FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Scheduling Strategies for the Furniture Industry

Hugo Soares da Costa Faria Pires

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Jorge Manuel Pinho de Sousa (FEUP)

Second Supervisor: João Pedro Tavares Vieira Basto (INESC TEC)

June 28, 2021

Resumo

Desenvolvimentos tecnológicos aliados a padrões de produção mais exigentes têm constantemente testado os limites nas indústrias, alterando a perceção do que é possível e desejado em processos de manufatura. Tais melhorias têm frequentemente custos reduzidos, mas potencial para significativamente aumentar desempenho, permitindo uma grande vantagem competitiva.

Neste caso de estudo, é considerada uma fábrica na indústria de produção de móveis, onde a produção pode ser melhor aproximada pelo modelo *flow shop* e a característica mais importante é os tempos de *setup* dependentes da sequência. Com isto em mente, o objetivo era o de implementar uma estratégia de escalonamento da produção para melhorar a produtividade.

Para solucionar este problema, uma meta-heurística denominada de *iterated greedy with local search* é implementada, sendo responsável por organizar as ordens de produção na forma que melhor beneficia o tempo de produção e, consequentemente, a produtividade. Adicionalmente, foram testados mais dois métodos com o programa de simulação Flexsim: OptQuest, a ferramenta de otimização integrada, e uma regra local que permite que cada estação defina a sua sequência de trabalhos. Por fim, o método com o melhor desempenho foi comparado com uma heurística que tinha sido previamente criada para este problema.

Em testes foi verificado que a meta-heurística baseada em *iterated greedy with local search* é o método com melhor desempenho, já que criou as soluções com os menores tempos de produção e, em consequência, gerou a maior produtividade.

Abstract

Technological developments and more demanding production standards have constantly been pushing the envelope, changing the perception of what is possible and desired in manufacturing processes. Such improvements are often made at marginal cost, yet have the potential to significantly benefit performance, enabling a strong competitive advantage.

In this case study, a factory in the furniture industry is considered, where production can be best approximated by the flow shop model and the most critical characteristic is sequence-dependent setup times. With that in mind, the objective was to implement a production scheduling strategy to improve productivity.

To address this problem, an iterated greedy with local search meta-heuristic was implemented, being responsible for organising production orders in the way that best suits makespan and, consequently, productivity. Additionally, two more methods were tested with the Flexsim simulation software: OptQuest, the built-in optimiser tool, and a local rule which allowed each workstation to define its sequence of jobs. Lastly, the best performing of the previous methods was compared to an heuristic that had previously been created for this problem.

Through testing, it was found that the iterated greedy with local search meta-heuristic is the best performing method, as it produced solutions with the lowest makespan values and, consequently, generated the highest productivity.

Acknowledgements

First and foremost, I would like to thank my supervisors, Engineer João Basto and Professor Jorge Pinho de Sousa, whose expertise and guidance were invaluable to the whole process. Additionally, I would like to extend my gratitude to Engineer Romão Santos for all the help and advice provided.

Secondly, I would like to acknowledge my friends, who have made the last five years much more enjoyable and have helped me get to this point.

Last but not least, my appreciation also goes out to my family for the motivation and encouragement provided all through my studies.

Hugo Faria Pires

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 1
- 1.3 Objectives . . . 2
- 1.4 Methodology . . . 2
- 1.5 Structure . . . 3

2 Bibliographical Review **5**
- 2.1 Operations Scheduling . . . 5
- 2.2 Types of Operations Scheduling Problems . . . 6
  - 2.2.1 Job Shop . . . 6
  - 2.2.2 Flow Shop . . . 6
  - 2.2.3 Open Shop . . . 7
  - 2.2.4 Additional Characteristics . . . 7
- 2.3 Objective Functions . . . 7
- 2.4 Permutation Flow Shop Problem . . . 9

3 Approach **13**
- 3.1 Problem Description . . . 13
- 3.2 Basic Heuristic . . . 14
- 3.3 Meta-Heuristic Method . . . 15
  - 3.3.1 Makespan Calculation . . . 16
  - 3.3.2 NEH . . . 17
  - 3.3.3 Iterated Greedy . . . 17
  - 3.3.4 Iterated Local Search . . . 17
  - 3.3.5 Termination Criteria . . . 18
- 3.4 A Lower Bound on the Optimal Solution . . . 18
- 3.5 Simulation . . . 19
  - 3.5.1 General Concepts for the Simulation Model . . . 19
  - 3.5.2 Process Flow . . . 21
- 3.6 OptQuest . . . 24
- 3.7 Local Rules . . . 24

4 Computational Results **29**
- 4.1 Test Instance . . . 29
  - 4.1.1 Structure . . . 30
  - 4.1.2 Lower Bound . . . 30
  - 4.1.3 Iterated Greedy with Local Search . . . 31

4.1.4 OptQuest 33
4.1.5 Local Rules 36
4.1.6 Basic Heuristic 37
4.1.7 Conclusions 37
4.2 Case Study 38
4.2.1 Flow Shop Adaptation 39
4.2.2 Real Model 41

5 Conclusions and Future Work **43**
5.1 Conclusions 43
5.2 Future Work 44

References **45**

List of Figures

1.1 Methodological approach . . . 3

3.1 Iterated Greedy algorithm combined with Local Search, adapted from [1] . . . 16
3.2 Source . . . 19
3.3 Queue . . . 20
3.4 Processor . . . 20
3.5 Sink . . . 21
3.6 Simulation model . . . 21
3.7 Zones . . . 22
3.8 Process Flow . . . 22
3.9 Complete Process Flow . . . 23
3.10 Process Flow for Local Rules . . . 25
3.11 Lists . . . 25
3.12 Token addition to list . . . 26
3.13 Local Rules . . . 26

4.1 Test Structure . . . 30
4.2 Makespan vs time for IG-LS with test data set . . . 32
4.3 Makespan vs number of iterations for IG-LS with test data set . . . 32
4.4 Makespan vs time for OptQuest starting from NEH solution . . . 33
4.5 Makespan vs number of iterations for OptQuest starting from NEH solution . . . 34
4.6 Makespan vs time for OptQuest starting from IG-LS solution . . . 35
4.7 Makespan vs number of iterations for OptQuest starting from IG-LS solution . . . 35
4.8 Results from test instance . . . 38
4.9 Makespan vs time for IG-LS with case study data set . . . 40

List of Tables

4.1 Performance Tests 29
4.2 Machine Processing Times 30
4.3 Processing and setup values for each machine 31
4.4 Parameters for Iterated Greedy Algorithm 31
4.5 Results from the Iterated Greedy Algorithm 33
4.6 Results from the OptQuest Method 34
4.7 Results from the OptQuest Method starting from IG-LS solution 36
4.8 Makespan values for local rules starting from NEH solution 36
4.9 Makespan values for local rules starting from IG-LS solution 36
4.10 Parameters for case study 40
4.11 Results from case study adapted to flow shop 41
4.12 Results from case study 41

Abbreviations and Symbols

PFSP	Permutation Flow Shop Problem
SDST	Sequence-Dependent Setup Times
WIP	Work-in-Progress
IG-LS	Iterated Greedy with Local Search
WS	Workstation

Chapter 1

Introduction

1.1 Context

In a constantly changing environment, the need for companies to implement continuous improvement strategies is ever increasing. In this context, the search for methods that allow cost reduction and production capacity increase has been growing and is now a crucial activity in several industries. Such strategies frequently allow, at a marginal cost, better utilisation of previous investments and prevention of new ones.

In the furniture industry, one of the main needs is increasing flexibility in production processes, either in variety, or quantity. When it comes to variety, it concerns mass customisation, that is, the existence of customisation in products fabricated at a large scale. In quantity, this flexibility is related to shorter product life cycles, the existence of more competition, which may cause unstable demand, as well as extraordinary events, such as the COVID-19 pandemic, that created big demand fluctuations for some products.

Furthermore, with constant improvements in computational power, processing capacity has increased, enabling more developments in artificial intelligence that, in their turn, introduce solutions that were not previously feasible in the industry. Another advantage of this increased power is the diminishing of algorithm execution times, which makes tools like these more interesting for studying ever more complex problems.

1.2 Motivation

In this situation, the case of a large factory in the furniture industry is explored, where demand varies and an extensive range of products is produced.

Considering the existing range, the factory operates using a set of machines that can process different types of products although, when the type of product changes, there is a corresponding setup time, that depends both on the previous and the current items being manufactured at each machine.

Still, concerning setup times, there is a compromise between maximum production and meeting delivery times and, something else that has to be considered is that minimising these times on one machine may affect the rest. Besides the previously mentioned characteristic, products may have different routes in the production system, which adds complexity to the scheduling process.

Considering this, one very useful tool would be a holistic scheduling method that, in advance, evaluates the global impact of the developed production sequences on the overall productivity of a factory. In this scenario, this is measured through the sum of concluded production orders in a given time horizon, thus increases with a decrease in the total production time, that is, makespan.

These scheduling strategies have the capacity to produce very satisfactory results for companies as they can improve the flexibility of production, increasing the use rates of machinery, productivity and consequently, capacity. For these reasons, an efficient implementation of these strategies can result in reductions of around 20% of production time. [2]

1.3 Objectives

In this project the main objective consists of creating a customised heuristic or meta-heuristic to schedule the production of an existing factory in the furniture industry, to enable an increase in productivity.

When executing this algorithm, it will be possible to map out the production orders or, in other words, the sequence of jobs that will be worked on at each of the machines. For the best possible results, the characteristics of the process have to be taken into consideration, as they can have a large impact on production times.

To find the best solutions, several heuristics and meta-heuristics will be studied in the development process and the one that best suits the characteristics of the problem will be chosen.

Once the approach has been chosen, it will be tested and validated with data sets that represent the production environment at the factory, so that its performance can be accurately assessed.

1.4 Methodology

This dissertation follows the application of optimisation and simulation methods to a scheduling problem, whose purpose is to minimise the makespan, which will be reflected on the productivity.

During this study, heuristic methods will be applied, that implement problem-specific approaches, as well as more generalised meta-heuristics. Also represented in figure 1.1 is the basic heuristic, which is an algorithm that had previously been developed for this problem and that will be compared to the new methods.

The first algorithm to be applied is the NEH constructive heuristic, as it generates an initial sequence for the remaining approaches to start from. After this first solution, three other methods follow: the iterated greedy with local search meta-heuristic, which implements an optimisation technique, local rules, which use simulation to test a more distributed approach, and OptQuest,

which implements generalised meta-heuristics through simulation. In an attempt to further increase productivity, the results from the iterated greedy meta-heuristic will also be used as initial solutions to the local rule and OptQuest, which, in their turn, will recombine the order of the jobs.

After the results from the previous implementations have been analysed, the data set will transition from a simpler test instance to the larger real instance from the case study. In this second stage, the best performing technique from the previous step will be compared against the basic heuristic to conclude which of the two can best maximise productivity.

Lastly, it should be noted that the implementation of the iterated greedy meta-heuristic and the NEH heuristic make use of the Java programming language, whereas the local rule and OptQuest rely on the Flexsim simulation software.

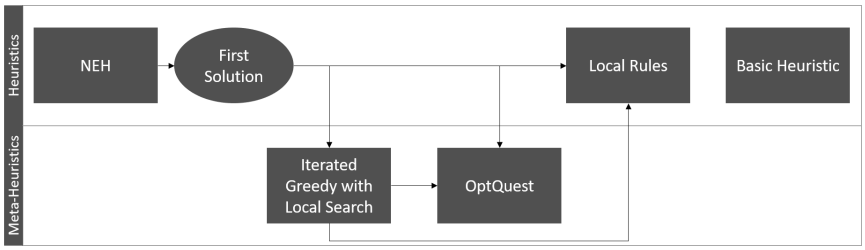

Figure 1.1: Methodological approach

1.5 Structure

Concerning the organisation of this document, it is divided into five chapters.

This first chapter was meant to introduce the theme, the motivation and the objectives of the dissertation, as well as an overview of the applied methodology.

In chapter 2, the scheduling problem concept is presented, along with its main variants and characteristics. Additionally, state-of-the-art solutions to the problem are reviewed.

Regarding chapter 3, it details the problem and the methods that were chosen, along with the previous approach, referred to as the basic heuristic, that had previously been developed to address it.

In chapter 4, the algorithms are tested and the corresponding results are analysed.

Lastly, chapter 5 contains the main conclusions that were drawn followed by suggestions of future work.

Chapter 2

Bibliographical Review

In this chapter, the main results from a bibliography research are presented, which allow for a better understanding of the key concepts related to the problem being studied.

Firstly, the concept of the operations scheduling problem is defined in section 2.1, then its most common types are subsequently explored in 2.2, along with some additional characteristics. A review of what the predominant scheduling objectives are is also presented in 2.3.

Lastly, in 2.4, comprehensive literature analysis of the current solution methods was carried out.

2.1 Operations Scheduling

Operations scheduling problems can be defined considering several variants of their parameters. In order to specify the different types of scheduling problems, a three-field $(\alpha|\beta|\gamma)$ classification method can be used, where α specifies the machine environment (single, parallel, others), β the characteristics of the job and γ, the optimisation criterion. [3] Regarding the first parameter, only multiple machine variants of the scheduling problem will be considered throughout the next chapters.

Supposing that m machines exist, $M_j(j = 1,\ldots,m)$, and that these have to process n jobs, $J_i(i = 1,\ldots,n)$, a scheduling solution consists of allocating time intervals to these machines to process the jobs. Usually, these solutions are represented using Gantt charts, either concerning the machines or the jobs to be processed. [4]

A job J_i is made up of n_i operations (tasks) $O_{i1},\ldots,O_{(i,ni)}$ and each operation has a corresponding processing time p_{ij}. Additionally, there is also a release date r_i related to the moment when the first operation becomes available. An operation can be processed in any of the compatible machines $u_{ij} \subseteq M_1,\ldots,M_m$ and, should this set only consist of one resource, it is a dedicated machine or, if it represents all the machines, these are parallel. At last, there can exist a cost function $f_i(t)$, that tracks the cost of completing a job J_i at time t. [4]

A solution is considered valid if no two-time intervals overlap on the same machine or the same job and if it complies with all the specific problem characteristics. This same solution is

optimal if it minimises a given optimisation criterion. [4] For this to happen, the cost function has to be set in a way that considers the actual real conditions in which the model will be applied. [5]

2.2 Types of Operations Scheduling Problems

The aim of this section is to analyse the most common scheduling problems in production industries. Depending on its type, production can be set to have high or low product variety, larger or fewer quantities, as well as different flow types.

Most of the problems with multiples machines are $\mathscr{NP}$-hard, meaning that they are based on combinations and that, usually, there are no exact algorithms to solve them in practical and acceptable computational time. One other common characteristic to consider is precedence relations, which note the need to complete one or more tasks before processing of a certain operation can take place. [6]

Considering that in reality, each problem comes with its own set of particular restrictions, in the description of every problem type there will only be considered the simplified and generalised models.

2.2.1 Job Shop

A job shop problem is often related to the production of small batches of a big variety of products. Consequently, the order with which the items go through the machines can vary with the product that is being manufactured and, as such, for each product type, there is a predefined machine sequence. In addition, the same machine can be re-used in later stages of the same production process, originating a re-entering flow.

There also exist precedence relations that take up the following form [4]:

$$O_{i1} \rightarrow O_{i2} \rightarrow O_{i3} \rightarrow \cdots \rightarrow O_{(i,ni)} \quad \text{for} \quad i = 1,\ldots,n \tag{2.1}$$

2.2.2 Flow Shop

In this type of problem, the transformation sequence is always the same, not depending on the product type. The set of jobs are processed sequentially on the machines, thus having the same processing order. In other words, the flow is constant but each product type is worked on a specific set of machines, having the following precedence relations [4]:

$$O_{ij} \rightarrow O_{(i,j+1)} \quad (i = 1,\ldots,m-1) \quad \text{for} \quad i = 1,\ldots,n \tag{2.2}$$

This problem, where the processing sequence is necessarily the same on all resources, is called permutation flow shop problem (PFSP). [7] In other variants, the order with which the jobs come in may not remain the same throughout production and, in some cases, there may even occur job passing.

2.2.3 Open Shop

An open shop scheduling problem is a multiple-machine, multiple-operations problem where there are no precedence relations between operations.

As such, it can be defined as a series of operations that have to be processed in specified amounts of time in each of the machines, without a specific order. So, should the routes not be deterministic, the problem is classified as an open shop. [8]

In this type of problem, the solution space is much greater than in the flow shop and job shop variants but, by its nature, is not very common.

2.2.4 Additional Characteristics

Aside from the characteristics that are inherent to the different types of scheduling problems, there are other variables that can affect the order with which the jobs can be sequenced. For this reason, this section will present some of the most frequent properties.

- Preemption

  Preemption, also known as job splitting, occurs when processing can be interrupted and resumed later on, even if not at the same machine. This behaviour might occur several times during the execution of a task. [4]

- Parallel resources

  Resources, such as machines, can be set in parallel when there are operations with processing times much larger than others, potentially causing bottlenecks. With this approach, production items can be sent to these resources on an alternate basis, improving the production process.

- Setup times

  During the manufacturing process, changes in the type of product that is being produced can routinely arise. As a result of this, some workstations might, for example, need adjustments, which take up a certain amount of time.

  One common variant of the PFSP has to do with the existence of setup times that depend on the production sequence (SDST). [7] In this situation, these times not only depend on the job that is going to be processed, but also on the one that came beforehand, increasing the need for adequate sequencing.

  These setup times can be deterministic when they assume a predefined value, or stochastic if harder to predict due to variations such as breakdowns or human factors.

2.3 Objective Functions

The objective of production scheduling is to find a solution (or solution programme) that minimises a cost function $f_i(t)$. In practical situations, multiple objectives are often considered, making the

problem more complex and, consequently, there is a search for good compromises between these objectives instead of optimal solutions.

- Flow time

  This time, to be minimised, is defined as the difference between the moment when a job is made available (r_i) and when the job is completed (C_i).

$$F_i = C_i - r_i \tag{2.3}$$

  The main indicator that motivates the optimisation of the flow time is WIP, as the greater that is, the higher the costs for the company. The minimisation of this amount of time can be represented as in the following expression [9]:

$$\sum_{i=1}^{n} w_i F_i \rightarrow Min \tag{2.4}$$

  The weight given to the job i, w_i, represents the priority factor of one job when compared to the others. As such, the weight can represent, for example, the inventory costs or the value added to one job. [10]

- Delivery Date

  Deviation from the delivery date, when compared to the production date, is monitored when lateness is the factor to be minimised. The main indicator is maximum lateness, given by the following expression [9]:

$$L_{max} = \max_{1 \leq i \leq n} (C_i - d_i) \tag{2.5}$$

  However, in a real scenario, it is more useful to first calculate T_i and then consider the weights w_i that correspond to each job:

$$T_i = max(0, C_i - d_i) \tag{2.6}$$

$$\sum_{i=1}^{n} w_i T_i \rightarrow Min \tag{2.7}$$

  With Ci being the moment when the last operation is effectively finished and di the corresponding previously established delivery date.

- Makespan

  This objective is characterised by the minimisation of the time that takes to go from the beginning of the first to the end of the last operation from a given set of jobs. It is the most

common goal when applying scheduling techniques, as it has a direct impact on costs. This objective can be represented in the following way [9]:

$$C_{max} = \max_{1 \leq i \leq n} \quad C_i \rightarrow Min \tag{2.8}$$

With C_i being the necessary time to process job i.

- Machine Utilisation

  Machine utilisation is an important metric in production systems as, more often than not, taking the most out of the available resources is something that is desired. As a result, utilising the average usage of the machines, the following criterion can be maximised [9]:

$$MU = \frac{\sum_{i=1}^{n} \sum_{k=1}^{m} p_{ik}}{m.C_{max}} \tag{2.9}$$

2.4 Permutation Flow Shop Problem

As it will be analysed in chapter 3, the problem type that is most similar to the actual problem in the factory is the permutation flow shop problem with sequence-dependent setup times, PFSP-SDST for short. As previously described, in PFSP it is assumed that all jobs have the same operation sequence in all machines and SDST means that these setup times vary not only with the task that is going to be processed but also with the one that came right before it. [11]

Given the complexity of this type of problem, $\mathscr{NP}$-hard, heuristic techniques are usually employed instead of exact optimisation methods. Meta-heuristic methods consist in the application of different algorithms to an initially provided solution, enabling better results. [12] Additionally, a lower bound can be considered so that there is a theoretically optimal solution to compare to the achieved results.

Taking into account that the problem is of the PFSP-SDST type, in this section there will be conducted a review of some methods that have been developed for similar environments.

Ruiz et al. [7] developed an advanced genetic algorithm with several adaptations, like the incorporation of a special initialisation technique of the population and new crossover operators. Aside from that, should the makespan result not improve after a certain number of generations, part of the population is also replaced. To have other algorithms to compare this one to, adaptations were made to existing meta-heuristics that had had positive results in non-SDST flow shop problems. In testing, using the Taillard [13] reference instances, it was concluded that the new genetic algorithm was significantly better than the adaptations of the other methods.

Sioud and Gagné [14] presented an optimised version of the migrating bird optimization (EMBO) algorithm and a heuristic, named STH, tailored to the setup times of the specific problem, aiming to minimise the makespan. Moreover, other improvements were made, such as an adapted neighbourhood and a new restart mechanism. After the heuristic was applied, its results were compared with other recent algorithms and STH revealed itself to be the most promising one.

Finally, after applying EMBO, it was also concluded that the achieved total production times were smaller when compared to other tested solutions.

Ruiz and Stützle [1] employed an iterated greedy (IG) heuristic method for a problem with two objectives, minimisation of makespan and weighted tardiness. As the IG method needs an initial solution to start from, the NEH heuristic from [15] is also applied. Considering SDST, two adaptations were developed, one of them featuring local search to further improve solutions provided by the iterated greedy heuristic. In short, the algorithm starts by removing a given number of elements from a solution and then adding these back in the way that best suits the objective, followed by a search of better neighbouring solutions. In the end, although easier to implement than many other methods, the iterated greedy heuristic presented results unmatched by any existing algorithms.

Li and Ma [16] put forward a multi-objective algorithm intended not only to minimise makespan but also flow time. For this, a discrete artificial bee colony algorithm based decomposition method was used, as well as a problem-specific constructive heuristic and a local search method, to improve the quality of the initial solution and to find better solutions for the non-optimised individuals, respectively. To assess the performance of these algorithms, the Taillard [13] reference problems were once again used, having been concluded that, by and large, the obtained results were superior to that of other algorithms.

Mansouri et al. [17] analysed a multi-objective problem, with the goal of minimising makespan and energy consumption in a two-machine flow shop. The chosen approach was a mixed integer linear multi-objective method, along with a constructive heuristic, Schedule Development Heuristic, for a quick analysis between objectives for the generated solutions. Ramezanian et al. [18] had a similar problem and solution to this first one. Starting from a mixed integer linear programming model, two heuristics were applied, iterated greedy and local search, and tested against the augmented $\varepsilon - constraint$ method. Taking a look at the results, it was inferred that the algorithm based on the iterated greedy method produced better results than its local search counterpart and that the first one was also capable of supplying high-quality solutions in reasonable amounts of time, thus surpassing the augmented $\varepsilon - constraint$ reference. On average, the solutions reduced the overall energy consumption by 15%.

One extension of the permutation flow shop problem for which there is significant literature is the hybrid variant. In this sort of problem, there is at least one production stage that has more than one machine, thus having parallel resources. [19]

Ruiz and Maroto [20] adapted a genetic algorithm to this hybrid extension. Previously it had been successfully employed in simpler flow shop problems that did not have sequence-dependent setup times. Among other optimisations, four new crossover operators were set up, along with improved restart mechanisms. To evaluate the performance of this meta-heuristic, nine other existing methods were adapted and tested on multiple random instances of the problem, from which it was concluded that the developed algorithm was somewhere between 53 and 135% more effective than the second-best at improving the overall manufacturing time.

Naderi et al. [21] looked into two advanced algorithms to specifically deal with the hybrid

layout of the resources and SDST. The first one is a dynamic dispatching heuristic while the second one is a meta-heuristic based on iterated local search, having both been coded to minimise makespan. When compared to seven other literature algorithms, the new approaches were able to provide better results. The heuristic was faster and more effective while the meta-heuristic ended up having better results in all small or medium instances and large setup times.

Pan et al. [22] suggested nine algorithms to optimise makespan. Six of these were meta-heuristics based on trajectory, three based on iterated local search and other three on iterated greedy. The last algorithms are population-based meta-heuristics: improved fruit fly optimisation, improved migrating bird optimisation and discrete artificial bee colony optimisation. These algorithms were compared to various adaptations of the most recent solutions to this problem, such as genetic algorithms, and it was found that the proposed meta-heuristic methods provided better results, particularly the one based on the discrete artificial bee colony method.

Chapter 3

Approach

In the first section of this chapter the problem is described in greater detail, according to the types of problems, characteristics, and objectives previously analysed.

In 3.2 the heuristic that was developed in a previous approach to this problem is explained. This basic heuristic was specifically designed around the case study and, for that reason, could not be applied to any other permutation flow shop problem.

The remaining sections focus on the three approaches that were implemented: the state-of-the-art method for flow shop problems with sequence-dependent setup times for the makespan objective, addressed in 3.3, OptQuest, the optimiser functionality built into the Flexsim simulation software, described in 3.6, and local rules which is a method that considers the uncertainty associated with setup times, explained in 3.7.

Additionally, in section 3.4, the algorithm that was developed to calculate a lower bound on the optimal solution for the problem is described and, in 3.5, the simulation model on which the OptQuest and the local rules approaches are based on is analysed.

3.1 Problem Description

The problem that is being analysed concerns the scheduling of production orders for a factory in the furniture industry. Regarding the type of problem, the one that it comes closest to is the flow shop model as, although there is no single flow of workstations, the existing flows seem to have great similarity between them. Consequently, the flows will, later on, be adapted so that there will only be one sequence of workstations, no matter the product being manufactured.

Additionally, there are setup times that depend on the sequence of tasks for each of the machines (SDST), in particular in the painting section where no colour transference can occur between two distinct products. Other variables that affect the production schedule are shifts, existing sequencing rules, as well as different routes between products.

The objective of this scheduling problem is maximising productivity by minimising makespan, that is, the total amount of time needed to produce a given number of jobs. As setup time can be

subject to uncertainty and has influence on makespan, it needs to be taken into account when creating new sequences of jobs.

To generate solutions that are true to the production environment, 14 weeks worth of real data from the facility will be used, while also taking into consideration the existing machines, production orders and production and setup times. As previously said, there are several other variables that affect the production schedule and, to consider them all, the results from the algorithm will then be fed to a simulation model that assesses the productivity of a given solution.

3.2 Basic Heuristic

In this section, the heuristic that was initially developed for this scheduling problem is going to be detailed, so that it can be understood what advantages and disadvantages it has, as well as what can be improved with the new method.

When this approach to the problem was first implemented, the focus was twofold: to balance the workload between machines and to minimise setup time. This allowed for more evenly distributed operating time between workstations whilst also contributing to lower setup times.

As this was not interpreted as a flow shop problem, there were several flows of workstations, depending on the type of product being manufactured. To simplify the sequencing logic, macro flows were created, with each one representing a set of flows that shared a common characteristic, for example, all the products that go through a specific machine.

One other aspect that was taken into account was that not every machine has the same amount of tasks to process, or even similar processing times for each job, which can cause bottlenecks in the production process. As such, a weight criterion was implemented where each workstation had a value attributed according to its potential impact on makespan, which was then multiplied by its setup times. Out of all of the available machines, three were identified as being critical and, for that reason their setup times will be the ones that have larger priority.

Algorithm 1: Basic Heuristic Method

```
Result: Optimised sequence of production orders
begin
    for Each week do
        while Unsequenced production orders exist do
            Choose macro flow with the smallest percentage of processing done;
            for Every production order in the macro flow do
                Assess associated setups;
            end
            Put production order with the smallest setup in the list of sequenced jobs;
            Remove production order from the list of unsequenced jobs;
            Update processing time in macro flow;
        end
    end
end
```

The first step of the algorithm is to choose a macro flow; this is done by checking which of the existing ones has the smallest percentage of processing already done, with the tie-breaker being the greatest total processing time for that week. Once the macro flow has been picked, the production order list can be accessed and from there it is chosen the job that entails the least amount of setup time. Lastly, this production order is added to the list of sequenced jobs, removed from the unsequenced and the total processing time of that macro flow is updated. All of these steps are part of a loop that runs until there are no more jobs left to be sequenced.

The described algorithm is executed separately for every week of production data, as the jobs were previously scheduled to a specific week, taking into account load distribution and delivery time conditions that are not considered in this method.

One shortcoming of this heuristic method is that operations scheduling is not taken into account, that is, the time at which operations begin and end. Consequently, by only considering setup times on the most critical macro flow, solution quality may be hindered, as makespan, the total amount of production time needed, is not directly contemplated.

3.3 Meta-Heuristic Method

From the bibliographical review carried out in 2.4, it was concluded that the method put forward by Ruiz and Stützle [1] represents the state-of-the-art for flow shop problems with sequence-dependent setup times for the makespan objective. As such, that is the meta-heuristic method that will be applied to this problem.

The chosen method is tasked with finding a new solution by changing the order with which the jobs are processed. It starts from an initial sequence provided by the NEH heuristic, to which a

local search procedure is applied so that the neighbouring solution space can be searched for better solutions. Next, the iterated greedy algorithm is used, once again combined with local search, and new sequences are created while a termination criterion has not been reached.

This method was implemented using the Java programming language and follows the classic flow shop design with setup times, therefore not considering existing local rules and shifts at the factory. An overview of the complete method is presented below.

```
procedure IteratedGreedy_for_SDST_flowshop
    π := NEH_RMB;                          % Initialization
    π := LocalSearch_Insertion(π);
    π_b := π;
    while (termination criterion not satisfied) do
        π' := π;                           % Destruction phase
        for i := 1 to d do
            π' := remove one job at random from π' and insert it in π'_R;
        endfor
        for i := 1 to d do                 % Construction phase
            π' := best permutation obtained by inserting job π_R(i) in all possible positions of π';
        endfor
        π'' := LocalSearch_Insertion(π');  % Local Search
        if O(π'') < O(π) then              % Acceptance Criterion
            π := π'';
            if O(π) < O(π_b) then          % check if new best permutation
                π_b := π;
            endif
        elseif (random ≤ exp{−(O(π'') − O(π))/Temperature}) then
            π := π'';
        endif
    endwhile
    return π_b
end
```

Figure 3.1: Iterated Greedy algorithm combined with Local Search, adapted from [1]

3.3.1 Makespan Calculation

During the execution of this algorithm, every time a solution is generated, its corresponding makespan, represented by $O(\pi)$, is assessed to decide if the sequence is kept or discarded.

To find the total production time of a series of jobs, the implemented method finds the starting and completion times of every operation of each job, from the first one in the sequence until the last.

For the starting time of an operation to be found, it has to be understood whether there were any tasks preceding the current one and, should there have been any, what their completion times were. An operation will then only commence when the current job is made available and the workstation is ready, which may imply setup.

When it comes to the completion time, this is found by adding the processing time of the operation to its starting time. Once all operations have been analysed and allocated a completion time, the makespan will correspond to the moment the last operation of the last job of the sequence is finished.

3.3.2 NEH

The NEH heuristic was proposed by Nawaz et al. [23] and implements a simple initial ordering logic to provide a better starting point for a meta-heuristic.

Given a set of jobs, the heuristic starts by ordering them from the one that has the greatest processing time to the one that has the smallest. Once that has been done, each job is removed from that list in that order and is inserted into the position that minimises the makespan value amongst all the available places. For example, if the first job from the ordered list had already been put at position 0, the second job can now be inserted either before or after that one, depending on what solution incurs the smallest production times.

For this problem, as there were setup times involved, the Ríos-Mercado and Bard [15] approach was implemented, which calculates the makespan as previously described in 3.3.1. Furthermore, when the best position for a specific job is trying to be found, the Taillard acceleration [24] is used, which allows completion times to only be calculated once and, for every other position, the makespan is simply incremented with the additional processing and setup time required.

3.3.3 Iterated Greedy

The iterated greedy algorithm is composed of three main phases: destruction, construction, and assessment.

In the destruction phase, a previously set amount d of jobs is randomly removed, without repetition, from the current solution and kept on a new list. In the construction part of the algorithm, these jobs are added back to the given sequence, following the NEH logic, that is, assessing for each production order which position grants the lowest makespan value before setting its place.

Once a new solution is generated, the acceptance criterion is applied to verify whether the new sequence is kept or discarded. Should the new solution carry a makespan value lower than the current one, then it becomes the current solution and, if it is also better than the best solution found up until that point, the best sequence is updated as well. When the makespan is higher than the present one, a value that depends on the difference between the two solutions and a temperature variable is calculated and, only if this is greater than a random number, the solution will be accepted.

The temperature is given by the following expression [1]:

$$Temperature = T * \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}}{n * m * 10} \tag{3.1}$$

where T is a pre-set value, p_{ij} is the processing time, n is the number of jobs, and m the number of machines.

3.3.4 Iterated Local Search

After every construction phase, the local search algorithm is summoned to verify if better neighbouring solutions exist. One such solution can be defined as a sequence of jobs in which at least

one of them is in a different position regarding the initial order.

To do this, a for loop that runs through every job in a sequence, without repetition, was implemented, where each job is removed from its position and is fed to the NEH method. In the end, if the solution was improved when compared to the initial sequence, the previously described loop is run again until no more improvement occurs and the result is returned to the iterated greedy algorithm.

This additional step enables improvement of the solutions generated by the iterated greedy algorithm, without ever making them worse.

3.3.5 Termination Criteria

As previously mentioned, the method represented in figure 3.1 runs until the termination criteria are reached. In this implementation, a time-based criterion was used.

Time is one of the easiest criteria that can be implemented, as it only requires a few instructions. Before the algorithm is run, a variable is set to equal the current time plus the amount of time wished for the method to run and, once the execution starts, this variable is checked before a new iteration is started. If the desired time value has already been reached, a new iteration cannot be started and execution finishes.

This criterion was chosen as it was the best way to compare the different approaches that were implemented, although it is not optimal, as the algorithm can get stuck or have already reached the best possible solution, and keeps running.

3.4 A Lower Bound on the Optimal Solution

A lower bound serves as a guideline as to what is theoretically possible to achieve by a given solution to a problem, which in this particular case, translates to how low the total amount of time needed to produce a batch of products can be.

At first, one such value was calculated just based upon the maximum processing time between all the machines, that is, the lower bound value is equal to the maximum sum of processing time at any of the machines.

However, knowing that the more accurate the lower bound is, the more useful it becomes in assessing a given solution, a second version of the method was developed, now considering setup times.

To reflect some of the influence that these times have over the makespan value, for every machine the minimum setup times of every job were summed up and added to the previously calculated values. However, to take into account the fact that the first operation (task) processed on a machine does not need setup, it was assumed that the job with the highest setup value between the minima was put on first and its setup was deducted, to reduce the overall impact on the makespan value.

One other aspect that was considered to further improve this calculation was the time at which a machine first started processing. As in a flow shop environment processing follows a specific

sequence between machines, work is only made available to a machine after the prior workstation has finished working on it. Consequently, to minimise the gap between the lower bound and what can actually be achieved, the minimum of the sum of processing times that had elapsed up until a machine was added on to each of the total values.

$$LB = \max_{\forall j \in M}\left(\sum_{i=1}^{n} p_{ij} + \sum_{l=1}^{n} \min s_{klj} - \max\left(\min_{\forall l \in J} s_{klj}\right) + \min_{\forall i \in J}\left(\sum_{o=1}^{j-1} p_{io}\right)\right), k \in J \tag{3.2}$$

Where i, k and l concern the existing jobs, j and o the machine in which they are processed, p_{ij} is the processing time of job i at machine j and s_{klj} is the setup time between jobs k and l at workstation j.

3.5 Simulation

In this section, the simulation approach will be detailed. To do this, firstly the Flexsim simulation elements that compose the simulation model will be explained, followed by the process flow, which implements the logic of how the system works. This section is the foundation work for the OptQuest and local rules approaches that will later be implemented.

3.5.1 General Concepts for the Simulation Model

Firstly, the one element that is essential to the whole system is the token, which is the connecting element between the 3D model and the process flow. Each token is connected to a flow item in the model and goes through the activities present in the process flow.

Source

The source is where the flow items for the simulation model are created. It manages the arrival of the jobs to be processed, which in this case will be simultaneous for every job of the same week.

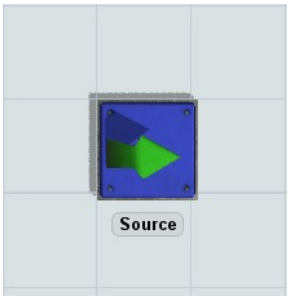

Figure 3.2: Source

Queue

This element precedes every workstation and creates a buffer with all the flow items waiting to be processed. Analysis of the different queues can help understand where the bottlenecks of a given process are.

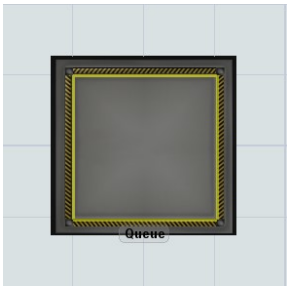

Figure 3.3: Queue

Processor

As the name suggests, the processor is the workstation where processing takes place and works by pulling the items from the corresponding queues, according to established rules. Both processing and setup times are read from tables containing values specific to the problem instance.

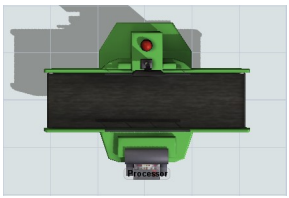

Figure 3.4: Processor

Sink

The sink does the opposite task of the source element, destroying the flow items that have been created, once a given job has been processed and has no further tasks. It is placed after the last workstation.

Figure 3.5: Sink

3.5.1.1 3D Model

To implement the chosen test instance, five processors were used along with the corresponding queues, source, and sink, representing the workstations and remaining elements from the problem, resulting in the model that follows.

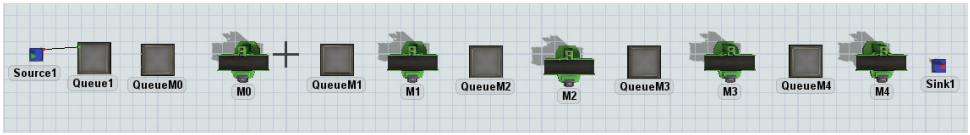

Figure 3.6: Simulation model

An additional queue was added right after the source, to help with ordering the jobs according to the provided sequences and then immediately sending these to the queue corresponding to the first machine, thus not having any impact on makespan.

3.5.2 Process Flow

To code the logic associated with the flow of items through the different elements, the process flow view of the model was used. This enabled one single sequence of commands to control movements throughout the model.

Figure 3.7: Zones

To enable the sequence of commands to work at every step of the model, one zone for each workstation was created, identified in the previous figure as QueueM0, for example. In each of these zones there can be several tokens waiting for their turn but only one active token, mirroring a job at a workstation.

The group of activities represented in the following figure is responsible for the transport, processing, and setup that occurs in the simulation model.

Figure 3.8: Process Flow

Firstly, a group with the queues that precede the workstations was created, so that a token, when arriving at any of those queues, can enter the first activity no matter the queue. Then, as previously mentioned, these tokens are only allowed to enter a zone if there are no other flow items already inside. If there is a waiting line to a zone, items are pulled by the date of arrival, thus maintaining the initial sequence. Additionally, it is important to note that a zone contains

the activities starting from when a token is moved to a machine and only ends when the setup is finished.

When a token does enter a zone, the item is then moved to the corresponding workstation to be worked on. However, each of these items only carries its token id, corresponding to the job identification, with no information regarding the processing time. As such, a custom code activity was inserted to search for this value in the corresponding table, according to the workstation and the token id.

Now that processing is finished, the item is then moved to the next queue unless it already is at the last workstation, in which case it goes directly to the sink to exit the model. Although there is no more processing for that item at that machine, the token does not exit the zone just yet, as the workstation may need setup before the coming job. Just like with the processing time, the setup time is also contained in a table, this time linking two jobs at a certain workstation. Consequently, the setup activity is tasked with searching for this time between the job that just finished processing and the one that is next on the sequence and assigns it to that particular machine. The one exception is when workstations process the last job, as no more setup is required.

Lastly, once the setup is finished, the token exits the zone, and the following item can be processed at the machine.

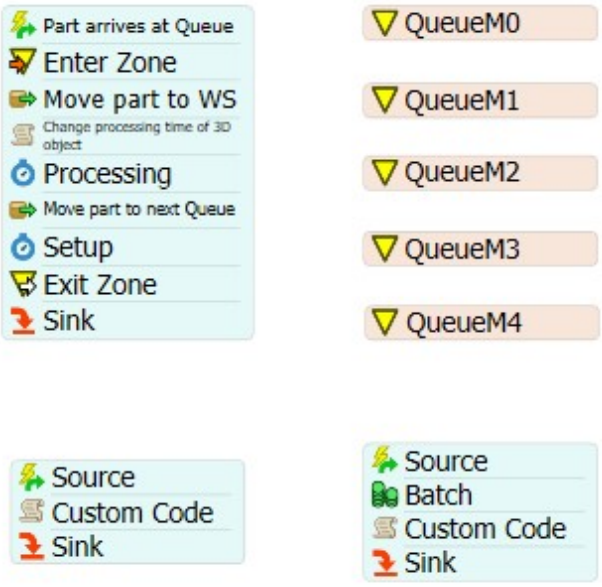

Figure 3.9: Complete Process Flow

Figure 3.9 represents the complete process flow, where two additional groups of activities were added. These can be seen as auxiliary functions to the other group, as the one on the bottom left corner is triggered to write the desired sequence to a variable when the simulation starts, and the one on the right prints the makespan value when the whole batch of jobs has been processed.

3.6 OptQuest

With the layout and corresponding process flow logic set up, the optimiser functionality built into the Flexsim software can now be used to generate its own sequences of production orders.

OptQuest is based on the application of various meta-heuristics, namely tabu and scatter searches. This approach was implemented so that several solutions are kept and combined to generate others, which not only lowers the likelihood of the method not evolving past a local optimum but also enables a much broader range of solutions to be found at any one point. [25]

Firstly, the parameters for the optimiser have to be set. As in this particular situation, each production order has an identification number ranging from 1 to the number of jobs, a parameter is set as a sequence of numbers to be ordered in the best possible way.

For a particular sequence of jobs to be assessed, a performance measure was also set. As the objective of scheduling these production orders is to minimise the makespan, this parameter was set accordingly.

Similarly to the algorithm that implements the meta-heuristic, this method also has to have termination criteria to know when to stop looking for a better solution. To compare these methods on equal terms, this criterion was also set regarding elapsed time.

One important aspect to note is that OptQuest implements a generalised approach to problems, meaning that no problem-specific characteristics are accounted for when the algorithm runs and, in consequence, the impact that these have on a solution is only assessed at the end of the iteration.

3.7 Local Rules

One strategy that could help improve the makespan times is to introduce local rules to the workstations, meaning that not only would the sequencing of jobs be done at a factory level but, in certain circumstances, at the machine level as well.

The local rule that was chosen was very straightforward: when a machine has finished processing, it checks its queue and picks the item that involves the smallest setup time. The goal of this rule is to minimise the amount of time that resources are doing setup and, by doing that, reduce the overall quantity of time needed to produce a set of orders. To implement this, some changes were made to the process flow.

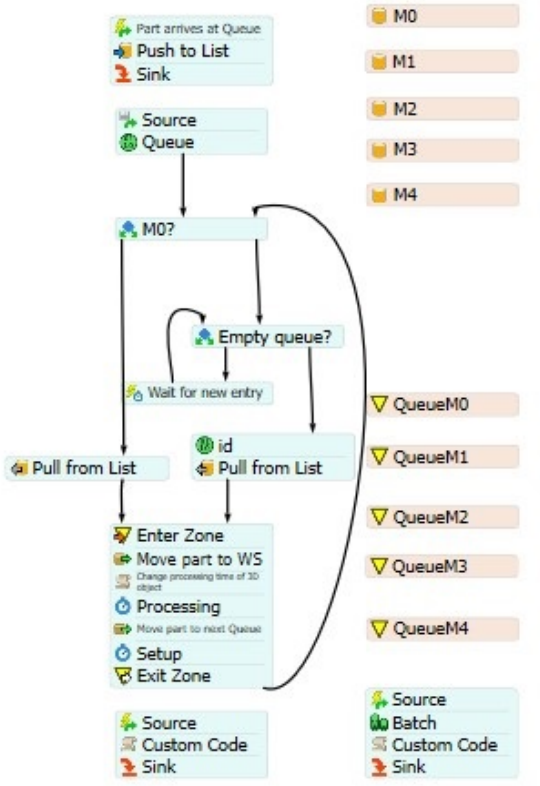

Figure 3.10: Process Flow for Local Rules

Figure 3.10 shows the process flow window with the local rules approach implemented and, as can be visualised, there are several additions to this model when compared to the centralised approach from 3.9.

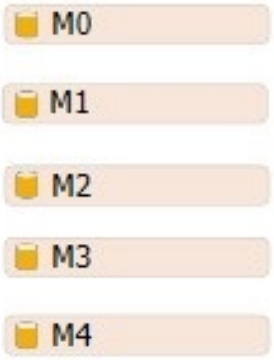

Figure 3.11: Lists

One of these additions is lists, one for the queue of each workstation. These shared assets were not needed before as tokens were automatically allowed inside the zones by arrival order. In this situation, as a more complex pull operation will be taking place, there needs to be a list that can

be queried using custom code. As such, these lists store the tokens corresponding to the items that are present at each of the queues.

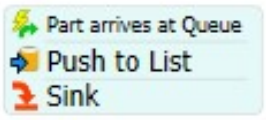

Figure 3.12: Token addition to list

Once a token arrives at a queue, it enters the block of activities from 3.12 where it will be added (pushed) to the list that corresponds to the queue, from where it will, later on, be removed (pulled).

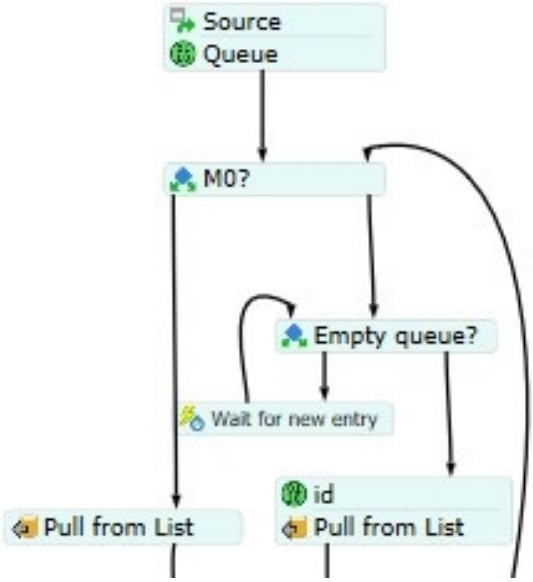

Figure 3.13: Local Rules

Moving on to figure 3.13, the first activity creates five tokens, one for each of the workstations and the second one assigns a label to each of them, stating the corresponding queue. What this essentially enables is for each one of the queues to be first checked for tokens and get the production process started.

In the example shown in the figure, the third activity is a decision point concerning machine M0; if the token is at the queue for that machine then it follows the path at the left and if it is at any other queue, the right side of the image is chosen. By changing the criteria at the decision point, the local rule can be applied to any combination of workstations.

Following the path at the left means that the local rule is not applied and the token is directly pulled from the list and is forwarded to the block of activities described in subsection 3.5.2, where zones are implemented.

On the other hand, the path at the right applies the aforementioned local rule. So, for machines other than M0, their queues are checked for tokens; if there are none it waits for one to arrive, if there are tokens a custom code is run to identify which one of the available tokens implies the smallest setup time. To do this, the ids from the waiting tokens are retrieved and, after consulting the table that contains the setup times between two ids for the machine in question, the token with the smallest time is chosen to be pulled from the list. Should there be no more jobs to process, the setup time is zero. At last, the token enters the section described in 3.5.2.

As can be seen, this process flow logic runs in a loop, that is, once a token exits a zone it goes back to the decision point where it is checked what the machine number is and, consequently, whether the local rule is going to be implemented or not. This strategy was chosen so that each machine can have the local rule or not, allowing different options to be tested and more data to be gathered concerning the effectiveness of such strategy.

Chapter 4

Computational Results

Following the implementation of the methods described in the previous chapter, the algorithms will be put through a series of tests to verify their respective performances and draw comparisons between them.

In the following table an outlook of the different tests is presented, where each row refers to the different data sets that were used and the columns to the methods that are going to be compared.

Table 4.1: Performance Tests

	IG-LS	**Basic Heuristic**	**OptQuest**	**Local Rules**
Test Instance	x	x	x	x
Case Study (FSSDST)	x	x		
Case Study	x	x		

The first data set was obtained from the article where the meta-heuristic method was proposed [1], the second one (FSSDST) is an adaptation of a production data set to the classic flow shop with SDST and the third takes into account every feature of the production environment, such as specific rules or shifts, making it more accurate and, at the same time, virtually impossible to calculate a lower bound for.

The test instance was a smaller and simpler set of data that allowed initial conclusions to be drawn, to understand what was the best approach for the larger and more complex production set. As such, the newly implemented meta-heuristic is compared against the results from OptQuest and from the local rule variant, as well as with the basic heuristic.

4.1 Test Instance

In this first data set, there are 50 jobs to be processed on 5 sequential workstations, following the flow shop design logic. For every job there is a corresponding processing time on each of the machines and, between two jobs, there is also a setup time at each workstation.

4.1.1 Structure

Regarding the structure of the tests that are going to be performed, the initial starting point is the NEH heuristic method, which will provide a non-optimised sequence of production orders for the remaining methods.

In the second stage, the remaining algorithms are applied and tested five times each, enabling a deeper understanding of what the best technique for this problem is. As such, the iterated greedy meta-heuristic is tested, along with OptQuest, the local rule and the basic heuristic method.

To verify whether OptQuest and the local rule would perform differently when compared to the previous step, these methods were put in use once again, now starting from the solutions generated by the meta-heuristic, instead of the ones from NEH.

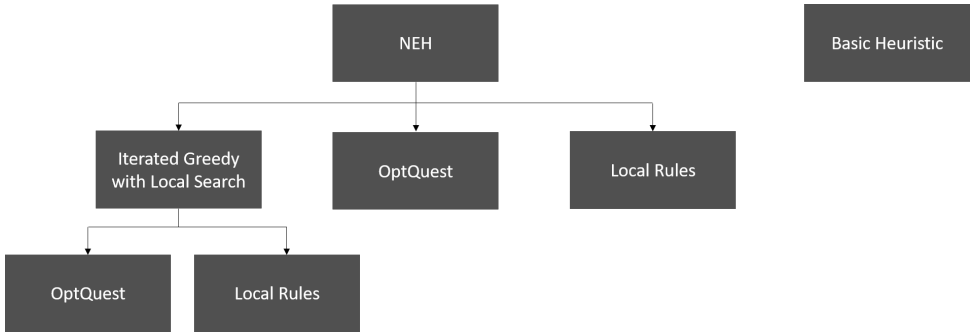

Figure 4.1: Test Structure

4.1.2 Lower Bound

The first attempt at a lower bound on the optimal solution was solely based upon the maximum processing time between all of the machines. Like so, for each one of the five machines, the processing times of every job were added up as displayed in the following table.

Table 4.2: Machine Processing Times

Machine	Processing Time (sec.)
M0	2750
M1	2470
M2	2464
M3	2784
M4	2258

Bearing in mind that the machine that finishes processing last is the one that sets the production time, the lower bound is 2784 seconds, corresponding to machine M3.

In the second approach to a lower bound, both the amount of time it took for the first job to arrive at a given machine and setup times were taken into account, generating higher and more realistic makespan values.

Table 4.3: Processing and setup values for each machine

Machine	Processing	Total Setup	Max Setup	Previous Processing	Final Value
M0	2750	76	6	0	2820
M1	2470	75	4	3	2544
M2	2464	77	4	18	2555
M3	2784	86	5	52	2917
M4	2258	75	6	90	2417

Lastly, just like in the first version of this method, M3 is the machine that defines the lower bound, which now corresponds to 2917 seconds.

4.1.3 Iterated Greedy with Local Search

Firstly, to apply the iterated greedy with local search method, the initial solution has to be generated using the NEH algorithm, previously described in 3.3.2, which resulted in a makespan of 4571 seconds.

Once an initial solution exists, the iterated greedy method can be applied to find solutions that can improve on the existing makespan. To track the performance of this method, the timestamp and makespan value of each of the generated results were saved to be displayed in graphs.

Regarding the parameters mentioned in 3.3.3, the number of jobs and machines are gathered from the current data set and the number d of removed jobs and T match the values proposed in the original article.

Table 4.4: Parameters for Iterated Greedy Algorithm

Parameter	Value
n	50
m	5
d	4
T	0.5

To have a better understanding of how the makespan values evolve with time, the termination criterion was based on elapsed time and set at 40 minutes. This time value was chosen upon analysis of the evolution of the best solution for both the meta-heuristic and OptQuest methods.

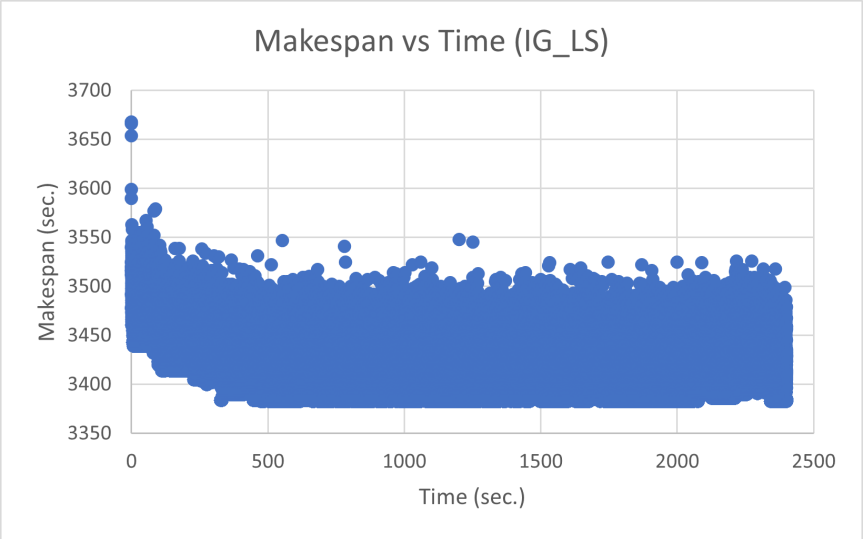

Figure 4.2: Makespan vs time for IG-LS with test data set

This first graph represents the solutions generated in one of the tests to the iterated greedy method, concerning their timestamp and associated makespan value. Thanks to the destruction, construction and local search mechanisms, there is a vast and immediate improvement over the initial sequence provided by the NEH algorithm, showcasing the efficiency of this method. Furthermore, due to the acceptance criteria that follow the previous steps, the amplitude of solutions with regards to the makespan is kept fairly constant right from the beginning until the end of the test.

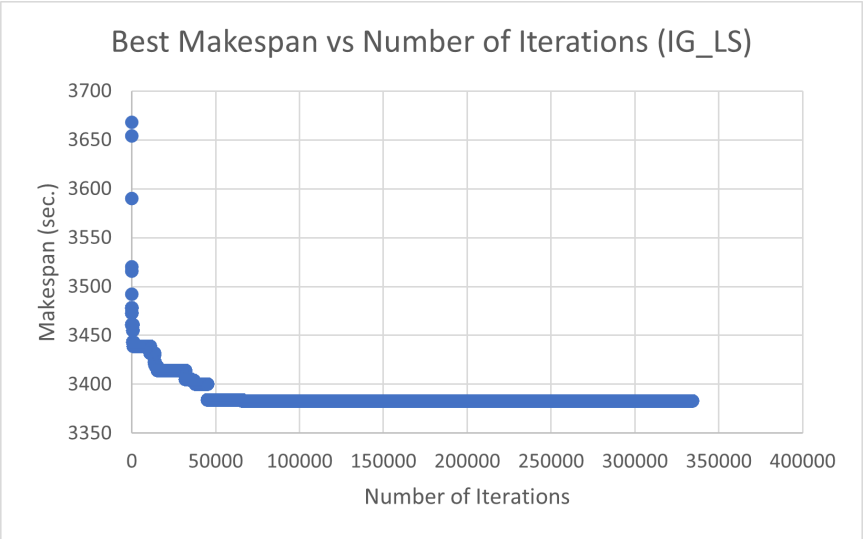

Figure 4.3: Makespan vs number of iterations for IG-LS with test data set

The second graph represents the evolution of the best makespan value with the number of iterations of the algorithm. As can be observed, in this test there was a significant number of

iterations, meaning that the algorithm runs rather quickly but, the main takeaway from this curve is that there were only needed less of a sixth of those iterations to find a solution that could not be further improved upon.

To ensure that the implemented meta-heuristic was performing as expected, the results were directly compared to those published by the author of this method and, as similar makespan values were achieved, the implementation was validated. Lastly, an overview of the five tests that were performed is presented below.

Table 4.5: Results from the Iterated Greedy Algorithm

Indicator	**Minimum (sec.)**	**Average (sec.)**	**Maximum (sec.)**
Value	3383	3391	3399

4.1.4 OptQuest

Besides the iterated greedy with local search algorithm, one other meta-heuristic based sequencing method was applied, using the optimiser feature from the simulation software. So to have comparable results, the OptQuest was also run for the same amount of time, 40 minutes or 2400 seconds, and the starting point was the same, the sequence created by the NEH heuristic.

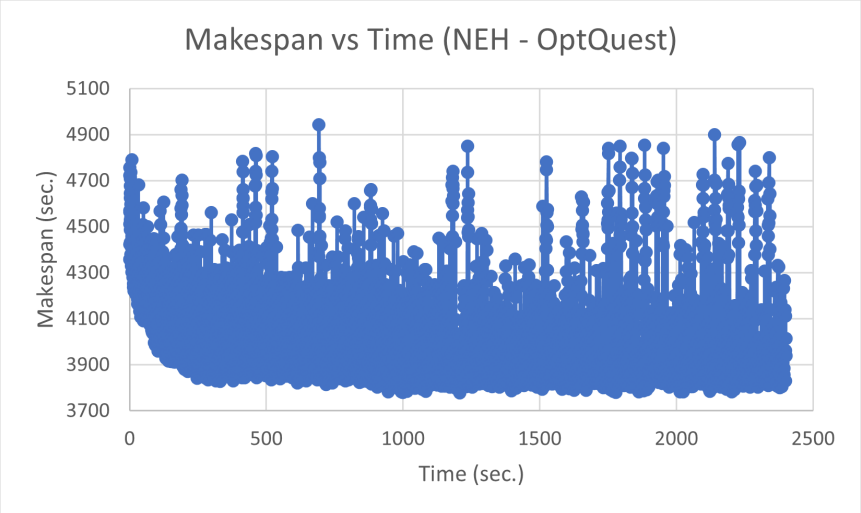

Figure 4.4: Makespan vs time for OptQuest starting from NEH solution

In this first graph, comparing to what was shown in 4.1.3, the amplitude of the solution values is much greater at any point in time, as poor makespan values occur throughout the test and not just at the beginning. This can be due to looser acceptance criteria or the fact that the makespan is only assessed at the end of the iteration.

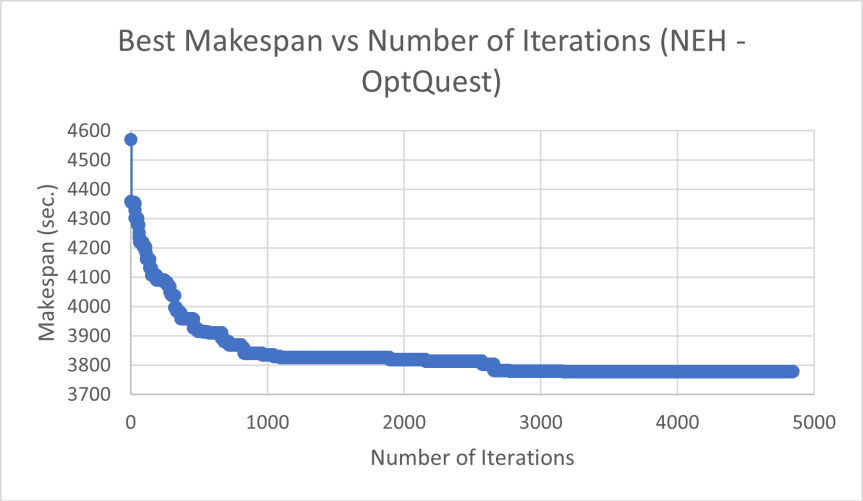

Figure 4.5: Makespan vs number of iterations for OptQuest starting from NEH solution

This second figure illustrates the evolution of the best makespan with the number of iterations and, when compared to the same graph for the first method, there are several noticeable differences. Whereas in the meta-heuristic there were more than three hundred thousand iterations, in this method that number was closer to five thousand for the same amount of time, implying that the latter needed a greater amount of time to run one iteration. Consequently, the best solution is only found close to the middle of the x-axis.

Table 4.6: Results from the OptQuest Method

Indicator	**Minimum (sec.)**	**Average (sec.)**	**Maximum (sec.)**
Value	3688	3732	3779

Regarding the makespan values that OptQuest was able to achieve, all three indicators point to significantly greater times when compared to the iterated greedy with local search algorithm. For example, the average makespan was 3732, while the latter method achieved a value of 3391, which appears to indicate that the former tool is less effective.

One additional assessment that was done to OptQuest had to do with understanding if it could further improve already optimised solutions. To do this, instead of the initial solution being provided by the NEH heuristic, a solution from the iterated greedy with local search was chosen instead.

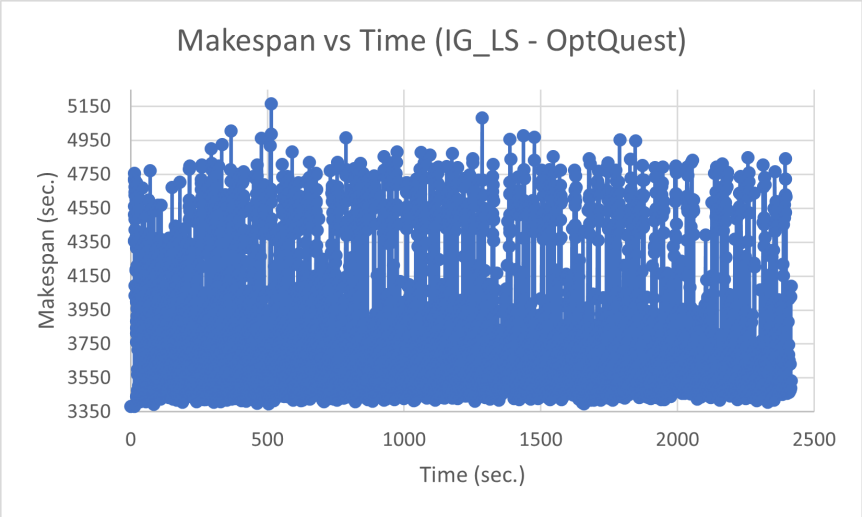

Figure 4.6: Makespan vs time for OptQuest starting from IG-LS solution

As was to be expected, throughout the graph represented in 4.6 there does not seem to be a trend or a sharp change in the range of values obtained for the makespan, as the initial solution was already optimised.

This lack of evolution is confirmed in the graph below, where the best makespan is tracked and shows that the first solution generated by OptQuest had a makespan of over 4300 seconds which was immediately brought down to the 3400 level, closer to the initial solution, although never improving it.

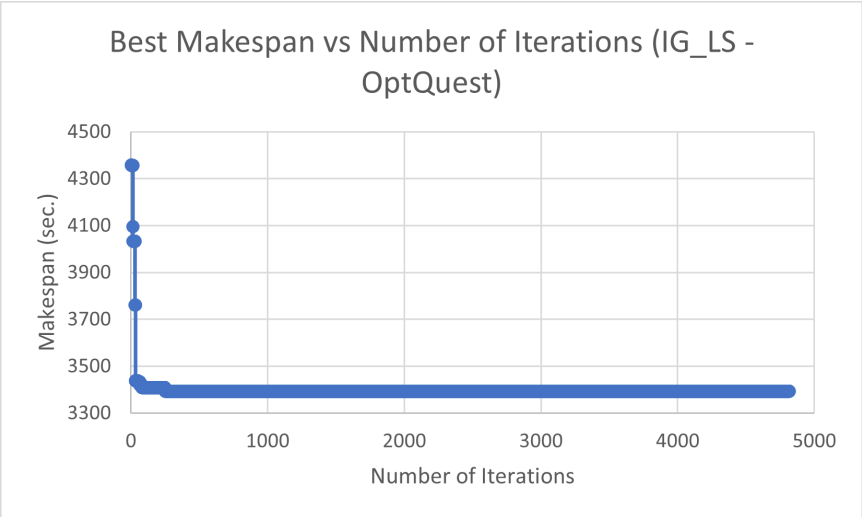

Figure 4.7: Makespan vs number of iterations for OptQuest starting from IG-LS solution

As a result of applying the OptQuest algorithm to the solutions output from the previous meta-heuristic method, all indicators show that those sequences were not improved as, for example, the average makespan went from 3391 to 3401 seconds.

Table 4.7: Results from the OptQuest Method starting from IG-LS solution

Indicator	**Minimum (sec.)**	**Average (sec.)**	**Maximum (sec.)**
Value	3393	3401	3418

Put in a nutshell, the optimiser feature implemented by OptQuest is not only less effective than the iterated greedy with local search method, as it leads to worse results but is also less efficient, as it takes longer to reach its best solution. Moreover, applying OptQuest to already optimised solutions did not provide further improvement to the overall production times.

4.1.5 Local Rules

Following testing done on the meta-heuristic implementations, it will now be studied whether local rules on workstations produce a positive or negative effect on the makespan value.

As detailed in 3.7, the local rule that was applied to each machine was tasked with searching the corresponding queue and selecting the job that involved the fastest setup time.

As when a workstation changes the order of its jobs sequence, all the subsequent machines have their order affected as well, this rule was applied on a machine-by-machine basis from the last one in the flow to the first, enabling a better understanding of the consequences of its application.

Table 4.8: Makespan values for local rules starting from NEH solution

M0	**M1**	**M2**	**M3**	**M4**	**Makespan (sec.)**
				x	4534
			x	x	4504
		x	x	x	4500
	x	x	x	x	4539
x	x	x	x	x	4603

The first sequence that was tested was the one that resulted from the NEH heuristic, which had a makespan of 4571 seconds. When the local rule was applied to the last three machines, the makespan was lowered to 4500 seconds but when the first two machines were also brought in, this value went back up, that is, the solution was worsened.

To further explore the impact that this rule has on makespan, it was also tested on sequences that resulted from the iterated greedy method, which had an average makespan value of 3391 seconds.

Table 4.9: Makespan values for local rules starting from IG-LS solution

M0	**M1**	**M2**	**M3**	**M4**	**Average Makespan (sec.)**
				x	3843
			x	x	3836
		x	x	x	3840
	x	x	x	x	3858
x	x	x	x	x	4054

In this example, the average of the makespan of the original solutions was not decreased by the local rule in any of the tested scenarios. Furthermore, the results were varied: the least impacted solution went from an initial makespan of 3383 seconds to 3391 in the best scenario, while the solution that was worsened the most went from 3390 to 4423 seconds, also in its most favourable combination of workstations with the local rule. Overall, the average of the best scenarios for each solution was 3836 seconds, up from the 3391 seconds before the application of the local rule.

In short, the initial solution was only slightly improved when it started from the NEH heuristic, which was a sequence that was far from optimised. Even then, when more than three machines had the local rule, the makespan was no longer improved when compared to the initial one, as when changing the processing sequence to minimise setup times in one machine, this order also changes for the remaining workstations, potentially increasing their setup times. As such, this rule does not prove to be as effective as the more centralised approach of the meta-heuristics, which consider the setup times of all machines when a new sequence is created.

4.1.6 Basic Heuristic

In this next test, the heuristic that had been developed in a previous approach to the actual scheduling problem at the factory is going to run the test instance, which is tailored for the classic flow shop problem.

As this method uses a set of established rules that do not depend on random functions, for a given input the output will always be the same, hence it only needs to be tested once for a particular data set. With the sequence generated, the corresponding makespan was calculated to be at 3958 seconds, which is worse than both the iterated greedy with local search method and OptQuest.

As explained in 3.2, this method is mainly focused on minimising setup times and balancing machine loads and does not consider when operations begin and end. For that reason, less optimised solutions are created, which have higher makespan values.

4.1.7 Conclusions

In this section of the results, several strategies for optimising the production sequence of a test instance were tested. The corresponding results are summarised in the following diagram.

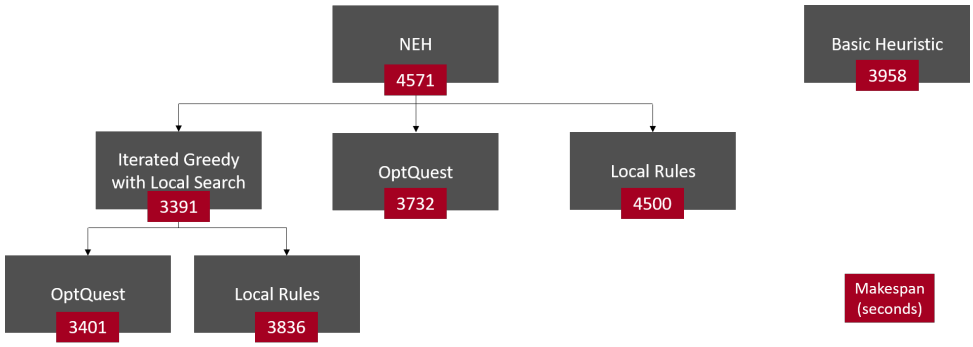

Figure 4.8: Results from test instance

Starting from the NEH heuristic, this solution was used as an initial sequence for three other methods. Firstly, the iterated greedy with local search method was able to significantly improve the makespan, then OptQuest also reduced this time, although not by as much, and the local rule only generated a small improvement over the initial makespan.

In the following step, both OptQuest and the local rule were applied to the solutions created by the iterated greedy method, in an attempt to further improve results. Analysing the obtained values, both methods were not capable of optimising the sequences previously generated by the meta-heuristic but, whereas OptQuest produced similar results, the local rule did not.

One other method that was tested was the basic heuristic that had previously been created for this problem. This algorithm, much like NEH, is a constructive heuristic that does not need an initial solution to start from and follows a set of rules, thus always generating the same result for the same data set. The solution that it created had a makespan of 3958 seconds, which is better than NEH but not as optimised as the meta-heuristic method or OptQuest.

To sum up, the method that came closest to the lower bound of 2917 seconds was the iterated greedy with local search meta-heuristic at 3391 seconds and, for that reason, this algorithm was chosen to be applied to an instance from the case study.

4.2 Case Study

Considering that all the implemented strategies have been tested with the test instance, now the best among those will be used to optimise the case study and compared to the basic heuristic that existed before.

This data set corresponds to 14 weeks and 1197 production orders, being processed at 13 machines at the factory. This represents a real and much larger instance of jobs, when compared to the test instance.

In the first stage, these two algorithms will be compared considering the flow shop adaptation of the problem, as that is the environment the iterated greedy with local search meta-heuristic was designed to work with. To do that, a sequence of machines was defined so that there was a single flow of products between the workstations.

The last test will be done in a simulation model that portrays the real environment at the factory, taking into account all the characteristics. This will ultimately dictate how useful the newly implemented method is for increasing the productivity of the factory, compared to the basic heuristic.

4.2.1 Flow Shop Adaptation

For the new algorithm to work with data from the case study, it first needs to be put in a format that can be read and understood by the method. As the basic heuristic does not implement a flow shop-based approach, there is no pre-defined single flow of workstations for the jobs to follow. As such, for a unique flow to exist, there can be no conflicts between existing routes for the products, that is, considering two workstations (A and B) no two routes can exist wherein one product goes from A to B and, in the other, from B to A.

A new method was coded to verify the applicability of a flow shop design, according to the previous considerations. In its execution, it was found that some flows to one workstation stood in the way of the single route approach but, as it only had a relatively small number of orders, this machine was simply removed from the main flow.

After no conflicts remained, a single flow was created to allow the problem to be interpreted as a flow shop. As such, a list was made for every machine, containing all the machines that preceded the first one in at least one route, and these were sequenced according to the following algorithm:

Algorithm 2: Creation of single flow

Result: Single sequence of workstations

begin

  Get workstations without precedents and add to the freeWS list, the remaining add to nonFreeWS;

  while *freeWS.size() > 0* **do**

    Get the first workstation from freeWS and add it to the SingleFlow list, remove it from freeWS and update the precedents in nonFreeWS;

    If a workstation in nonFreeWS now has no precedents, add to freeWS;

  end

  Check that nonFreeWS.size() == 0

end

What this algorithm essentially does is verify whether a machine has other machines preceding it in any flows/routes; if it does not have any it is added to the new single flow and is removed from the precedents of the remaining workstations. This procedure is repeated until all machines are in the new flow.

Some other modifications that had to be made with the changeover to the new data set were to do with the calculation of the makespan and the lower bound values. In particular, the methods were adapted to consider null values in the matrices by using more complex search methods.

Now that a single sequence of workstations exists, the iterated greedy with local search meta-heuristic can be applied. In this instance, there are 1197 production orders to be processed in 13 machines and parameters d and T were kept at the values proposed by the author of the method, explained in 3.3.3.

Table 4.10: Parameters for case study

Parameter	Value
n	1197
m	13
d	4
T	0.5

Just like in the previous tests, the evolution of the best makespan was documented, this time in minutes instead of seconds because of the size of the data set. One other modification that was made to this process was to only save information about the iterations that had improved or equalled the current best makespan, as otherwise, the file would get very large.

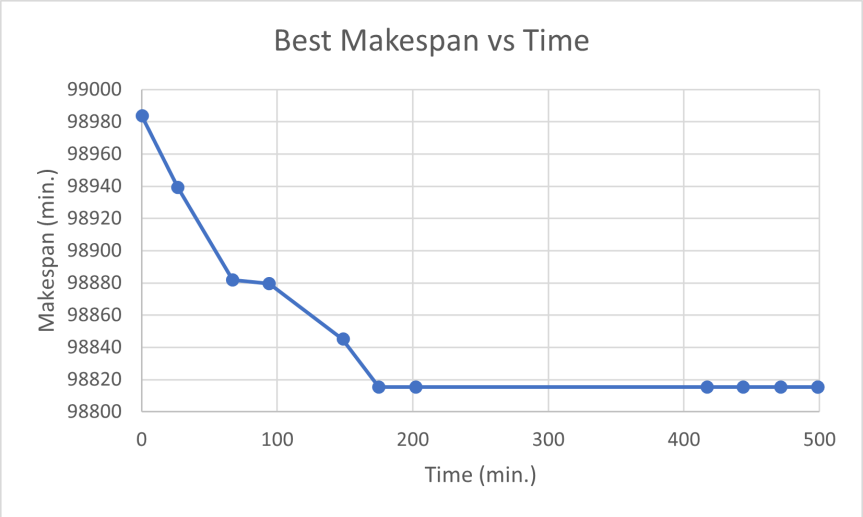

Figure 4.9: Makespan vs time for IG-LS with case study data set

In this graph, the initial starting point, which is generated from local search to the NEH solution, already had a very low makespan value compared to the 97585 minutes estimated by the lower bound. Nevertheless, the meta-heuristic was able to significantly decrease that time and converge to a solution 160 minutes better in less than half of the available time, now extended to eight hours.

Feeding the same data set to the basic heuristic, which runs a set of instructions focused on setup times and load distribution, the result in a flow shop model was 124949 minutes, which is much worse than the 98815 recorded by the iterated greedy. Despite the large difference in makespan, this result was expected as this former method was not intended to be used in a flow shop environment, but in the real model.

Table 4.11: Results from case study adapted to flow shop

Method	**Makespan (min.)**
NEH	132686.95
Lower Bound	97584.96
IG-LS	98815.31
Basic Heuristic	124 949.40

4.2.2 Real Model

Making use of the solutions generated in the last step by the two methods, it will now be assessed just how effective they would be in the real-world model. To do this, a more accurate simulation model had to be used, where problem characteristics other than setup times can be considered, such as pre-existing local rules, shifts at the workstations and different routes.

Table 4.12: Results from case study

Method	**Productivity**
IG-LS	1661k
Basic Heuristic	1285k

The goal of this scheduling problem was to increase productivity at the factory, measured through the sum of concluded production orders in the fourteen-week time horizon, by decreasing overall production times. It was not abundantly clear how the iterated greedy method would perform in this test, as the algorithm was designed to work with a single flow but, through analysis of the above-represented table, it can be concluded that this method achieved substantially greater productivity than the basic heuristic.

The main difference between these methods has to do with the newer one considering operations scheduling, that is, the time at which operations begin and end, while the basic method does not. By only checking setup times and machine loads, this heuristic produces worse results than the meta-heuristic, even though the latter was designed to work with the flow shop model and not several routes.

With this last verification, it has now been proven that the newly implemented method, iterated greedy with local search, is able to produce sequences of production orders that result in smaller makespan times and, consequently, greater productivity, when compared to the basic heuristic method.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The main objective of this project was to create a customised method to schedule the production process of a factory in the furniture industry to increase productivity. In the end, this algorithm would be compared to the one that had previously been developed for this problem, to assess whether there had been an improvement.

The first approach that was implemented to sequence the production orders was an iterated greedy with local search meta-heuristic, using the Java programming language. To the same end, making use of the Flexsim simulation software, two more strategies were employed: OptQuest, an optimiser tool, as well as a local rule at the workstation level.

In initial tests, which used a test instance, the iterated greedy approach generated the best results, followed by OptQuest and then the local rule. OptQuest was not only not as effective as the first meta-heuristic, because it created solutions with higher makespan values, but was also less efficient as it took much longer to converge to its best solution. The local rule was even less successful, seeing that it worsened the initial solution in most cases and only achieved a slight improvement in the remaining sequences.

The main drawback of OptQuest is that it applies a generalised algorithm and does not take in any characteristics of the problem while generating a new solution, thus being less prone to finding optimal solutions. Regarding the local rule, it did not prove to be successful in minimising setup times as, by changing the order in which jobs are processed at a machine, the sequence for the subsequent workstations is affected as well, which can increase overall setup times.

Taking that the iterated greedy meta-heuristic was the best-performing method in the previous tests, this algorithm was then applied to the case study. In a first step, this new approach was compared to the basic heuristic still using the flow shop model of the problem, as it was the environment the meta-heuristic was designed for. The results showed that, once again, the iterated greedy method was able to generate solutions that had a significantly smaller total production time than the sequences created by the basic heuristic.

In the last test, the same two methods were put against each other in a simulation model that assessed the sequences using all characteristics of the problem, such as shifts and previous local rules. Once more, the newly implemented strategy achieved the best results amongst the two by creating a sequence that yielded 29% higher productivity levels.

Overall, the iterated greedy with local search meta-heuristic was the method that achieved the best results, despite only considering an adaptation of the problem and one main characteristic, the setup times, when creating new solutions. In contrast, the basic heuristic followed a set of rules that took into account setup times and distributed the load through the available workstations but did not check when operations began and ended at the machines, which ultimately led to less satisfactory productivity values.

5.2 Future Work

To further verify the applicability of the implemented algorithm at the factory, one step that could be taken would be to gather a second set of production data and rerun the simulation model that assesses the productivity of a given sequence.

One other strategy that could be developed would have to do with the objectives of the meta-heuristic. Whereas in this case study there is a single goal, to maximise productivity through the reduction of makespan, there are other relevant factors to production, such as minimising delays on deliveries. For that reason, a multi-objective outlook of the problem could also be of interest.

Last but not least, the iterated greedy with local search meta-heuristic could also be coupled with a digital twin of the facility, which provides a live view of the production floor. This combination would allow the latter to send real-time optimisation requests to the meta-heuristic, based on current events, and increase its flexibility.

References

[1] Rubén Ruiz and Thomas Stützle. An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143–1159, 2008. doi:10.1016/j.ejor.2006.07.029.

[2] Stéphane Dauzère–Pérès and Jean–Bernard Lasserre. On the importance of sequencing decisions in production planning and scheduling. *International Transactions in Operational Research*, 9(6):779–793, 2002. doi:10.1111/1475-3995.00388.

[3] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5(C):287–326, jan 1979. doi:10.1016/S0167-5060(08)70356-X.

[4] Peter Brucker. *Scheduling algorithms*. Springer, 5 edition, 2007.

[5] Rinnooy Kan A. H. G. *Machine scheduling problems: classification, complexity and computations*. Nijhoff, 1976.

[6] Satyasundara Mahapatra, Rati Dash, and Sateesh Pradhan. *Heuristics Techniques for Scheduling Problems with Reducing Waiting Time Variance*. 08 2017. doi:10.5772/intechopen.69224.

[7] Rubén Ruiz, Concepción Maroto, and Javier Alcaraz. Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. *European Journal of Operational Research*, 165(1):34–54, 2005. doi:10.1016/j.ejor.2004.01.022.

[8] Ali Asghar Rahmani Hosseinabadi, Javad Vahidi, Behzad Saemi, Arun Kumar Sangaiah, and Mohamed Elhoseny. Extended Genetic Algorithm for solving open-shop scheduling problem. *Soft Computing*, 23(13):5099–5116, 2019. doi:10.1007/s00500-018-3177-y.

[9] Günther Zäpfel, Roland Braune, and Michael Bögl. *Metahuristics Search Concepts*. 2010.

[10] Michael L. Pinedo. *Scheduling Theory, Algorithms, and Systems*. Springer, 5 edition, 2016. doi:10.1007/978-3-319-26580-3_5.

[11] Nicolau Santos, Rui Rebelo, and Joao Pedro Pedroso. A tabu search for the permutation flow shop problem with sequence dependent setup times. *International Journal of Data Analysis Techniques and Strategies*, 6(3):275, 2014. doi:10.1504/ijdats.2014.063062.

[12] Jabrane Belabid, Said Aqil, and Karam Allali. Solving permutation flow shop scheduling problem with sequence-independent setup time. *Journal of Applied Mathematics*, 2020:1–11, 2020. doi:10.1155/2020/7132469.

[13] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993. doi:10.1016/0377-2217(93)90182-M.

[14] A. Sioud and C. Gagné. Enhanced migrating birds optimization algorithm for the permutation flow shop problem with sequence dependent setup times. *European Journal of Operational Research*, 264(1):66–73, 2018. doi:10.1016/j.ejor.2017.06.027.

[15] Roger Z. Ríos-Mercado and Jonathan F. Bard. Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, 110(1):76–98, 1998. doi:10.1016/S0377-2217(97)00213-0.

[16] Xiangtao Li and Shijing Ma. Multiobjective Discrete Artificial Bee Colony Algorithm for Multiobjective Permutation Flow Shop Scheduling Problem with Sequence Dependent Setup Times. *IEEE Transactions on Engineering Management*, 64(2):149–165, 2017. doi:10.1109/TEM.2016.2645790.

[17] S. Afshin Mansouri, Emel Aktas, and Umut Besikci. Green scheduling of a two-machine flowshop: Trade-off between makespan and energy consumption. *European Journal of Operational Research*, 248(3):772–788, 2016. doi:10.1016/j.ejor.2015.08.064.

[18] R. Ramezanian, M.M. Vali-Siar, and M. Jalalian. Green permutation flowshop scheduling problem with sequence-dependent setup times: a case study. *International Journal of Production Research*, 57(10), 2019. doi:10.1080/00207543.2019.1581955.

[19] R. Linn and W. Zhang. Hybrid flow shop scheduling: a survey. *Computers and Industrial Engineering*, 37(1):57–61, 1999. doi:10.1016/S0360-8352(99)00023-6.

[20] Rubén Ruiz and Concepción Maroto. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. 169:781–800, 2006. doi:10.1016/j.ejor.2004.06.038.

[21] B. Naderi, Rubén Ruiz, and M. Zandieh. Algorithms for a realistic variant of flowshop scheduling. *Computers and Operations Research*, 37(2):236–246, 2010. doi:10.1016/j.cor.2009.04.017.

[22] Quan Ke Pan, Liang Gao, Xin Yu Li, and Kai Zhou Gao. Effective metaheuristics for scheduling a hybrid flowshop with sequence-dependent setup times. *Applied Mathematics and Computation*, 303:89–112, 2017. doi:10.1016/j.amc.2017.01.004.

[23] Muhammad Nawaz, E. Emory Enscore, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983. doi:10.1016/0305-0483(83)90088-9.

[24] Alexander J Benavides. A New Tiebreaker in the NEH heuristic for the Permutation Flow Shop Scheduling Problem. 2018.

[25] Frontline Solvers. OptQuest Solver Engine for Excel. https://www.solver.com/optquest-solver-engine-excel (accessed April 23, 2021).