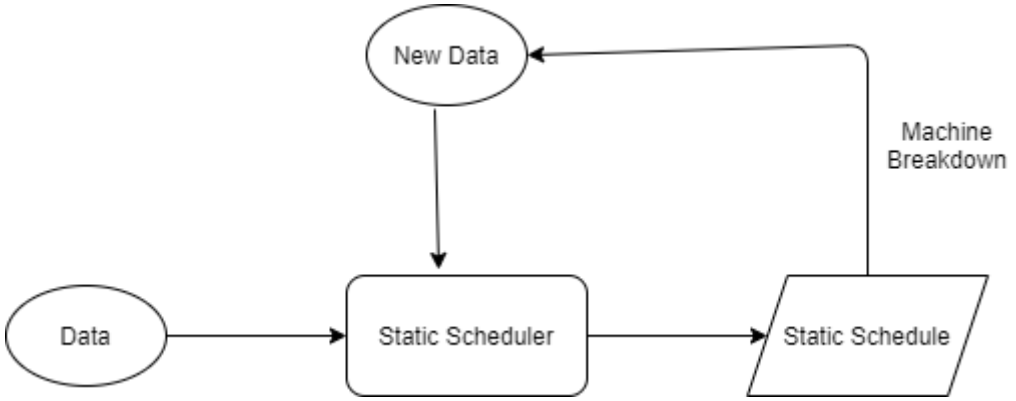

Figure 6 – Early Machine Breakdown Heuristic

This provides a way to make the best possible optimization of the problem.

3.3.3 Late Job/Operation Disruption

This subsection solves the issue of late cancelation/addition of a job or operation on the schedule. This time, as production is already using the schedule, completely changing it not possible. Thus, after acquiring the time of cancelation, the data from the jobs/operations that have already ran/ are already running is removed from the problem. The static scheduler will use the remaining data to schedule the rest of the time that has not yet passed. This technique is called schedule repair. Figure 7 represents the workflow of this heuristic

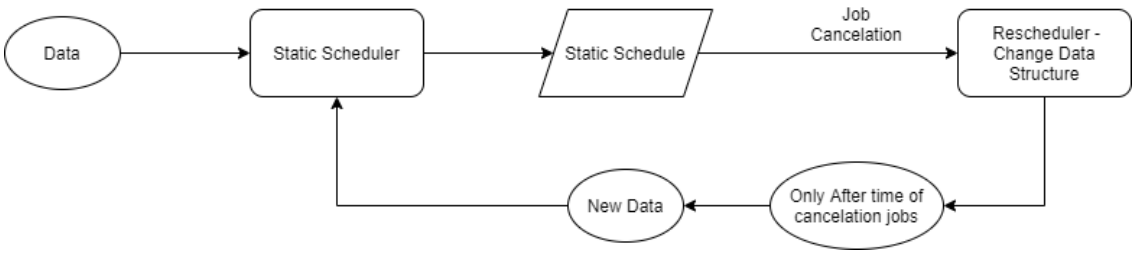

Figure 7 – Late Operation Cancelation Heuristic

3.3.4 Late Machine Disruption

This subsection presents the heuristics for the late machine addition/removal. It is similar to the late job/operation disruption presented above. Given a disruption time, the schedule will have its data changed in order to affect that disruption (if a new machine appears, a new machine is added to the data). The jobs and operations that have already ran will be removed from the initial data and the static scheduler will run the rest of the information, providing a new partial schedule for the remaining of the time. Figure 8 represents the workflow of this heuristic

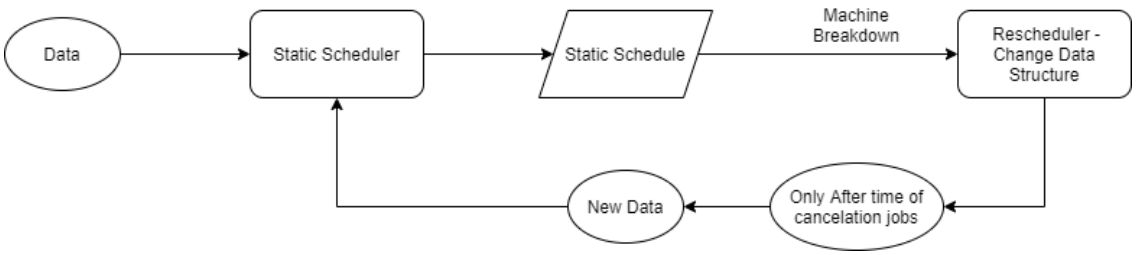

Figure 8 – Late Machine Breakdown Heuristic

4 Results

This section contains results on the two algorithms, GA and ABC, comparing them according to the test methodology explained in Section 1.4.3 of this document. To do so, the two developed algorithms will be compared against each other using the same data, to see which has better capabilities on scheduling known problems. Furthermore, the best algorithm of the two will be compared to other benchmark solutions that use the same metric of evaluation of solutions (the makespan), to prove its viability among state-of-the-art solutions. At last, the rescheduling problems were tested for this algorithm.

4.1 Test Data

In order to compare the developed algorithms, the Brandimarte dataset [60] was used. The Brandimarte instances have a range of 10 to 20 jobs, 4 to 15 machines, 3 until 15 operations per job 1 to 20 units of time for the processing time. M_{ik} is the maximum number of assignable machines per operation, meaning not all the machines can perform all the operations. Table 4 provides insights on the number of jobs, number of machines, operations per job, processing time and maximum number of assignable machines.

Table 4 - Information from the Brandimarte dataset instances [51]

Instances	Job	Machines	Operations per Job	Processing time	$\lvert M_{ik} \rvert$
Mk01	10	6	5 to 7	1 to 7	3
Mk02	10	6	5 to 7	1 to 7	6
Mk03	15	8	10	1 to 20	5
Mk04	15	8	3 to 10	1 to 20	3
Mk05	15	4	3 to 10	1 to 10	2
Mk06	10	15	15	1 to 10	5
Mk07	20	5	5	1 to 20	5
Mk08	20	10	5 to 15	5 to 20	2
Mk09	20	10	10 to 15	5 to 20	5
Mk10	20	15	10 to 15	5 to 20	5

The instances are structed in a file each. The file has at least 2 numbers in the first line: the first is the number of jobs and the second the number of machines (the third is not necessary, it is the average number of machines per operation). After that line, every line represents a job. The first number of the line is the number of operations of that job. The second number, always bigger than 1 is the number of machines that can process the first operation. Then, each pair of numbers (machine, processing time) represent a possible machine to process that job and the time taken to process it in that machine. Figure 6 represents the data for the mk04 instance.

```
15  8   2
8  1 1 6 2 1 6 7 9 2 6 7 3 1 2 4 2 7 5 3 1 8 3 9 8 9 3 2 3 4 8 3 2 2 5 5 6 7 2 6 1 4 7
7  1 6 1 2 6 1 4 7 1 1 6 2 6 7 3 1 3 2 3 4 8 3 2 1 6 2 1 7 2
6  1 6 1 3 2 3 4 8 3 2 3 3 2 7 1 4 4 2 4 2 7 5 2 1 7 3 7 2 4 4 3 1
5  1 7 2 1 1 6 2 1 6 7 9 2 6 7 3 1 2 4 5 5 7
7  1 7 2 2 1 6 7 9 2 4 4 3 1 3 1 8 3 9 8 9 2 1 7 3 7 3 2 3 4 8 3 2 2 4 5 5 7
9  1 6 2 2 4 4 3 1 3 3 2 7 1 4 4 2 6 1 4 7 2 4 5 5 7 3 1 8 3 9 8 9 2 1 7 3 7 1 6 1 2 1 6 7 9
5  2 5 5 6 7 2 1 7 3 7 2 6 1 4 7 1 6 2 2 6 7 3 1
6  2 4 5 5 7 2 5 5 6 7 3 2 3 4 8 3 2 1 6 2 1 6 1 2 1 6 7 9
9  1 1 6 2 1 6 7 9 2 4 4 3 1 3 1 8 3 9 8 9 2 4 2 7 5 2 6 1 4 7 1 7 2 2 1 7 3 7 3 2 3 4 8 3 2
5  2 5 5 6 7 1 1 6 1 7 2 2 4 5 5 7 2 1 6 7 9
4  3 1 8 3 9 8 9 1 1 6 3 2 3 4 8 3 2 2 4 2 7 5
6  2 4 2 7 5 1 6 1 1 1 6 2 1 7 3 7 3 1 8 3 9 8 9 1 7 2
4  1 6 2 2 6 7 3 1 2 6 1 4 7 2 5 5 6 7
3  2 5 5 6 7 1 6 1 2 4 2 7 5
6  2 4 5 5 7 1 7 2 3 1 8 3 9 8 9 3 2 3 4 8 3 2 3 3 2 7 1 4 4 1 1 6
```

Figure 9 – Brandimarte MK04 Instance Data [57]

This instance has 15 jobs and 8 machines that they can be processed in. It has an average number of machines per operation of 2. To exemplify this structure, the first job, second line, will be used. Figure 7 shows that line in detail.

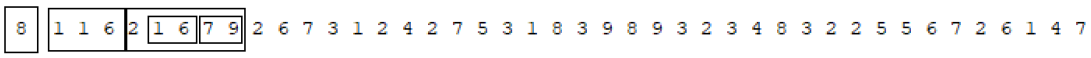

Figure 10 – First job of the MK04 instance

The first number shows that there refers to 8 operations in this job. The first operation can be done in only one machine (machine 1) and takes 6 units of time to do it. The second operation can be done in 2 machines: machine 1 and 7, taking 6 units of time in the first machine and 9 units on the seventh.

4.1.1 Genetic Algorithm

The genetic algorithm proposed was implemented to solve the above-mentioned specific cases. The algorithm was run five times for each case, given different hyper-parameters. These parameters are presented in table 5. The results presented below are the best results between the five runs.

Table 5 – Hyper-parameterization for the Genetic Algorithm

Population Size	Generations	Initialization Parameters	Crossover Parameters	Mutation Parameters
4000	800	20% Random; 40% MWR; 40% MOR	55% POX	2% PPS 2% Assignment 6% Intelligent
5000	1000	20% Random; 40% MWR; 40% MOR	55% POX	2% PPS 2% Assignment 6% Intelligent
5000	1000	10% Random; 45% MWR; 45% MOR	55% POX	20% PPS 20% Assignment 40% Intelligent
4000	800	10% Random; 45% MWR; 45% MOR	55% POX	4% PPS 4% Assignment 12% Intelligent
5000	1000	10% Random; 45% MWR; 45% MOR	55% POX	2% PPS 2% Assignment 6% Intelligent

The best results came from the algorithm configured to have a population size of 5000 individuals and 1000 generations. It had the initial population generation hyper-parameterized

with 20% probability to generate the individual using the random dispatching rule, 40% to generate using the most workload remaining rule and 40% to generate using the greatest number of operations remaining rule. Regarding crossover, it was configured to have a 55% probability of doing the Precedence Preserving Order-based crossover instead of the assignment crossover, which has a probability of 45%. Furthermore, the PPS mutation was set to have a 2% probability of occurring, the assignment mutation set to 2% mutation and the intelligent mutation to 6% probability.

The results for every instance are represented in the table 6 below for the best result. The first column represents the instances of the Brandimarte dataset. The second the best result from the proposed genetic algorithm. The third one contains the best-known lower bound, found in [61].

Table 6 - Comparison from the Genetic Algorithm Solution and the Best-Known Lower Bound

Instances	Best Makespan Result	Best – Known Lower Bound
MK01	40	36
MK02	26	24
MK03	204	204
MK04	60	48
MK05	173	168
MK06	63	33
MK07	139	123
MK08	523	523
MK09	311	299

The results were promising, as the values did not deviate a lot from the best known lower bound.

4.1.2 Artificial Bee Colony

The artificial bee colony algorithm was implemented to solve the same cases as the genetic algorithm. For cohesion purposes, this algorithm run the same number of times as the previous one (5), with different parameterizations. Table 7 shows the different parameterizations used.

For the algorithm configuration, consider *n* the number of jobs and *m* the number of machines.

Table 7 – Hyper-parameterization of the ABC algorithm

Generations	Generation Termination Criteria	Number Of Bees	Initialization Parameters	Crossover Parameters	Mutation Parameters
3*n*m	1.5*n*m	5*n Emp 13*n Onl 0.4*n Sco	20% Random; 40% MPT; 40% GMPT	50% Each For Machine 100% POX for operation sequencing	90%
2*n*m	1.5*n*m	3*n Emp 11*n Onl 0.2*n Sco	20% Random; 40% MPT; 40% GMPT	50% Each For Machine 100% POX for operation sequencing	90%
2*n*m	1.5*n*m	3*n Emp 11*n Onl 0.2*n Sco	10% Random; 45% MPT; 45% GMPT	50% Each For Machine 100% POX for operation sequencing	5%
3*n*m	1.5*n*m	5*n Emp 13*n Onl 0.4*n Sco	10% Random; 45% MPT; 45% GMPT	50% Each For Machine 100% POX for operation sequencing	5%
2*n*m	1.5*n*m	3*n Emp 11*n Onl 0.2*n Sco	20% Random; 40% MPT; 40% GMPT	50% Each For Machine 100% POX for operation sequencing	5%

The best algorithm was the one configured to have *2*n *m* generations. The number of generations for the algorithm to terminate if there are no better solutions is *1.5*n*m*. Regarding the number of bees for each of the three types (employed, onlooker and scout), *3*n*, *11*n* and *0.2*n* were set respectively. As for the initial population, the random rule has a probability of 20% of happening, the global minimum processing time rule has a 40% probability of happening and the local minimum processing time rule as a probability of 40% as well. Regarding the operation sequence vector initialization, the same probabilities are applied, 20% for the random rule and 40% for the other two rules. On the crossover parameters for machine

assignment, the two methods (two-point crossover and random) have a 50% probability each. For operation sequencing, POX is used 100% of the time. At last, the mutation probability was set to 90%.

The results of the implementation are represented in table 8.

Table 8 - Comparison from the Artificial Bee Colony Algorithm and the Best-Known Lower Bound

Instances	Best Makespan Result	Best – Known Lower Bound
MK01	39	36
MK02	26	24
MK03	204	204
MK04	60	48
MK05	169	168
MK06	58	33
MK07	140	123
MK08	523	523
MK09	308	299

This algorithm also provides interesting results, reason why both algorithms should be compared, in order to check which of them is providing better results.

4.1.3 Comparison

To compare these two algorithms effectively, the mean improvement of all the instances between the two best algorithms' results will be used. The table 9 below presents the results for each of the algorithms, as well as the improvement between the two.

Table 9 - Comparison between the Genetic Algorithm and the Artificial Bee Colony algorithm and its quality

Instances	Genetic Algorithm	Artificial Bee Colony	Improvement
MK01	40	39	2,5
MK02	26	26	0
MK03	204	204	0
MK04	60	60	0
MK05	173	169	2.3
MK06	63	58	7.9
MK07	139	140	-0.7
MK08	523	523	0
MK09	311	308	0.9

The mean improvement of the artificial bee colony algorithm regarding the genetic algorithm is 1.43%, which means the first one is 1.43% better than the other. Considering that only in one

of the cases the genetic algorithm was better than the artificial bee colony, and that the difference was small (an improvement of 0.7 % from the GA to the ABC), it is safe to say that the ABC algorithm developed has a better performance than the genetic algorithm.

4.2 Benchmark Data

This section provides the comparison between some state-of-the-art benchmarks and the best algorithm between the two developed: the artificial bee colony algorithm.

The performance will be accessed by comparison with six algorithms using different techniques. The six algorithms are Hybrid Genetic Algorithm (hGAJobS), Learning and Evolving Genetic Architecture (LEGA), Parallel Variable Neighbourhood Search (PVNS), Knowledge- Based Ant Colony Optimization (KBACO), Tabu Search Algorithm with a fast Public Critical Block neighbourhood structure (TSPCB) and Artificial Bee Colony (ABC). The choice to compare with different algorithms is motivated by the fact that we want to know if this algorithm is viable among the solutions in the market, and not that it is better among the ones using the same method. Further explanation on the algorithms is given in the bullet points below:

- hGAJobS [62]- A hybrid genetic algorithm with local search optimization
- LEGA [63]- The genetic algorithm provides an effective integration between evolution and learning within a random search process
- PVNS [64]- The algorithm has a parallelization idea based on applying multiple independent searches increasing the exploration in the search space
- KBACO [65] - The hybrid algorithm provides an integration of the Ant Colony Optimization (ACO) and the knowledge mode;
- TSPCB [66] - This hybrid algorithm is based on three steps. In the first step, four machine assignment rules and four operation scheduling rules are developed to find a good set of initial solutions. In the second step, an effective neighbourhood structure to conduct the local search in the machine assignment module is proposed. And in the last step, based on public critical block theory, a speedup local search method with three kinds of insert and a swap neighbourhood structure is applied
- ABC [67] - The first implementation of the ABC algorithm to solve the FJSSP.

The results are presented in table 10.

Table 10 - Comparison between the proposed ABC and the benchmark algorithms

Instances	hGAJobS	LEGA	PVNS	KBACO	TSPCB	ABC	Proposed ABC	Best – Known Lower Bound
MK01	40	40	40	39	40	40	39	36
MK02	27	29	26	29	26	26	26	24
MK03	204		204	204	204	204	204	204
MK04	60	67	60	65	62	60	60	48
MK05	173	176	173	173	172	172	169	168
MK06	64	67	67	67	65	60	58	33
MK07	141	147	144	144	140	139	140	123
MK08	523	523	523	523	523	523	523	523
MK09	312	320	307	311	310	307	308	299

The results show that the proposed algorithm has a near state-of-the-art results. The variability on its operations is the main differentiator to the other state of the art algorithms, thus being the reason of the different results. Even though there are some of them that have better results than the ones found on the proposed algorithms, most of them have similar or worst results in the instances.

4.3 Rescheduling Results

This subsection showcases the re-scheduling capabilities of the proposed best algorithm. The reschedule works by picking the static scheduling, treating the input data to have the disruption removed (if it is a Job removed, remove the job, if it is a machine broken, remove the machine and put the operations on the other machines). The results of the algorithm are compared with the results from *Honghong et Al*. in "*The application of Adaptative genetic algorithms in FMS dynamic rescheduling*"[68].

4.3.1 The benchmark data

The benchmark data used in this test is, as stated, the one from "*The application of Adaptative genetic algorithms in FMS dynamic rescheduling*"[68]. The scheduling problem has 6 machines and 13 jobs. For each job the number of the operations is 3, 2, 3, 4, 3, 3, 2, 3, 2, 3, 3, 3, 3, respectively. There is a total of 37 operations. Table 11 hows the data from this article.

Table 11 - Benchmark data [59]

	M1	M2	M3	M4	M5	M6
J1	8	8	6	6	6	6
J2	5	5	9	0	0	0
J3	4	4	0	6	6	6
J4	5	5	0	8	8	8
J5	0	0	0	5	5	5
J6	10	10	6	9	9	9
J7	0	0	0	6	6	6
J8	12	12	8	8	8	8
J9	5	5	0	8	8	8
J10	6	6	0	7	7	7
J11	9	9	11	12	12	12
J12	7	7	0	8	8	8
J13	9	9	13	5	5	5

The original static schedule solution from the articles' algorithm is presented in the figure 9 below.

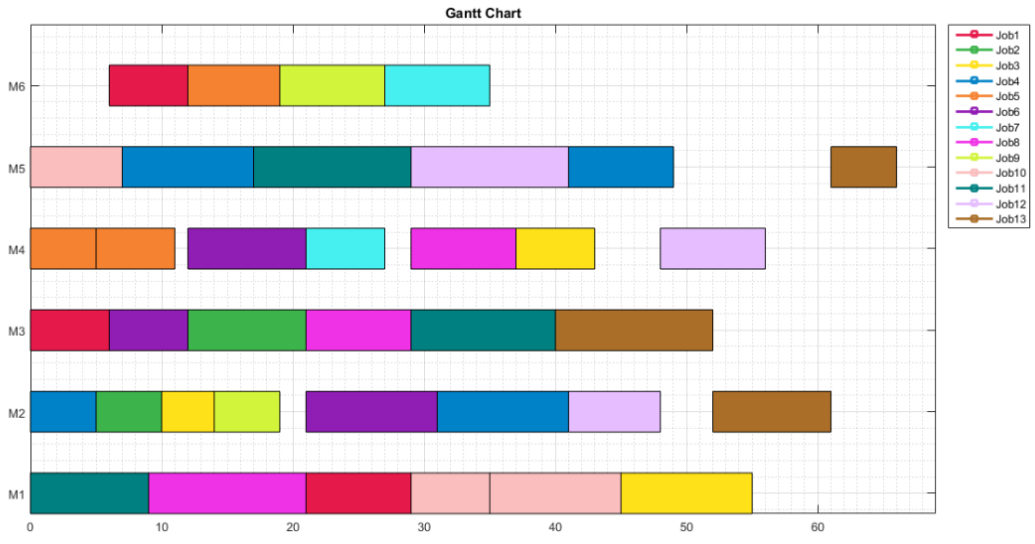

Figure 11 – Static schedule solution from the benchmark algorithm for the data

4.3.2 Comparing the algorithms for Early Job Cancelation

The algorithms were compared against the same problem explained above. To make this problem dynamic, Job 11 is going to be cancelled. The Makespan of the schedule before job cancelation is 66. The results showed in the article demonstrate a Makespan after cancelation of 62 (obviously smaller than without cancelation because there were less jobs to schedule). After five runs of the proposed algorithm, the best solution found had a makespan of 50, 12 units of time smaller than the benchmark's algorithm. This translates to a 19.4% improvement when compared to the benchmark algorithm. Figure 10 shows the result of the scheduler and table 12 shows the metrics of evaluation

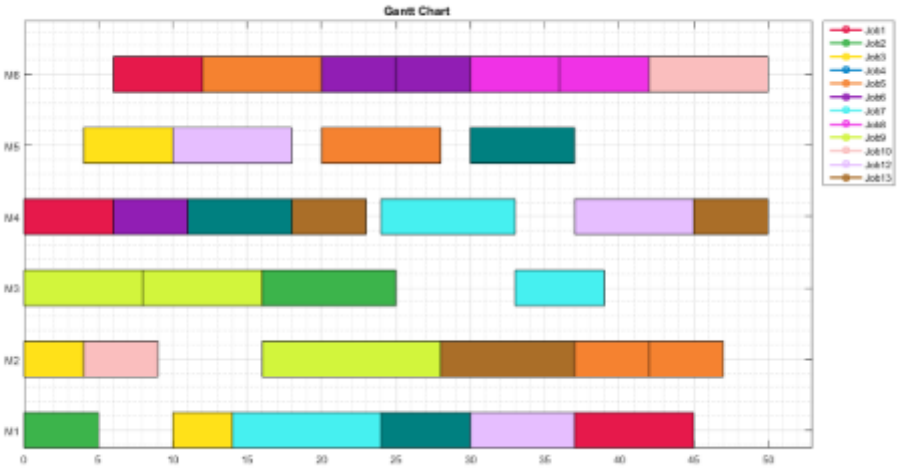

Figure 12 – Gant Chart for Early Job Cancelation

Table 12 - Results for Early Job Cancelation for the developed algorithm

Best Makespan	50
Improvement	19.4%

4.3.3 Comparing the Algorithms for Late Job Cancelation

The algorithms were compared against the same problem explained above. This time, as we are dealing with a late cancelation, a time of cancelation must be chosen. 8 units of time is the time presented in the benchmark algorithm solution and, therefore, that is the time that was used in the developed solution as well. Job 11 was the Job cancelled. The static schedule from the benchmark had a makespan of 66 and after the cancelation it turned to 62. The provided solution gave a makespan after cancelation of 55, 7 units of time better than the solution. This translates to a 11.29% improvement. Figure 11 represents the Gant Chart for the solution and Table 13 presents the metrics for the solution.

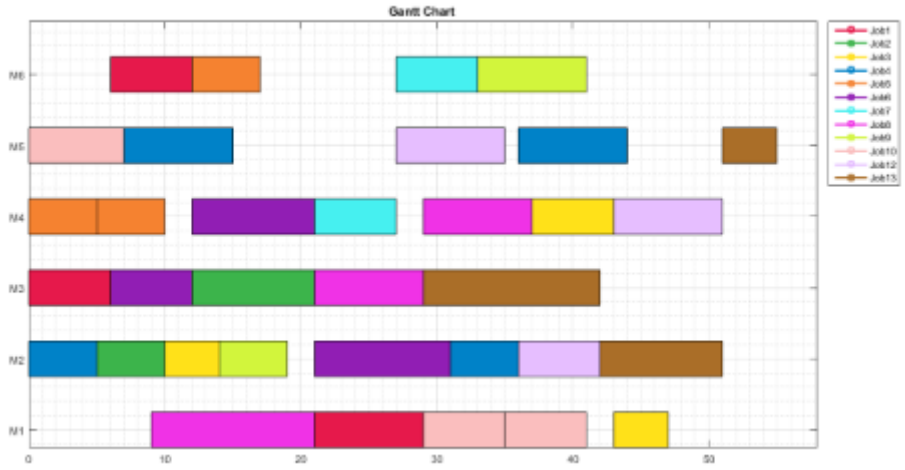

Figure 13 – Gant Chart for Late Job Cancelation

Table 13 - Results for Late Job Cancelation

Best Makespan	55
Improvement	11.29%

4.3.4 Comparing the Algorithms for Early Machine Breakdown

This subsection treats of the comparison between the results of the proposed algorithm and the benchmark one for late machine breakdown. Machine 6 was the machine that was chosen to remove from the schedule. For this problem, the benchmark solution provided a makespan of 70, while the proposed solution provided a result of 67. Therefore, there was a small improvement of 4.29%. Table 14 presents the results for this algorithm

Table 14 - Results for Early Machine Breakdown

Best Makespan	67
Improvement	4.29%

4.4 Conclusion Of The Results

The results show really good scores compared to the data and algorithm of the benchmark article. Nevertheless, this is nothing but a good indicator that the algorithm can provide state-of-the-art results. The results are for a really specific scenario that could be nearer to the actual necessities of a factory than it is. More tests need to be done against other algorithms and data to prove that the algorithm is actually as good as it seems to be.

Despite this last consideration, the algorithm shows really good results against this one benchmark article's algorithm.

5 Conclusions

The purpose of this thesis was to develop a good performance dynamic scheduling system, that would be capable of near real-time rescheduling of the jobs on a factory. To do so, a study was conducted on the state-of-the-art algorithms to solve optimization problems. Considerations were made on which algorithm would be better to solve the problem. Two algorithms were developed: a genetic algorithm and an artificial bee colony algorithm. In a first instance, the development passed through implementing static algorithms, disregarding the possible changes when the production is happening. Both were successfully implemented and had promising results. After having this conclusion, heuristics were developed to deal with dynamic events that could happen in a factory.

In regard to the first part of the development, the python language was used. The algorithms use a combination of rules to generate the initial population, preventing the generation of infeasible solutions. After testing the algorithms between them using the same data from the Brandimarte dataset, the artificial bee colony was found to be 1.43% better than the genetic algorithm solution. Furthermore, this higher quality algorithm was compared to other state of the art solutions, having as good or better results in most of the test instances. It is probable that the algorithm would provide even better solutions with fine-tuning in its hyperparameters.

Notwithstanding, this thesis primary goal was to produce good performance on dynamic environments and not in static ones. The algorithm provides heuristics to deal with two different types of disruptions: job disruptions and machine disruptions. It is capable of re-scheduling on addition of new jobs and cancelation of a job. It is also possible to only cancel part of the operations of a job. Furthermore, it has the capability of re-scheduling after a machine was removed or when a new machine is added. The scenarios were tested against a benchmark article solution. The results were also promising. The algorithms provided makespans as good or better than the ones from the benchmark solution.

All in all, two static scheduling algorithms were developed and heuristics capable of treating 6 types of dynamic events were implemented. All of them showcased good results, being good options to implement in a factory. Factories that make products to stock should probably consider the static solution, as there are less disruptions in a factory like that. On the other hand, factories that have a more make to order approach should consider the dynamic adaptation of the algorithm.

5.1 Future Work

Factories are a changing environment, and, with the implementation of Industry 4.0, the changes are faster than ever. So, it is important to consider that the Makespan is only one of the possible objective functions and possibly not the best. Testing with other possibilities is something that should be addressed in the future. Moreover, the possibility of combining different objective functions in a more complex one should be considered. This has a relation with an even wider formulation of the JSSP, the multi-objective flexible job shop scheduling problem (MOFJSSP). Besides the classic objective functions, considering things like the material preparation time could also be beneficial to factories, even though it could decrease the quality of the algorithm.

Furthermore, with the current World situation (war in Russia and Covid-19 pandemic), some factors that were not so problematic in the past are starting to be.

Firstly, the increase of the costs of energy is starting to decrease the margins that a factory can make producing. These costs should be considered in a scheduling algorithm, as they might impact the way a company wants to organize its production in the factories. It is possible to consider these energetical costs, experimenting with the time window of it throughout the generations of the algorithm, trying to improve these costs as well.

Secondly, considering the lack of materials that is happening nowadays (mostly on the semiconductor industry), it is possible that a factory, at a certain point of a schedule, runs out of materials to produce all of the orders they have. The re-scheduling algorithm could be altered to have in consideration the practical and costly implications of removing each job/operation, considering the cost in its objective function.

At last, the impact of the maintenance of a machine in a factory should be considered when scheduling the production. With the help of a Remaining Useful Life predictor for a machine, it is possible to plan the schedule considering timeframes where each machine is going to be down for maintenance. Having these times in consideration makes the static scheduling system more probable of being feasible in reality, without the need to re-schedule due to machine maintenance.

In conclusion, testing with as many real-life constraints as possible should provide a better, more feasible schedule that can practically be implemented in a factory and not only used as a guideline that needs to be rescheduled many times.

5.2 Final Remarks

Solutions like the proposed in this MSc dissertation can revolutionize how a factory works but there are also downsides to it. There is the possibility of this project being integrated in a cyber-physical system. That can be done by adding it as a functionality in a manufacturing execution system (MES) or enterprise resource planner (ERP). In these types of systems, it is possible for actions to be taken by machines itself without the supervision of humans. The best opportunities of these systems on factories appear when even some operators can be replaced by robots. When using the project developed in this manner, it is possible for hackers to change the result of the scheduling in ways that can harm the company financially, changing the order of the production orders so that it is non-lucrative for the company. There should be a concern in protecting the system this solution is built in.

Furthermore, the scheduling done using this solution does not consider the availability of operators for the machines. It must be considered that it can provide a solution that is impossible considering the human capital existent on a factory. It might provide a solution that makes an operator work more hours than it is supposed. Also, the fact that this algorithm can do the job a person was doing is also an issue. Moreover, it is important to consider that a person is going to be replaced by the scheduling system. It is important to consider what is the future of that person. The suggestion given is to train the person to use this system and provide a way for the person to continue doing something similar but with its help.

Bibliography

[1] "The 4 Industrial Revolutions - Institute of Entrepreneurship Development." https://ied.eu/project-updates/the-4-industrial-revolutions/ (accessed Dec. 27, 2021).

[2] "Here's what the 2021 global semiconductor shortage is all about." https://techwireasia.com/2021/10/heres-what-the-2021-global-semiconductor-shortage-is-all-about/ (accessed Dec. 27, 2021).

[3] "Production Management : it's Meaning, Definition, Function and Scope." https://www.yourarticlelibrary.com/production-management/production-management-its-meaning-definition-function-and-scope/27925 (accessed Nov. 29, 2021).

[4] P. Ávila, J. Bastos, and I. Cavaco, *Planeamento e Controlo da Produção - Uma visão integrada* . 2022.

[5] I. Carolina Azevedo Ferreira, J. Miguel da Costa Sousa Susana Margarida da Silva Vieira, C. Baptista Cardeira Supervisor, and S. Margarida da Silva Vieira, "Dynamic Scheduling using Artificial Bee Colony Algorithm Mechanical Engineering Examination Committee," Insituto Superior Tecnico, 2019.

[6] C. Shyalika, "Job Shop Scheduling Problem (JSSP): An Overview." https://medium.datadriveninvestor.com/job-shop-scheduling-problem-jssp-an-overview-cd99970a02f8 (accessed Jan. 02, 2022).

[7] L. Özdamar and G. Ulusoy, "A survey on the resource-constrained project scheduling problem," *IIE Transactions (Institute of Industrial Engineers)*, vol. 27, no. 5, pp. 574–586, 1995, doi: 10.1080/07408179508936773.

[8] "What is Python? Executive Summary | Python.org." https://www.python.org/doc/essays/blurb/ (accessed Jan. 15, 2022).

[9] "Trello." https://trello.com/en (accessed Jun. 05, 2022).

[10] "What is Scrum?" https://www.scrum.org/resources/what-is-scrum (accessed Jun. 05, 2022).

[11] "Gantt chart - Wikipedia." https://en.wikipedia.org/wiki/Gantt_chart (accessed Jul. 03, 2022).

[12] T. M. Hassan, H. A. Bassioni, A. Fahmy, and H. Bassioni, "What is Dynamic Scheduling?," 2014. [Online]. Available: https://www.researchgate.net/publication/267208350

[13] I. Sabuncuoglu and M. Bayõz, “Analysis of reactive scheduling problems in a job shop environment.” [Online]. Available: www.elsevier.com/locate/dsw

[14] L. K. CHURCH and R. UZSOY, “Analysis of periodic and event-driven rescheduling policies in dynamic shops,” *International Journal of Computer Integrated Manufacturing*, vol. 5, no. 3, pp. 153–163, May 1992, doi: 10.1080/09511929208944524.

[15] C. R. Reeves, *Modern Heuristic Techniques for Combinatorial Problems*. USA: John Wiley & Sons, Inc., 1993.

[16] R. Haupt, “A survey of priority rule-based scheduling,” *OR Spektrum*, vol. 11, no. 1, pp. 3–16, 1989, doi: 10.1007/BF01721162.

[17] S. Barman, “Simple priority rule combinations: An approach to improve both flow time and tardiness,” *International Journal of Production Research*, vol. 35, no. 10, pp. 2857–2870, 1997, doi: 10.1080/002075497194480.

[18] 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, doi: 10.1016/S0925-5273(96)00068-0.

[19] F. Zhang, Y. Mei, and M. Zhang, “Evolving Dispatching Rules for Multi-objective Dynamic Flexible Job Shop Scheduling via Genetic Programming Hyper-heuristics,” *2019 IEEE Congress on Evolutionary Computation, CEC 2019 - Proceedings*, no. August, pp. 1366–1373, 2019, doi: 10.1109/CEC.2019.8790112.

[20] J. C. Tay and N. B. Ho, “Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems,” *Computers and Industrial Engineering*, vol. 54, no. 3, pp. 453–473, 2008, doi: 10.1016/j.cie.2007.08.008.

[21] J. Huang and G. A. Süer, “A dispatching rule-based genetic algorithm for multi-objective job shop scheduling using fuzzy satisfaction levels,” *Computers and Industrial Engineering*, vol. 86, pp. 29–42, 2015, doi: 10.1016/j.cie.2014.12.001.

[22] P. Priore, A. Gómez, R. Pino, and R. Rosillo, “Dynamic scheduling of manufacturing systems using machine learning: An updated review,” *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM*, vol. 28, no. 1. Cambridge University Press, pp. 83–97, 2014. doi: 10.1017/S0890060413000516.

[23] A. Teymourifar, G. Ozturk, Z. K. Ozturk, and O. Bahadir, “Extracting New Dispatching Rules for Multi-objective Dynamic Flexible Job Shop Scheduling with Limited Buffer Spaces,” *Cognitive Computation*, vol. 12, no. 1, pp. 195–205, 2020, doi: 10.1007/s12559-018-9595-4.

[24] S. Jun, S. Lee, and H. Chun, “International Journal of Production Research Learning dispatching rules using random forest in flexible job shop scheduling problems

Learning dispatching rules using random forest in flexible job shop scheduling problems," *International Journal of Production Research*, vol. 57, no. 10, pp. 3290–3310, 2019, doi: 10.1080/00207543.2019.1581954.

[25] A. Oukil and A. El-Bouri, "Ranking dispatching rules in multi-objective dynamic flow shop scheduling: a multi-faceted perspective," *International Journal of Production Research*, vol. 59, no. 2, pp. 388–411, 2021, doi: 10.1080/00207543.2019.1696487.

[26] Z. Tian and S. Fong, "Survey of Meta-Heuristic Algorithms for Deep Learning Training," *Optimization Algorithms - Methods and Applications*, Sep. 2016, doi: 10.5772/63785.

[27] M. O. Okwu and L. K. Tartibu, "Particle Swarm Optimisation," *Studies in Computational Intelligence*, vol. 927, pp. 5–13, 2021, doi: 10.1007/978-3-030-61111-8_2.

[28] T. M. Shami, A. A. El-saleh, and S. Member, "Particle Swarm Optimization : A Comprehensive Survey," pp. 10031–10061, 2022.

[29] W. Xia and Z. Wu, "An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems," *Computers and Industrial Engineering*, vol. 48, no. 2, pp. 409–425, 2005, doi: 10.1016/j.cie.2005.01.018.

[30] H. Liu, A. Abraham, and C. Grosan, "A novel variable neighborhood particle swarm optimization for multi-objective flexible job-shop scheduling problems," *2007 2nd International Conference on Digital Information Management, ICDIM*, vol. 1, pp. 138–145, 2007, doi: 10.1109/ICDIM.2007.4444214.

[31] A. Sadrzadeh, "Development of Both the AIS and PSO for Solving the Flexible Job Shop Scheduling Problem," *Arabian Journal for Science and Engineering*, vol. 38, no. 12, pp. 3593–3604, 2013, doi: 10.1007/s13369-013-0625-y.

[32] L. Xu, D. Jiawei, and H. Ming, "Research on hybrid cloud particle swarm optimization for multi-objective flexible job shop scheduling problem," in *2017 6th International Conference on Computer Science and Network Technology (ICCSNT)*, 2017, pp. 274–278. doi: 10.1109/ICCSNT.2017.8343701.

[33] Z. Wang, J. Zhang, and S. Yang, "An improved particle swarm optimization algorithm for dynamic job shop scheduling problems with random job arrivals," *Swarm and Evolutionary Computation*, vol. 51, no. March, p. 100594, 2019, doi: 10.1016/j.swevo.2019.100594.

[34] M. Nouiri, A. Bekrar, A. Jemai, D. Trentesaux, A. C. Ammari, and S. Niar, "Two stage particle swarm optimization to solve the flexible job shop predictive scheduling problem considering possible machine breakdowns," *Computers and Industrial Engineering*, vol. 112, pp. 595–606, 2017, doi: 10.1016/j.cie.2017.03.006.

[35] S. v. Kamble, S. U. Mane, and A. J. Umbarkar, "Hybrid Multi-Objective Particle Swarm Optimization for Flexible Job Shop Scheduling Problem," *International Journal of*

Intelligent Systems and Applications, vol. 7, no. 4, pp. 54–61, 2015, doi: 10.5815/ijisa.2015.04.08.

[36] "AN IDEA BASED ON HONEY BEE SWARM FOR NUMERICAL OPTIMIZATION."

[37] L. Wang, G. Zhou, Y. Xu, and M. Liu, "An enhanced Pareto-based artificial bee colony algorithm for the multi-objective flexible job-shop scheduling," *The International Journal of Advanced Manufacturing Technology*, vol. 60, Jun. 2011, doi: 10.1007/s00170-011-3665-z.

[38] J. Q. Li, Q. K. Pan, and M. F. Tasgetiren, "A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities," *Applied Mathematical Modelling*, vol. 38, no. 3, pp. 1111–1132, Feb. 2014, doi: 10.1016/j.apm.2013.07.038.

[39] X. Long, J. Zhang, K. Zhou, and T. Jin, "Dynamic Self-Learning Artificial Bee Colony Optimization Algorithm for Flexible Job-Shop Scheduling Problem with Job Insertion," *Processes*, vol. 10, no. 3, 2022, doi: 10.3390/pr10030571.

[40] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533–549, Jan. 1986, doi: 10.1016/0305-0548(86)90048-1.

[41] A. Baykasoğlu, "Linguistic-based meta-heuristic optimization model for flexible job shop scheduling," *International Journal of Production Research*, vol. 40, no. 17, pp. 4523–4543, Nov. 2002, doi: 10.1080/00207540210147043.

[42] S. Jia and Z. H. Hu, "Path-relinking Tabu search for the multi-objective flexible job shop scheduling problem," *Computers and Operations Research*, vol. 47, pp. 11–26, Jul. 2014, doi: 10.1016/j.cor.2014.01.010.

[43] J. Q. Li, P. Duan, J. Cao, X. P. Lin, and Y. Y. Han, "A hybrid pareto-based tabu search for the distributed flexible job shop scheduling problem With E/T criteria," *IEEE Access*, vol. 6, pp. 58883–58897, 2018, doi: 10.1109/ACCESS.2018.2873401.

[44] J. H. Holland, *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. 1975.

[45] X. Huang, Z. Guan, and L. Yang, "An effective hybrid algorithm for multi-objective flexible job-shop scheduling problem," *Advances in Mechanical Engineering*, vol. 10, no. 9, pp. 1–14, 2018, doi: 10.1177/1687814018801442.

[46] I. Vlašić, M. Đurasević, and D. Jakobović, "Improving genetic algorithm performance by population initialisation with dispatching rules," *Computers and Industrial Engineering*, vol. 137, no. March, p. 106030, 2019, doi: 10.1016/j.cie.2019.106030.

[47] M. B. S. Sreekara Reddy, C. Ratnam, G. Rajyalakshmi, and V. K. Manupati, “An effective hybrid multi objective evolutionary algorithm for solving real time event in flexible job shop scheduling problem,” *Measurement: Journal of the International Measurement Confederation*, vol. 114, no. September 2017, pp. 78–90, 2018, doi: 10.1016/j.measurement.2017.09.022.

[48] Q. Liu, Q. K. Pan, L. Gao, and X. Li, “Multi-objective flexible job shop scheduling problem considering machine switching off-on operation,” *Procedia Manufacturing*, vol. 39, no. 2019, pp. 1167–1176, 2019, doi: 10.1016/j.promfg.2020.01.353.

[49] “What is reinforcement learning? The complete guide - deepsense.ai.” https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/ (accessed Jul. 02, 2022).

[50] Y. Du, J. qing Li, X. long Chen, P. yong Duan, and Q. ke Pan, “Knowledge-Based Reinforcement Learning and Estimation of Distribution Algorithm for Flexible Job Shop Scheduling Problem,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, pp. 1–15, 2022, doi: 10.1109/TETCI.2022.3145706.

[51] K. Lei *et al.*, “A Multi-action Deep Reinforcement Learning Framework for Flexible Job-shop Scheduling Problem,” *Expert Systems with Applications*, vol. 205, no. February, p. 117796, 2022, doi: 10.1016/j.eswa.2022.117796.

[52] S. Luo, “Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning,” *Applied Soft Computing Journal*, vol. 91, p. 106208, 2020, doi: 10.1016/j.asoc.2020.106208.

[53] B. A. Han and J. J. Yang, “A deep reinforcement learning based solution for flexible job shop scheduling problem,” *International Journal of Simulation Modelling*, vol. 20, no. 2, pp. 375–386, 2021, doi: 10.2507/IJSIMM20-2-CO7.

[54] K. Genova, L. Kirilov, and V. Guliashki, “A survey of solving approaches for multiple objective flexible job shop scheduling problems,” *Cybernetics and Information Technologies*, vol. 15, no. 2, pp. 3–22, 2015, doi: 10.1515/cait-2015-0025.

[55] I. A. Chaudhry and A. A. Khan, “A research survey: Review of flexible job shop scheduling techniques,” *International Transactions in Operational Research*, vol. 23, no. 3, pp. 551–591, 2016, doi: 10.1111/itor.12199.

[56] I. Kacem, S. Hammadi, and P. Borne, “Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems,” *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 32, no. 1, pp. 1–13, 2002, doi: 10.1109/TSMCC.2002.1009117.

[57] K. M. Lee, T. Yamakawa, and K. M. Lee, “Genetic algorithm for general machine scheduling problems,” *International Conference on Knowledge-Based Intelligent*

Electronic Systems, Proceedings, KES, vol. 2, no. April, pp. 60–66, 1998, doi: 10.1109/kes.1998.725893.

[58] "Hive | Artificial Bee Colony Algorithm in Python." https://rwuilbercq.github.io/Hive/ (accessed Jun. 04, 2022).

[59] "Crossover (genetic algorithm) - Wikipedia." https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm) (accessed Jul. 03, 2022).

[60] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Annals of Operations Research*, vol. 41, no. 3, pp. 157–183, 1993, doi: 10.1007/BF02023073.

[61] M. Mastrolilli and L. M. Gambardella, "Effective Neighborhood functions for the fjsp," *Journal of Scheduling*, vol. 3, pp. 3–20, 1999.

[62] M. Cunha, "Scheduling of Flexible Job Shop Problem in Dynamic Environment," pp. 1–10, 2017, [Online]. Available: https://fenix.tecnico.ulisboa.pt/cursos/memec/dissertacao/283828618789989

[63] N. B. Ho, J. C. Tay, and E. M. K. Lai, "An effective architecture for learning and evolving flexible job-shop schedules," *European Journal of Operational Research*, vol. 179, no. 2, pp. 316–333, 2007, doi: 10.1016/j.ejor.2006.04.007.

[64] M. Yazdani, M. Amiri, and M. Zandieh, "Flexible job-shop scheduling with parallel variable neighborhood search algorithm," *Expert Systems with Applications*, vol. 37, no. 1, pp. 678–687, Jan. 2010, doi: 10.1016/J.ESWA.2009.06.007.

[65] L. N. Xing, Y. W. Chen, P. Wang, Q. S. Zhao, and J. Xiong, "A Knowledge-Based Ant Colony Optimization for Flexible Job Shop Scheduling Problems," *Applied Soft Computing*, vol. 10, no. 3, pp. 888–896, Jun. 2010, doi: 10.1016/J.ASOC.2009.10.006.

[66] J. Q. Li, Q. K. Pan, P. N. Suganthan, and T. J. Chua, "A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem," *International Journal of Advanced Manufacturing Technology*, vol. 52, no. 5–8, pp. 683–697, 2011, doi: 10.1007/s00170-010-2743-y.

[67] L. Wang, G. Zhou, Y. Xu, S. Wang, and M. Liu, "An effective artificial bee colony algorithm for the flexible job-shop scheduling problem," *International Journal of Advanced Manufacturing Technology*, vol. 60, no. 1–4, pp. 303–315, 2012, doi: 10.1007/s00170-011-3610-1.

[68] Y. Honghong and W. Zhiming, "The application of Adaptive Genetic Algorithms in FMS dynamic rescheduling," *International Journal of Computer Integrated Manufacturing*, vol. 16, no. 6, pp. 382–397, 2003, doi: 10.1080/0951192031000077870.