A Tabu Search Algorithm for a Bi-Objective Flexible Job Shop Scheduling Problem with Sequence-Dependent Setup Times

A case study

João Côrte-Real Bivar Sacramento

Thesis to obtain the Master of Science Degree in

Industrial Engineering and Management

Supervisors: Prof. Tânia Rute Xavier de Matos Pinto Varela
Prof. Nelson Fernando Chibeles Pereira Martins

Examination Committee

Chairperson: Prof. Ana Isabel Cerqueira de Sousa Gouveia Carvalho
Supervisor: Prof. Nelson Fernando Chibeles Pereira Martins
Member of the Committee: Prof. Lídia Lampreia Lourenço

November 2019

Abstract

The increasing competitivity in the plastic container market is driving companies toward a greater focus on efficiency. This dissertation focuses on Logoplaste, a Portuguese plastic container manufacturer, specifically their factory Logoplaste Santa Iria (LSI). LSI produces different products and changing production between them creates long setup times which reduce production efficiency. Therefore, there is a need for a practical scheduling tool. The goal of this dissertation is to build a model to aid LSI in production scheduling. The problem is presented in detail, describing both production and planning processes, leading to the conclusion that the production process at LSI is a flexible job shop with sequence-dependent setup times (FJSSP-SDST). A literature review is carried out to identify key aspects of the problem and the best way to solve it. Several meta-heuristics are analyzed, and their advantages and disadvantages discussed. Conclusions drawn from the literature review highlight the lack of studies of this particular problem considering the two objectives studied: total tardiness and makespan. The algorithm, which is a Multi-Objective Tabu Search, is described. Each component its components is detailed and illustrated. Two production strategies are studied: Make-to-Order and Make to Stock. The tuning of parameters is presented, and the instances used to test the algorithm are described.

The results show that the MOTSA is viable for the FJSSP-SDST and can be applied to either production strategy with good results.

Keywords: Meta-Heuristics, Tabu Search, Flexible Job Shop Scheduling Problem, Sequence-Dependent Setup Times, Make-to-Order, Make-to-Stock

Resumo

O aumento da competitividade na indústria incentiva as empresas a focarem-se na eficiência. Como tal, muitas empresas estão a conduzir iniciativas para aumentar a produtividade nas operações, e a indústria da manufatura não é exceção. Esta dissertação foca-se na Logoplaste, uma empresa portuguesa que produz recipientes de plástico, especificamente na sua unidade Logoplaste Santa Iria (LSI). Esta fábrica produz vários produtos e trocar a produção entre eles gera tempo de preparação grandes, reduzindo a eficiência da fábrica. Assim, torna-se clara a necessidade de uma ferramenta prática para o escalonamento da produção.

O objetivo desta dissertação é desenvolver um modelo para ajudar a LSI a escalonar a produção. O problema é apresentado descrevendo os processos de planeamento e de produção, chegando à conclusão que se trata de uma *flexible job shop with sequence-dependent setup times* (FJSSP-SDST). Uma revisão de literatura foi feita para identificar os aspetos-chave do problema e abordagens para o resolver. Várias meta-heurísticas são analisadas e as suas características discutidas. As conclusões da revisão de literatura identificam a falta de estudos sobre este problema considerando os objetivos estudados: *total tardiness* e *makespan*. Cada componente do algoritmo, um Tabu Search Multi-Objectivo, é detalhado e ilustrado. Duas estratégias de produção são analisadas, *Make-to-Order* e *Make-to-Stock.* O afinamento dos parâmetros é apresentado e as instâncias utilizadas para testar o modelo são descritas.

Os resultados mostram que o algoritmo desenvolvido é viável para o FJSSP-SDST e pode ser aplicado a qualquer das duas estratégias de produção rapidamente e com bons resultados.

Keywords: Meta-Heuristicas, Tabu Search, Flexible Job Shop Scheduling Problem, Sequence-Dependent Setup Times, Make-to-Order, Make-to-Stock

Table of Contents

1. Introduction 1

1.1. Problem Contextualization 1
1.2. Problem Objectives 2
1.3. Dissertation Structure 3

2. Problem Description 5

2.1. Logoplaste 5
2.2. Production Process 6
2.3. Generic Problem Definition 8
2.4. Production Planning 10
2.5. Conclusions 12

3. Literature Review 13

3.1. Production Scheduling 13
3.2. Flexible Job Shop Scheduling Problem 16
3.3. Production Scheduling Algorithms 18

3.3.1. Exact Approaches 18
3.3.2. Meta-heuristics 19

3.3.2.1. Mono-Objective Local Search-Based Meta-Heuristics 19
Mono-Objective Population-Based Meta-Heuristics 26
3.3.2.2. 26
3.3.2.3. Mono-Objective Hybrid Algorithms 28
3.3.2.4. Multi-Objective Meta-heuristics 30

3.4. Concluding Remarks 32

4. Algorithm Characterization 33

4.1. Multi-Objective Tabu Search Algorithm (MOTSA) 35

4.1.1. Problem Initialization 35
4.1.2. Initial Solution 37
4.1.3. Neighbor Functions 39

4.1.3.1. Total Tardiness 41
4.1.3.2. Makespan 43
4.1.3.3. Neighborhood Generation 44

4.1.4. Tabu List 45

4.1.4.1. Assignment Tabus 45
4.1.4.2. Sequencing Tabus 46

4.1.5. Diversification Strategies 47
4.1.6. General Framework 48
4.2. Production Strategies 52
4.2.1. Make to Stock 52
4.2.2. Make to Order 53
5. Model Tuning and Results Analysis 55
5.1. Model Inputs 55
5.2. Parameter Tuning 56
5.3. Problem Instances 61
5.4. Results Analysis 62
6. Conclusions and Future Work 67
Bibliography 69

List of Tables

Table 1 - Classification Dimensions of a scheduling problem 14
Table 2 - Example processing times (minutes) 36
Table 3 - Example setup times (minutes) 36
Table 4 - Example orders (due date minutes) 37
Table 5 – Example stock levels 53
Table 6 – Example set of orders generated for the MTS production strategy 53
Table 7 - Summary table of neighborhood size sensitivity analysis 58
Table 8 – Neighborhood size sensitivity analysis for neighborhood size = **3** 59
Table 9 – Tabu List size sensitivity analysis results 60
Table 10 – Tuned Model Parameters 60
Table 11 – Instance descriptions 61
Table 12 – Summarized results for all instances with the Symmetrical Setup Times (MTO) 63
Table 13 - Summarized results for all instances with the Asymmetrical Setup Times (MTO) 64
Table 14 - Summarized results for all instances with the Symmetrical Setup Times (MTS) 64
Table 15 - Summarized results for all instances with the Asymmetrical Setup Times (MTS) 65

List of Figures

Figure 1 – Injection molding machine components 6
Figure 2 - Gantt chart of a solution to a 3 jobs, 3 machines problem. 8
Figure 3 - Disjunctive graph representation of a 3 jobs, 3 machines problem 9
Figure 4 - Single stage, one processor (left) vs single stage, parallel processors (right) 14
Figure 5 - Single stage, one processor (left) vs single stage, parallel processors (right) 15
Figure 6 – Flow Diagram of the Initial Solution generation algorithm 38
Figure 7 – Comparison of the quality of solutions using a randomly generated initial solution vs the MOTSA Initial 39
Figure 8 -Gantt Chart of the Initial Solution for the Example Problem Data (time in minutes)............ 40
Figure 9 – Gantt Chart of a neighbor generated through an assignment move (time in minutes)............ 40
Figure 10 – Gantt Chart of a neighbor generated through a sequencing move (time in minutes)............ 40
Figure 11 – Gantt Chart of a candidate solution illustrating an assignment Tabu Status. 45
Figure 12 - Gantt Chart of a candidate solution illustrating a sequencing Tabu Status. 46
Figure 13 - Gantt Chart of a solution generated by a random swap (swapped solutions are highlighted in green. 47
Figure 14 – Flowchart of a macroscopic view general framework of the MOTSA 48
Figure 15 – Flowchart of a tardiness optimization stage’s iteration 49
Figure 16 – Flowchart of a makespan optimization stage’s iteration 50
Figure 17 – Flow Diagram of the general framework of the MOTSA 51
Figure 18 – Flowchart of parameter tuning process 56
Figure 19 - Evolution of Objective function values over time (Neighborhood Size = 5) 59
Figure 20 - Comparison of each execution's results with the average results for the Case Study instance in the MTO strategy 63
Figure 21 - Comparison of each execution's results with the average results for the Case Study instance in the MTS strategy 65

List of Abbreviations

FJSSP	– Flexible Job Stop Scheduling Problem
GA	– Genetic Algorithm
HDPE	– High Density Polyethylene
JSSP	– Job Shop Scheduling Problem
LSI	– Logoplaste Santa Iría
MILP	– Mixed Integer Linear Programming
MIP	– Mixed Integer Programming
MTO	– Make to Order
MTS	– Make to Stock
OR	– Operations Research
PET	– Terephthalate
PP	– Polypropylene
SDST	– Sequence Dependent Setup Times
TS	– Tabu Search

1. Introduction

1.1. Problem Contextualization

In an increasingly competitive industrial environment, companies must try to maintain a competitive advantage over their competitors in order to survive and bring benefits to their stakeholders. One of the ways to remain competitive is to reduce operating costs by increasing the efficiency of operations, to be able to offer their products and services at a lower price, or with a bigger profit margin and bring better results. To this end, much of the focus in the management of manufacturing companies goes toward manufacturing planning and control. Thus, there is a need for a methodology which helps increase efficiency to reduce costs and become more competitive.

The plastic container industry is a mature and highly competitive market and, as such, price is a key factor for success. In a market which offers low unitary profit margins, economies of scale must be taken advantage of wherever possible and production efficiency is of extreme importance. This dissertation will focus on Logoplaste, a Portuguese company which produces plastic containers. The plastic containers are made through plastic injection molding and Logoplaste Santa Iria (LSI), the factory that this dissertation will be focusing on, has multiple machines available to manufacture their products. Different machines have different cycle times, throughput and are assigned different molds. To make the most use of every machine, it's important to allocate orders to each machine efficiently, as well as sequencing orders for each machine. This is known in OR as a scheduling problem and is one of the most common, and difficult, problems.

Operations Research (OR) as an activity is widely used in businesses and industries. The development of computers and increases in computational power greatly contributed to the advances in OR as the problems considered were typically too complex to solve without the aid of a computer. Naturally, organizations heavily invested into the use of computers, not only with the purpose of doing OR but also to use IT systems which support all kinds of managerial activities. The digitalization of activities allowed the gathering of large amounts of data which can be used in optimization models to effectively improve operations in the organization. The focus of this dissertation is the use of OR techniques in manufacturing systems, namely production scheduling, which is one of the biggest areas of interest in OR.(Hillier 2001)

LSI's biggest client is FIMA and, as such, scheduling production of their orders is key. Thus, this dissertation aims to develop a production scheduling model to aid Logoplaste in scheduling their manufacturing operations to minimize completion time and tardiness of FIMA's orders according to LSI and FIMA's contractual requirements.

1.2. Problem Objectives

This section deals with defining the objectives of this dissertation, aligning the company's objectives with the structure of the present document and the methodology to be followed in the dissertation.

1. First, a description of the problem’s general context is given, followed by a description of the company's background, context in which this problem arises, and objectives.
2. Define the company's needs in order to be able to clearly identify objectives. The company is looking to develop a model to schedule production with the goals of:
   - Reducing the total production time;
   - Eliminate, or reduce, tardy delivery of orders.
   - Helping decide between a Make to Order (MTO) and a Make to Sell (MTS) approach.
3. Characterize and define the specific combinatorial optimization problem to be solved according to the company’s operation. This section is very important, since it serves as a guideline to the following section, the literature review, where past and recent approaches to similar problems are studied.
4. Literature review of the past and recent developments in solving the problem at hand to understand the best approaches to this problem. This chapter comprises of several sections identifying the problem present followed by a separate section for each type of approach, namely:
   - Exact algorithms;
   - Local search-based meta-heuristics;
   - Population-based meta-heuristics;
   - Hybrid meta-heuristics;
   - Multi-objective meta-heuristics;
5. After the literature review, a gap in research should be identified. The dissertation should focus on this gap to develop a model which fills it. In doing so, it attempts to bring something new to the analysis of this type of problem.
6. Draw conclusions on the algorithm to be implemented in the dissertation, based on the methods studied and the gap identified.
7. Delineation of the methodology to follow in the dissertation.

1.3. Dissertation Structure

This dissertation’s structure will be as follows:

1. This first chapter focuses on describing the problem’s context as well as this dissertation’s goals and structure.
2. The second chapter presents the problem characterization, namely company description, contract details and manufacturing process.
3. In the third chapter, a review of literature is conducted on the available literature regarding the FJSSP to identify different problem formulations as well as algorithms and mechanisms that are better for solving this problem.
4. The fourth chapter focuses on characterizing the algorithms developed.
5. The fifth chapter discusses the tuning of parameters.
6. The sixth chapter will present the results of the model on several instances for a thorough test.
7. The sixth chapter presents conclusions of this dissertation.

2. Problem Description

This chapter describes the characteristics of the problem. The structure of the chapter will be as follows: first, an introduction to Logoplaste will be done to understand the context in which the company appeared, evolved and currently operates in. The production process is described, and the production planning process is reviewed to understand the basis in which the factory managers currently schedule the production. Lastly, some final thoughts are laid out to summarize the key characteristics of the problem to better guide the literature review.

2.1. Logoplaste

Logoplaste is a Portuguese company founded in 1976. They specialize in plastic containers made of polyethylene terephthalate (PET), polypropylene (PP) and high-density polyethylene (HDPE). They quickly grew with big customers such as Nestlé, Yoplait and Coca-Cola to become market leaders. Fast expansion allowed Logoplaste to begin operating abroad and Logoplaste now operates in 16 countries with 63 factories, being the fifth biggest plastic bottle producer in Europe. They also pioneered and currently lead the 'Hole in the Wall' concept, which consists of setting up their production facilities within or near the clients' own production facilities. This makes the company much more competitive and effective as the lead times are considerably reduced and healthy long-term partnerships are developed with clients.

The various factories report to the headquarters which are in Cascais. The headquarters coordinate the various factories since Logomolde (the unit responsible for producing, testing, and performing maintenance on molds) and the Innovation Lab (the unit responsible for the design and development of prototypes) are both in the headquarters' complex.

This dissertation is concerned with one of Logoplaste's unit, Logoplaste Santa Iría (LSI). This factory was setup to serve FIMA, a company that belongs to the Unilever Jerónimo Martins Group. LSI is in the same industrial complex as FIMA and supplies FIMA with PP containers for dairy products and other food products such as butter, margarine, chocolate spread, etc. Recently, LSI started to produce for Lactogal as well, a company which produces dairy products.

2.2. Production Process

The products manufactured at LSI are made through Injection Molding. In this production process, first, the material from the hopper is fed into the barrel. Due to the screw's rotation, the material moves along the barrel through the various heating sections, progressively heating to the desired temperature. It should be noted that the screw's diameter increases along the length of the screw to reduce the thickness of the material layer, increasing pressure and making it easier to achieve a homogeneous mixture.

Once enough material is at the desired temperature and viscosity, the screw plunges forward and pushes the material through the injection nozzle into the mold cavity until it is filled, when the screw comes back and material flow through the nozzle is stopped.

Once the mold cavity is filled, it is cooled by with a water-cooling system. Once the material has cooled enough for the product to achieve a solid state, the movable platen returns to its original position and the ejector pins push the product out.

Once the part has been taken out of the mold cavity, the clamp closes the mold again and the screw tip lunges forward to inject the next product's material into the mold.

The typical injection molding machine is shown in Figure 1.

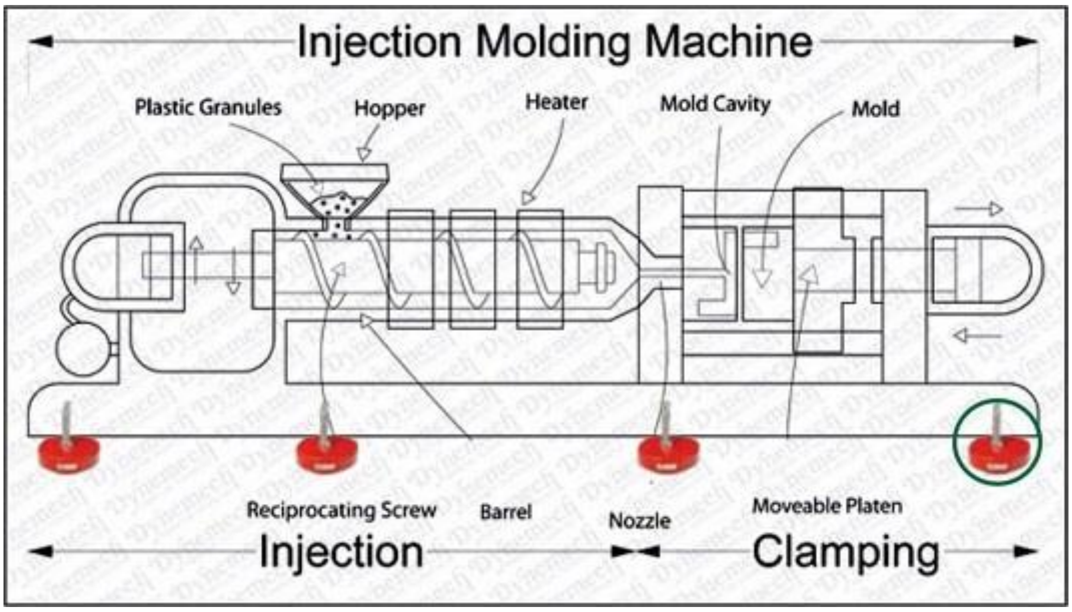


Figure 1 – Injection molding machine components

In LSI's case, the material in the hopper is polypropylene (PP). There are multiple dyes used depending on the final product's color. The products' labels are placed in the mold head before the injection through air compressors and are kept in place through static electricity. When the PP is injected, the label is fused with the PP and thus is incorporated in the final product. The labels are placed simultaneously with the removal of the finished parts from the cavity.

Operators place the finished products in boxes and then in pallets. Pallets contain only one reference and are stored in the warehouse for later shipping. The warehouse operates along a First In First Out (FIFO) strategy, meaning the first products shipped are the first products produced. Production quantities are always a certain number of pallets.

Given that LSI's client produces food products, hygiene is a major concern and, as such, LSI constantly monitors for bacteria on their products.

Furthermore, molds are of extreme importance to guarantee production continuity as well as product quality. For this reason, LSI has three molds of each type: one for production, one undergoing maintenance and one standing by. It should be noted that molds can have 1, 2 or 4 cavities.

Since this work concerns only the scheduling of production for FIMA's orders, this dissertation will only consider the resources and process for FIMA's orders.

LSI has 8 machines dedicated to FIMA’s orders. In total, LSI manufactures 30 different product references for FIMA, corresponding to 15 bottoms and 15 covers.

Bottoms and covers are different references, so they are manufactured separately. Some products share the same mold and even the same color, however the labels are different for each reference. Switching between product reference implies setup times. This is the time required to shift from one reference to another. There are three types of components to be switched and, as such, three types of setup (molds, dyes and labels). Molds take the most time, taking up to 5 hours (it takes longer the more cavities the mold has), followed by dye, which takes about 30 minutes, and labels, which take only a few minutes.

The problem data is presented in Appendix 1.

2.3. Generic Problem Definition

Given the characteristics described, it is concluded that this problem is a Job Shop Scheduling Problem.

In the JSSP, a set J of jobs $J = \{J_1, \dots, J_j, \dots, J_N\}$ jobs is to be processed on a set M of machines $M = \{M_1, \dots, M_k, \dots, M_M\}$. A job is an ordered subset of a set O of operations $O = \{O_1, \dots, O_i, \dots, O_n\}$. Jobs have due dates, denoted d_j. Precedence constrains are defined for the operations so that, if an operation O_i is to be performed before O_j, then O_j can only start after O_i has been completed. Operations are processed by specified machines, as such:

$$M(O_i) = M_k$$

taking p_{ik} time units to process. A machine can only process one operation at a time, and a job can only be processed by one machine at a time. Most research assumes operations to be non-preemptive, that is, once a certain operation is started, it cannot be interrupted and then completed later. The flexible job shop is an extension in the sense that, for a certain operation O_i, there are r machines available (Brandimarte 1993).

Let C_i be the completion time of job J_i. The minimize the two objective functions (makespan and total tardiness), i.e.

$$min(\max_{i=1,\dots,n} C_i)$$

$$min(\sum_{n=1}^{N} C_i - d_i)$$

by determining the operation sequence in each machine.

Earlier representations of the JSSP's solution would typically be a Gantt Chart such as the one seen in Figure 2. The x axis represents time, the y axis represents machines and tasks T_{ij} represent the j^{th} operation of job J_i. Note that this Gantt Chart refers to a problem where jobs can be composed of multiple tasks. The products made in LSI, however, are all made through a single-stage process.

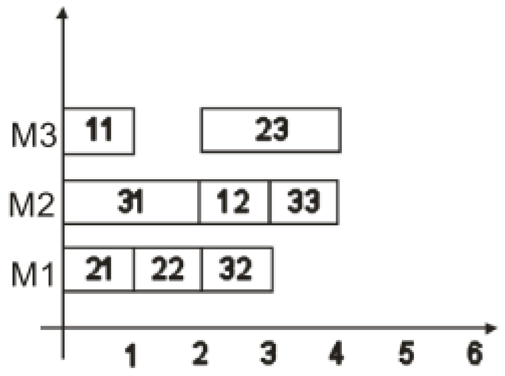


Figure 2 - Gantt chart of a solution to a 3 jobs, 3 machines problem.

While the Gantt Chart provides a good visual representation of the problem, the most common representation is the disjunctive graph model, introduced in (Roy and Sussman 1964).

The disjunctive graph model is a directed graph $G = \{ V, A, E \}$, where:

- V is the set of vertices such that $V = O \cup \{0, n+1\}$, where operations 0 and $N+1$ are dummy operations.
- A is the set of conjunctive arcs which define precedence constrains and connects consecutive operations of the same job, as well as the dummy operations and the first and last operations of each job. The weight of each arc indicates the processing time p_i of that operation.
- E is the set of disjunctive arcs which connect any two operations assigned to each machine.

Figure 3 represents the disjunctive graph of a 3 jobs, 3 machines problem with 8 operations. The weights of undirected edges are not represented in the figure for simplicity.(Blazewicz, Pesch, and Sterna 2000)

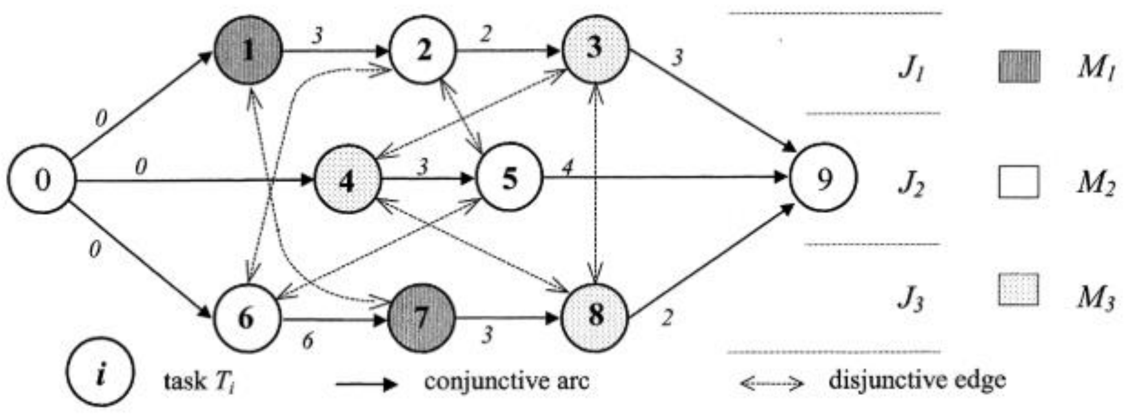


Figure 3 - Disjunctive graph representation of a 3 jobs, 3 machines problem

A feasible schedule corresponds to a directed acyclic graph and its makespan is equal to the length of the longest path in the graph from O_0 to O_{n+1}.

Different models emerge from this representation where notation is changed to reduce computational requirements and enable different solution methods.

2.4. Production Planning

Production planning is the process of allocating resources to plan the manufacture of a given set of products. Production planning is typically done on three different levels:

- Strategic level, which is concerned with long-term decisions such as factory location and capacity, productive technology, quality, etc. These decisions usually contemplate variables to be considered over several years.
- Tactical level, which is concerned with medium-term decisions, such as annual capacity and resource requirements.
- Operational level, which is concerned with short-term decisions such as monthly or weekly scheduling of capacity and resources.

Furthermore, in each of these levels the production planning process can deal with different dimensions, such as materials requirements planning (MRP), master production schedule (MPS), enterprise resource planning (ERP), production scheduling and manufacturing resource planning (MRP II). In this dissertation, we will focus on the operational level of production scheduling.

As mentioned before, FIMA produces dairy and other food products, which have a short expiry date. Because LSI is in the same industrial complex as FIMA, lead times are low which enables FIMA to operate on a low inventory of containers. The short expiry dates drive FIMA to operate reliant on actual demand as opposed to forecasted demand and to schedule their production to the latest due date to avoid having finished product expire in the warehouse. For these reasons, FIMA is prone to changing the orders placed on LSI. Thus, it's important for LSI to be able to adapt quickly to variations in demand.

We can see how this type of operation can harm both factories, as failed supply from LSI to FIMA can disrupt FIMA's production, and abnormal orders can cause LSI to have excessive stock of certain products and greatly increase setup times, reducing overall efficiency.

In the beginning of the year, FIMA sends LSI an annual demand plan. This plan is used at LSI to generate an annual production plan which helps managers plan capacity and materials requirements. Every week a delivery plan is sent to LSI to fulfill, along with an updated annual demand plan.

To reduce the probability of production disruption, LSI and FIMA established safety stock levels at both LSI and FIMA, which are higher for LSI. These stock levels are set per reference and are updated weekly in the weekly delivery plan.

Product labels can be updated by FIMA at any time due to new design or ingredients. Should this happen, all stock currently kept of that product reference is thrown away, and FIMA pays for that stock, provided it's within the established limits. Thus, the stock levels have a maximum threshold which acts as a safeguard for LSI.

Operationally speaking, LSI operates on a week-to-week basis. It's important to describe the planning steps each week. Considering that the planning activity starts at week 0, the process is as follows:

- Day 1: LSI receives the weekly delivery plan for week 1. Note that this weekly plan includes stock levels;
- Day 2: LSI receives the updated annual demand plan for week 1; LSI then updates their own annual production plan based on the weekly delivery plan, the annual demand plan from FIMA and the stock levels;
- Days 3 and 4: Production scheduling for each machine is done from weeks 0 to n (n depends on the stock levels and inventory policies);
- Day 5: LSI receives the updated weekly delivery plan for week 1 with updated quantities.

This process is done manually by a team of schedulers at LSI without use of any software. Manually scheduling this production process is very time intensive and ineffective, because there are many jobs and many machines to consider, along with the setup times needed to switch between jobs. Furthermore, this scheduling is done weekly with data from the delivery plans sent by FIMA, along with stock information at LSI, which must be cross-checked. This typically results in unnecessarily long setup times which increases total production time and may cause tardiness. Also, the schedulers could be performing other useful operations in the meantime, which would represent a productivity gain parallel to those of the schedule itself.

2.5. Conclusions

LSI has a relatively simple production process, featuring a single production stage for all of their products. There are multiple machines available for certain operations, and there are multiple different operations to be performed. There are three types of changeovers, namely for molds, labels and dyes. As such, the setup times for each product depend on the sequence of jobs being processed on that machine. Because production quantities are always full pallets of product, production at LSI is a batch process. This means that both processing times and production quantities are fixed for each job.

However, the planning process is complex due to the contractual requirements set alongside FIMA and is critical for the efficiency of the factory, as poor planning can cause high setup times and disrupt not only LSI's production, but FIMA's as well.

The planning process is currently done manually which is extremely inefficient and ineffective. For this reason, LSI would benefit greatly from a model capable of receiving the necessary information from the weekly and annual plans, as well as the machine and product specifications, and returning a production schedule for the week in question. Such a model must be relatively easy to implement and work with on a weekly basis. It should also accept updates to information such as product references and stock levels.

The high setup times, which sometimes occur during production of a set of orders, can have great impact in the makespan which, in turn, may cause tardiness.

It should be noted that, given that LSI has a supply contract with FIMA, the products' due dates must be given great importance and, as such, no tardiness should be allowed, though exceptions can be made if there are significant improvements in the makespan at the cost of a slight increase in tardiness. As such, a multi-objective analysis considering both criteria, makespan and tardiness, should be beneficial and worthwhile to implement.

Given the problem at hand, it is convenient to develop a scheduling model and algorithm to aid LSI in scheduling its production given the orders from FIMA with the goals of minimizing the total makespan and total tardiness. Taking these requirements into account, the following chapter will focus on reviewing the available literature on this type of problem, with the intent of identifying the work that has been done on the subject and filling the gaps found in said review. First, the specific type of scheduling problem present will be identified and defined. Then, several different algorithms will be analyzed to understand the algorithms developed so far and their advantages and disadvantages, namely exact algorithms, meta-heuristics, hybrid algorithms and a section on multi-objective algorithms.

3. Literature Review

The aim of this chapter is to bring an overview of production scheduling concepts, principles and approaches developed so far. In the end of the chapter, a global understanding of the problem, as well as knowledge of efficient and effective methods to solve it are identified. The structure of this chapter will be as follows: first, a description of a combinatorial optimization problem is given; then the specific type of scheduling problem present is identified; the main algorithms for solving it are presented, while trying to identify their main advantages and disadvantages; lastly, the most promising approach to solve the problem will be determined.

Since production scheduling is one of the main problems being studied in the combinatorial optimization field, there is plenty of literature focusing on it, using both exact and meta-heuristic approaches. However, each article seems to solve a different case, making it difficult to directly compare algorithms. Nevertheless, we can still infer on useful techniques for each algorithm as well as parameter tuning.

3.1. Production Scheduling

A combinatorial optimization problem is a problem, in which the objective is to minimize a given function $f(s)$ where s is a solution belonging to the solution space X, such that:

$$f(s^*) = \min f(s)\,, s \epsilon X$$

X is defined by a set of constraints such as precedence, priority, capacity, due dates, etc. The objective function f can be completion time, total cost, tardiness, etc.

A scheduling problem is a resource allocation and operation sequencing problem, a subset of combinatorial optimization problems where resources, typically equipment, labor, raw materials and utilities, must be allocated to a set of operations to be completed while optimizing a given criterion.

Scheduling problems may be diverse regarding the types of constraints needed to model the problem and objective functions. (Graves 1981) identifies three dimensions for classifying a scheduling problem: requirements generation, processing complexity and scheduling criteria. Table 1 depicts the three dimensions considered and offers a brief explanation of each.

Requirements generations distinguishes between an open shop and a closed shop. In an open shop, no inventory is kept, so production orders are determined only by customer request, so it is a sequencing problem in which operations must be sequenced on machines or facilities. Open shop systems are typically referred to as Make to Sell (MST). In closed shop problems, inventory is used to satisfy customer orders, so production orders are a function of inventory management decisions, thus scheduling in a closed shop deals with lot sizing as well as sequencing. Closed shop systems are usually referred to as Make to Stock (MTS).

Processing complexity regards the number of operations associated with each job. The following breakdown can be made:

- Single stage, one processor
- Single stage, parallel processors
- Multistage, flow shop
- Multistage, job shop

Table 1 - Classification Dimensions of a scheduling problem

Classification Dimensions	Description	Characterization
Requirements generation	Is concerned with the how the factory fulfills customer requests	• Open shop (MTO) vs Closed shop (MTS)
Processing complexity	Is concerned with the number of tasks in each job as number of machines available for each task	• Single stage vs Multistage • One Processor vs Multiple Processors
Scheduling criteria	Is concerned with the type of criteria to be optimized in the problem	• Schedule Performance • Schedule Cost

The single stage, one processor problem is the simplest form of production scheduling, where all operations can be performed on a single processor (machine).

The single stage, parallel processors problem is an extension of the latter case, in which operations can be processed by any of the parallel processors. A visual representation of these cases can be seen in Figure 4.

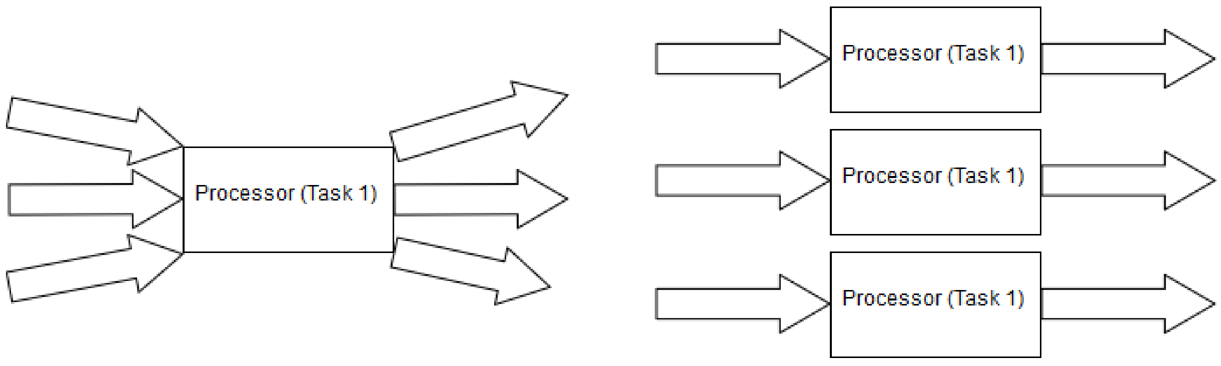


Figure 4 - Single stage, one processor (left) vs single stage, parallel processors (right)

In the multistage problems, each job is processed on a series of processors, usually with strict precedence orders.

In the flow shop problem, jobs are processed on a set of machines with identical precedence restrictions. A generalization of the flow shop problem is the flexible flow shop problem in which, for every processing step, there are multiple machines available.

The job shop problem, however, has no restrictions on processing steps ordering for a given job, and jobs can be alternatively routed as desired. For this reason, the job shop problem is the more general one and, as such, the breakdown presented is considered sufficient to address most processing environments. The job shop problem is the most general case of the open shop problem and thus the most difficult. A generalization of the job shop problem is the flexible job shop scheduling problem (FJSSP) in which there are multiple machines available to process one or many operations.

Figure 5 is a visual representation of the difference between a flowshop and a jobshop.

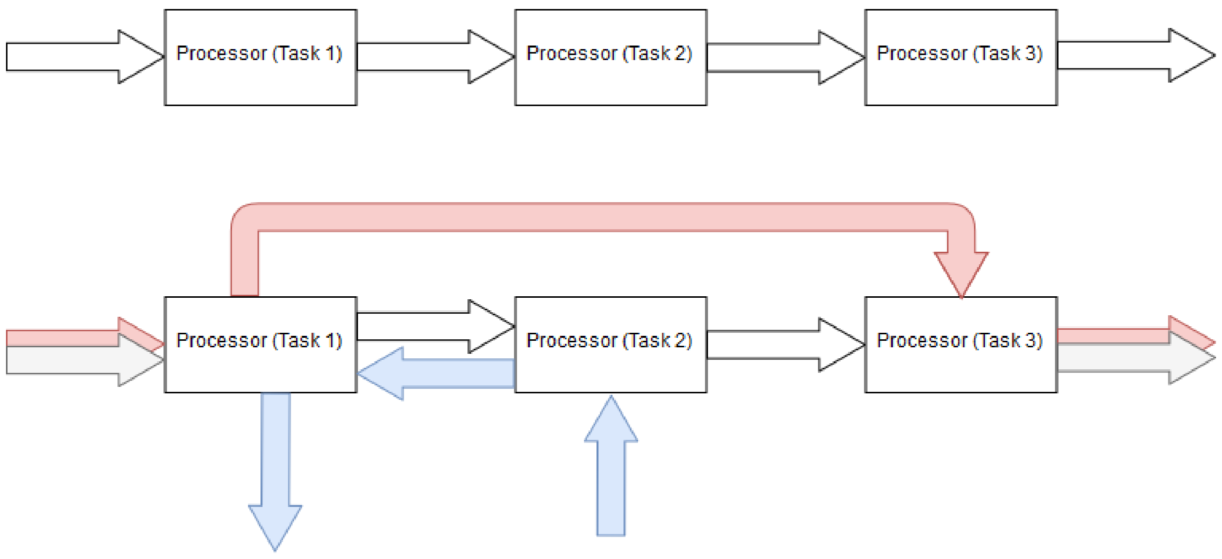


Figure 5 - Single stage, one processor (left) vs single stage, parallel processors (right)

Scheduling criteria can be divided into two categories, namely schedule cost and schedule performance. The first is mostly used in closed shop problems whereas schedule performance is the most used criteria type in open shop problems. Schedule cost should include fixed and variable manufacturing costs, inventory and stock-out costs and system costs for defining and monitoring the schedule. Schedule performance is usually measured by one of the following criteria, all of which are to be minimized.

- Resource utilization level, such as equipment or materials usage.
- Tardiness, which is the positive difference between a job's completion time and its due time.
- Flow time, which is the time between completion of a job and the time since when it was available to be processed.
- Makespan, which is the maximum completion time for a set of jobs, that is, the time between the start of the first job and the completion time of the last job.

Two further dimensions could be considered: the nature of the requirements specification, which can be either deterministic or stochastic, and the scheduling environment, which can be static or dynamic. Though most scheduling problems are stochastic and dynamic, due to the uncertain nature of processes and the environment, models built for stochastic and dynamic problems are usually deterministic and static. This makes the modelling of the problem simpler and easier to apply, while still producing applicable results.

Another important distinction to make in production systems is regarding the production type, i.e. batch or continuous. The first one relates to processes which produce final product in lots and have a finite duration for a given quantity produced. Continuous production regards processes which flow continuously without interruption, except when maintenance is required (Lee et al. 2015).

Furthermore, scheduling problems can be classified according to the production process's setup times, as setup times can be a highly influential factor when setup actions, e.g. cleaning, or changeovers, e.g. swapping material or mold, are required to complete a given schedule. Setup times can either be Sequence Dependent or Sequence Independent. Sequence dependent setup times are present whenever a machine's processing time for a given job depends on both the job previously processed and the job to be processed immediately after. Sequence Independent setup times are present when the setup time for a given job is the same, for example, when performing cleaning between two batches of the same product.

According to the description of the problem laid out in the previous chapter and the problem classification presented above, it is concluded that the scheduling problem present is a Flexible Job Shop Problem with Sequence Dependent Setup Times, referred to in the literature as FJSSP-SDST. For this reason, the review of literature focuses on this problem type as much as possible. However, it is also convenient to look at classical job shop scheduling problems to better understand the evolution of job shop scheduling algorithms.

3.2. Flexible Job Shop Scheduling Problem

The Job Shop Scheduling Problem is one of the most widely studied optimization problems and is known to be NP-hard, meaning there is no efficient algorithm to solve it in polynomial time, except for very small instances, such as a problem having only $m \leq 2$ machines. For this reason, the best option to solve the JSSP is using heuristic methods, which excel in finding good solutions, even if sub-optimal, in reasonable running time.

The FJSSP-SDST is even harder to solve than the classical JSSP, as it is an extension of the latter, the difference being the flexibility and the sequence-dependent setup times, which add to the complexity of the problem (Garey, Johnson, and Sethi 1976).

Since each stage of a FJSSP has multiple available machines, it consists of two sub-problems: the assignment problem, which deals with assigning jobs to machines, and the scheduling problem, which deals with the sequence in which jobs are processed on each machine. With respect to these two sub-problems, two main approaches can be distinguished (Brandimarte 1993):

1. Concurrent: the assignment and scheduling problems are solved simultaneously.
2. Hierarchical: the assignment and scheduling problems are solved separately to reduce the complexity of each problem being solved. In the FJSSP, this is the approach more commonly used. It should be noted that, once the machine assignments have been determined, the problem being solved is a classical JSSP.

Further classification can be made regarding information flows in a hierarchical approach: in a one-way approach, the higher level problem is solved first, followed by the lower level; in a two-way approach, the higher level problem is solved, followed by the lower level problem, and the lower level solution gives some insight into the higher level problem, so it can be solved yet again.

Before delving into the literature available on this problem, a formal definition of the general JSSP is presented to provide a better basis for understanding the approaches discussed.

3.3. Production Scheduling Algorithms

This section focuses on a review of the literature available on JSSP algorithms. The different types of algorithms available will be identified, characterized and their advantages and disadvantages pointed out. As mentioned before, there are two broad categories of algorithms to choose from: exact algorithms and non-exact algorithms. The first category refers to approaches which solve a given problem to optimality while the second refers to algorithms which provide good solutions within reasonable time.

3.3.1. Exact Approaches

Exact methods were the first approach used to solve the job shop scheduling problem, and the most common techniques used are Branch and Bound (B&B), Mixed-Integer Programming (MIP), Mixed-Integer Linear Programming (MILP) and Constraint Programming (CP) (Graves 1981).

Both (Bowman 1959) and (Wagner 1959) developed Mixed-Integer Linear Programming models for the JSSP makespan minimization, using different formulations, time-indexed and rank-based, respectively, concluding that these models are capable of solving such problems to optimality, however not in reasonable or economically viable time. (Manne 1960) also developed a Mixed Integer Programming model based on the disjunctive formulation of the problem.

(Liao and You 1992) proposed an alteration to the disjunctive model, however it was recently showed to be inferior to the original formulation of (Manne 1960), which remains the best MIP formulation for the JSSP, in a computational study of MIP models done by (Ku and Beck 2016). This study also showed that Constraint Programming (CP) approaches outperform MIP models, and that for large instances, CP models can be competitive with the high performing hybrid CP-TS algorithm of (Beck and Feng 2011).

(B. J. Lageweg, J. K. Lenstra 1977) presented a combination of B&B and implicit enumeration techniques to solve the JSSP with respect to makespan minimization. This algorithm reportedly solved a 6 jobs, 6 machines instance of JSSP to optimality.

(J. Carlier 1989) were the first to solve a known 10 jobs and 10 machines problem instance to optimality by use of a B&B algorithm based on the disjunctive graph model.

(Applegate and Cook 1991) developed an efficient B&B algorithm which was able to solve their famous 10 jobs 10 machines problem quite quickly. (Brucker et al. 1994) also accomplished this feat as well and improved known solutions to other benchmark instances.

(Artigues and Feillet 2008) developed a B&B algorithm to solve the JSSP considering sequence dependent setup times (JSSP-SDST) with respect to makespan minimization, combined with heuristics to perform feasibility checks, improving known solutions on common benchmark instances.

(Zhu and Heady 2000) developed a MIP formulation to the JSSP-SDST with regards to the minimization of the sum of Earliness and Tardiness. They were able to solve the problem to optimality in small and medium-sized instances, noting that more efficient MIP solvers would be required to solve industrial-scale problem instances.

3.3.2. Meta-heuristics

Meta-heuristics were adopted due to their ability to find good solutions within reasonable time.

Heuristic approaches are problem dependent, as the parameters need to be tuned for each problem. These approaches often get trapped in local minima and fail to explore the entire solution space. Meta-heuristics, on the other hand, are more general and, as such, easier to apply in most problems. The term 'meta' is used to refer a higher level heuristic, that is, a meta-heuristic is a framework for heuristic applications as they can be generically applied to most problem types. However, meta-heuristics still need to be tuned to the problem they are being applied to.

Meta-heuristics can be of two types:

- **Local search meta-heuristics**, or trajectory-based, start from one feasible solution and, with each iteration, that solution is updated and may or may not provide an improvement to the objective function. This procedure is repeated until stop criteria are satisfied.
- **Population based meta-heuristics** start with a population of solutions, and each iteration updates the population. This type of meta-heuristics draws from the observations of nature to improve a population's quality.

First, Mono-Objective Meta-Heuristics are studied to understand how each objective function is tackled, and only then are Multi-Objective Meta-Heuristics studied to understand what strategies can be used to address the Multi-Objective aspect of the problem.

3.3.2.1. Mono-Objective Local Search-Based Meta-Heuristics

Tabu Search

The Tabu Search (TS) algorithm is one of the most popular local search meta-heuristics for its success in many different combinatorial optimization problems. By making use of adaptive memory and responsive exploration of the search space, the Tabu Search is able to efficiently explore the solution space without getting trapped in local minima and learning lessons from bad decisions taken in the process (Glover 1998).

Since the Tabu Search is a memory-based algorithm, it uses both short and long-term memory structures to store solutions visited during the search. The memory structures are good tools to both intensify and diversify the search. As such, it's important to make good use of them to efficiently guide the algorithm through a given a solution space and find good solutions quickly and at a low computational cost.

The tabu search algorithm can be broadly described as such:

Consider an objective function $f(x)$ and a set of solutions X. As any other local search method, in each iteration the solution x is updated until an end-condition is met. Every solution $x \in X$ has a neighborhood $N(x)$, which is the set of all solutions which can be reached from x by performing a *move*. Each iteration updates the *tabu list*, which contains forbidden solution attributes (such as arcs in the disjunctive graph model), with the last choice made. The *tabu list*'s length is usually fixed and different rules can be applied to determine which elements are dropped at each iteration. By restricting the set of moves available given a solution x, the *tabu list* effectively changes the neighborhood, making it $N^{*}(x) \neq N(x)$. Longer-term memory may also change the neighborhood $N^{*}(x)$ by considering solutions not found in $N(x)$ as a diversification tool or by encouraging the selection of previously identified elements of elite solutions as an intensification strategy.

(Dell'Amico and Trubian 1993), (Nowicki and Smutnicki 1996) and (Nowicki and Smutnicki 2005) proposed Tabu Search algorithms to solve the classical JSSP makespan minimization.

(Dell'Amico and Trubian 1993) developed a TS algorithm to solve the JSSP with the goal of minimizing the makespan, in which the feasible starting solutions are generated through a bi-directional method. This is because single-direction algorithms for generating initial solutions usually showed deteriorating quality for the final operations because these strongly depend on the first assignments. The use of a bi-directional method allowed two good 'halves' of the initial solution to be determined and the resulting complete solution was still good. The neighborhood structures used are extensions of the neighborhood structures described in (Laarhoven, Aarts, and Lenstra 1992), which will be seen later.

They used a finite size tabu list with a FIFO strategy, that is, in each iteration, an attribute is recorded and the oldest one is removed from the list. The tabu tenure, that is, the number of iterations an attribute is classified as tabu, varies depending on the current list size and whether the search is currently improving the objective function or not. A simple aspiration criterion was employed: a forbidden (tabu) move can be a candidate move if the resulting objective function value is lower than that of the best solution previously found. Randomization was introduced in this algorithm for critical decisions such as the choice between non-aspiring tabu moves or the choice of the length parameters *min* and *max*. A restarting mechanism was also introduced such that, after a certain number of iterations which do not find a new best solution, the current solution is set as the best. This article reported finding new best solutions (at the time) for many different benchmark problem instances within reasonable running times

(Nowicki and Smutnicki 1996) proposed a TS approach to the classical JSSP for minimizing the makespan. They developed a new neighborhood structure, which is effectively a subset of the neighborhood structure suggested by (Laarhoven et al. 1992). The moves not included in this neighborhood are shown to not be promising for immediate improvements. The authors applied a restarting mechanism to the algorithm which they called Tabu Search Algorithm with Back Jump Tracking (TSAB), where the algorithm restarts the search starting from a previously found elite solution.

The smaller neighborhoods used in this research are shown to make the algorithm faster and more efficient than other algorithms in the literature. This algorithm was reported to solve medium and large JSSP instances to optimality, or close to it, in very short running times.

(Nowicki and Smutnicki 2005) developed a TS approach, i-TSAB, for minimizing makespan of the classical JSSP. This algorithm is an improvement to their previous algorithm, TSAB. New neighborhood structures were used which were found to greatly reduce the time needed to find the best solution in each neighborhood. They noted that common benchmark instances indicated the presence of the Big Valley Phenomenon, which suggests that local optima of a given solution space are clustered around the global optimum. The algorithm uses elements of path relinking to generate new initial solutions from previously known elite solutions and tries to take the search to previously unvisited zones of the Big Valley. Local search starting from those solutions is done using TSAB. This algorithm showed very good accuracy and short running times when compared to former algorithms presented.

(Brandimarte 1993), (Dauzère-pérès and Paulli 1997), (Mastrolilli and Gambardella 2000), (C. Scrich, V. Armentano 2004), (Saidi-Mehrabad and Fattahi 2007), (Abdelmaguid 2015) and (Shen, Dauzère-Pérès, and Neufeld 2018) developed Tabu Search algorithms to solve the FJSSP in regards to makespan minimization, except for (C. Scrich, V. Armentano 2004) who proposed to minimize tardiness. (Saidi-Mehrabad and Fattahi 2007) and (Shen et al. 2018) considered the sequence dependent setup times case of the FJSSP.

(Brandimarte 1993) developed a hierarchical TS algorithm for both minimizing makespan and weighted tardiness in the FJSSP using simple neighborhood structures. Different neighborhood structures were tested except for the makespan case, in which only one was used. Long term memory was used in the routing (assignment) problem and short term memory was used to navigate the scheduling problem for a given route. The author noted that a medium term memory structure could be used to specify neighborhood structures. Though results were inconclusive regarding the performance of Tabu Search, this paper identified key mechanics to focus on in future research, such as the importance of initialization strategies, the coordination between the routing and scheduling problems and the use of diversification to avoid long chains of useless moves. The author also remarked that the flexibility of the Tabu Search framework seemed promising for future applications to the multi-objective problem.

(Dauzère-pérès and Paulli 1997) proposed a TS algorithm to solve the FJSSP to minimize the makespan. They present an extension of the disjunctive graph model to contemplate the flexible case of JSSP, along with a neighborhood structure that is proven to be optimum connected, that is, there is a finite number of moves starting from a solution s which lead to the optimal solution. Though the Tabu Search algorithm used in this approach was very simple, it was enough to show that the neighborhood structure suggested can be used as basis for a good heuristic for the FJSSP.

(Mastrolilli and Gambardella 2000) proposed a TS procedure to solve the FJSSP with respect to the makespan minimization. The main contribution of this paper was the introduction of two neighborhood functions, which reduce the size of the neighborhood, thus making the search more efficient. It was proved that both neighborhood functions always contain the neighbors with the lowest makespan, and that one of the neighborhoods developed is optimum connected. Optimum connectivity is verified when, starting from a given solution, there is a finite number of moves leading to the optimal solution. However, it was concluded that the optimum connectivity property does not guarantee better results. Interestingly, instead of a simple tabu list, the authors adopted a tabu matrix TM with size $n \times m$, where n is the number of operations and m the number of machines in the problem. Tabu elements are a pair (i, k), where i is the operation being moved and k the machine assigned to operation i before the move. The value of a matrix entry $TM_{(i,k)}$when action (i, k) is performed is set to $iter + T$, i.e. the iteration when the move occurred plus the tabu length T of that action. An action is then considered tabu if:

$$iter \leq TM_{i,k}$$

Computational results showed that the neighborhood structures defined contribute to efficient heuristics, finding new best (at the time) solutions and even some optimal solutions for many different benchmark problems.

(C. Scrich, V. Armentano 2004) proposed two TS approaches for tardiness minimization in the FJSSP, namely a hierarchical and a multi-start procedure. The neighborhood considered all solutions obtained by inverting an arc in the critical path of a job, in the directed graph of a solution. Before reassigning a job, the best possible insertion in that machine's job sequence was selected. The hierarchical approach showed better results for instances with low tardiness, while the multi-start procedure showed better results for instances with higher tardiness values. The multi-start procedure was shown to be less computationally intensive.

In (Saidi-Mehrabad and Fattahi 2007) the authors propose a two-phase concurrent TS approach for minimizing the makespan in the FJSSP considering sequence-dependent setup times (FJSSP-SDST). The algorithm first finds a feasible operation sequence and then assigns the operations to the best machines. Each iteration improves the sequence, and for each sequence the best machine assignments are found. In the first stage, the neighborhood used is generated by considering adjacent pairwise interchange, i.e. swapping of two adjacent operations in the critical path. In the second stage, the neighborhood used is generated considering machine alternatives for each operation. This algorithm showed to be very fast in finding good solutions, and it's concluded that it can find the optimal solution to small problem instances in a short time while having the advantage of being easy to implement. However, its performance was poorer when dealing with large problems.

(Abdelmaguid 2015) further developed the neighborhood search functions presented by (Mastrolilli and Gambardella 2000) for the FJSSP makespan minimization, presenting a greedy randomized neighborhood search function paired with a TS approach. The randomized greedy neighborhood search (RGNS) function chooses moves sequentially, first identifying an operation to be moved, then the machine to which it is moved and finally the position in which it is processed by that machine. This function is then integrated in a Tabu Search method. The tabu list is used in the RGNS which is run for a given number of iterations in each iteration of the algorithm. When a better solution is found, the tabu list is cleared. This algorithm showed good results for instances in which the average setup-to-processing time ratio is small.

In (Shen et al. 2018) a TS algorithm for minimizing makespan in the FJSSP-SDST is applied with results outperforming other algorithms in the literature, namely (Abdelmaguid 2015) and (Saidi-Mehrabad and Fattahi 2007). This algorithm uses neighborhood functions with different propositions for feasibility check, four different move types and preliminary neighbor evaluation. Since there are four move types, this algorithm has four different tabu lists. The tabu tenure is determined using reactive Tabu Search, which is described in (Battiti and Tecchiolli 1994). They used two different restarting mechanisms at different levels, one which slightly changes a current non-improving solution and another higher-level one that restarts the search with a randomly generated solution.

Computational experiments on known benchmark instances show that the algorithm brings improvements over other algorithms in the literature.

Simulated Annealing

The Simulated Annealing (SA) is probably the second most known local search meta-heuristic after the Tabu Search and was first introduced by (Kirkpatrick, Gelatt, and Vecchi 1983). The goal is to simulate the cooling (annealing) of a solid to its ground state by means of temperature reductions throughout iterations, T_i, until it reaches a predetermined minumium T_{min}, at which the algorithm returns the best solution found. Accepting only moves which lower the objective function's value will likely result in only locally optimal solutions. To avoid this, when the simulated annealing finds a move which increases the objective function's value, it uses a probability $P = exp((f(x^*) - f(x_i))/T_i))$ based on the difference in objective function value and the current temperature, to either accept or reject this move. It should be noted that the acceptance probability is lower the higher the difference in objective function value is as well as the lower the temperature is.

The algorithm can be described as follows: consider an objective function $f(x)$ to be minimized over a set X of solutions. Let the neighborhood $N(x)$ of a solution x be the set of solutions that can be achieved from x by performing a move. Starting with a given solution x:

- Compare $f(x)$ with $f(x')$ where f' is the best neighbor solution.
- If $f(x') < f(x)$, then x' is chosen as the starting solution in the next iteration.
- Otherwise, it is accepted with the probability defined earlier P.
- If rejected, the temperature is decreased, and the current solution is used as starting solution for the next iteration.

Since the search is affected by the temperature, developing an efficient annealing schedule is important. At higher temperatures the algorithm will identify grosser characteristics, while it will find more solutions, both better and worse, when cooler. The search is performed multiple times each iteration until it reaches a stable state. Then the temperature is lowered and the search is done again (Černý 1985). The annealing schedule is defined by the following parameters (Laarhoven et al. 1992):

- Initial temperature;
- Final temperature;
- Decrement rule of the temperature;

(Matsuo et al. 1988), (Laarhoven et al. 1992), (Aarts et al. 1994) and (Naderi, Ghomi, and Aminnayeri 2010) developed simulated annealing approaches to the JSSP makespan minimization.

In (Matsuo et al. 1988) a controlled search SA approach to the general JSSP is showed, using problem specific neighborhood structures. This study suggests that using tailored heuristics to solve the JSSP does not add to the effectiveness of the algorithm, but rather the efficiency.

(Laarhoven et al. 1992) presented a SA algorithm to minimize the makespan in the JSSP and compared its performance to that of previous algorithms developed, namely Matsuo et al.'s simulated annealing method, the shifting bottleneck procedure proposed in (Adams et al. 1988) and a time-equivalent iterative improvement algorithm, using the same simple neighborhood structure they proposed. The neighborhood considered was the set of solutions reached from x by applying a move simply consisting of reverting an arc in the critical path of a given machine.

The algorithm uses a slow annealing schedule, which increases the time complexity of the algorithm but increases the probability of finding a better solution. Indeed, it is generally verified to be of polynomial time. The computational results show that the simulated annealing method presented produces good quality results. The use of a slow cooling schedule allows great results to be achieved, however at the cost of very long running times. The authors considered that the long running times are compensated by the simplicity, ease of implementation and the quality of solutions found.

In (Aarts et al. 1994) the authors compare the SA algorithm earlier developed by (Laarhoven et al. 1992) with other algorithms such as multi-start iterative improvement, threshold accepting and Tabu Search. This comparison showed that SA can find better solutions than other heuristics for a long enough running time. However, it also showed that TS algorithms outperformed the SA proposed in both solution quality and running time.

Since SA's performance greatly depends on the chosen operators and parameters, (Naderi et al. 2010) explored some of the operators and parameters using the Taguchi method. This research applied SA to the JSSP-SDST with respect to makespan minimization.

To avoid getting trapped in local minima, this algorithm employed a neighborhood structure which considers not only close neighbors, such as the ones obtained from changing the position of a single operation. Instead, it considers farther neighbors which are obtained by relocating two randomly chosen operations to two randomly chosen positions. Three different cooling schedules were tested: linear cooling, exponential cooling and hyperbolic cooling. The exponential cooling was shown to be the most appealing and thus was the one used for the final computational experiments. Comparison with other algorithms in the literature, in this case two Genetic Algorithms and a Variable Neighborhood Search, showed that the proposed algorithm outperformed them. However, no comparison was done with a Tabu Search algorithm.

(Raaymakers 2000) and (Dauzere 2002) proposed SA approaches to the FJSSP makespan minimization. The case studied in (Raaymakers 2000) is a more complex one than the standard FJSSP as operations of a given job can be processed simultaneously and there are no-wait restrictions between certain operation.

(Raaymakers 2000) proposes a SA approach to a multi-purpose batch plant with no-wait restrictions. In this type of plant, there is a variety of different products to be produced by a set of multi-purpose machines. Some jobs might have no-wait restrictions between two operations due to the intermediate product being unstable. Also, operations of a given job may be processed on more than one machine at a time.

The initial solution is generated through dispatching rules. The neighborhood considered is a 'jump' type neighborhood, which has been shown to perform better in job shop problems in (Wennink 1995) and in the no-wait job shop later in (Macchiaroli et al. 1996). These neighborhoods are generated by applying 'jump' type moves, which consist of removing an operation from the schedule and reinserting it in a different position. Operations are removed from the critical path. Due to the no-wait restrictions, operations cannot be individually removed and, as such, removal and reinsertion are applied to whole jobs. Both operation sequence and machine allocations can be changed at the same time. This algorithm was compared to a Tabu Search approach used on the same problem type, and the results are shown to be similar, though the Tabu Search had lower running times.

(Dauzere 2002) applied a concurrent SA algorithm to the FJSSP makespan minimization. This algorithm used the same neighborhood structure proposed by (Mastrolilli and Gambardella 2000). To try to explore the most possible neighborhoods, the temperature is updated after a certain number of iterations which depends on the performance of a given temperature. This is measured by the number of non-improving iterations. For very low temperatures, the probability function is updated to encourage the selection of previously visited and non-improving solutions. The initial solution is generated using a problem-specific heuristic, in this case, a heuristic based in the Shortest Processing Time rule. Termination criteria should be determined experimentally for each case and can either be a set number of maximum iterations or several consecutive non-improving iterations. The algorithm was compared to that of (Brandimarte 1993) and (Dauzère-pérès and Paulli 1997). Results showed that this simulated annealing algorithm outperformed the hierarchical Tabu Search of (Brandimarte 1993), suggesting that a concurrent approach is more effective than a hierarchical one. However, this algorithm was not as effective as, though more efficient than, the Tabu Search of (Dauzère-pérès and Paulli 1997), further verifying the hypothesis that Tabu Search provides better results than the simulated annealing for the FJSSP.

3.3.2.2. Mono-Objective Population-Based Meta-Heuristics

Among population-based meta-heuristics, the Genetic Algorithm (GA) is the most used for scheduling problems due to its simple structure and mechanisms. It's also one of the most widely studied population-based algorithms, so it has been sufficiently explored to be applied to most problem types while returning good results. It's a nature-based algorithm which tries to take the ideas from evolutionary genetics and apply them to optimization. In this type of algorithm, solutions are seen as 'chromosomes' and their attributes, or elements, are appropriately named 'genes'. An initial population is first generated, e.g. randomly or through a constructive heuristic. In each iteration, the fitness of each individual (chromosome) is measured, usually through the value of the objective function. To generate a new population each iteration, there are a few basic operators: reproduction, crossover, and mutation. (Holland 1984)

For a given population, the fittest individuals are selected for reproduction which is made through the crossover operator, which randomly generates crossover points, i.e. the position which divides the two parents' genes in the offspring (the resulting child has the first parent's genes to the left of the crossover point and the second parent's genes to the right). After the crossover has been done, the child's chromosome then suffers a mutation with a given probability, which consists of changing a randomly chosen gene. Mutation is a diversification mechanism to avoid the population steering towards local minima.

(Mesghouni 1997), (Chen, Ihlow, and Lehmann 1999), (Pezzella, Morganti, and Ciaschetti 2008) and (Azzouz, Ennigrou, and Said 2016) all developed a genetic algorithm approach to the FJSSP makespan minimization. (Azzouz et al. 2012) considered the sequence dependent setup times case of the FJSSP. These articles study different encoding schemes as well as genetic operators.

The GA was first applied to the FJSSP makespan minimization in (Mesghouni 1997). They proposed an encoding scheme to represent solutions and fitting genetic operators. They represent a chromosome as a matrix with N rows and t columns, where each row is the ordered series of operations of a given job. Each entry has two terms (M_{ki}, t_{kn}) where M_{ki} is the machine which processes the operation T_i of a given job and t_{ki} is the starting time of the operation. For this representation scheme, two crossover operators are defined: a row and a column crossover. In the row crossover, two parents and a job are selected. Two children are generated, whose machine assignments of the chosen job are the same machine assignments of one parent. The remaining jobs take the machine assignments of the other parent. The column crossover follows the same logic but choosing two parents and the same operation on each job. Machine assignments of the selected operations are taken from one parent and the remaining operations of the same job are taken from the other parent. Mutation is done by randomly selecting a chromosome, a job and an operation for that chromosome, and assigning it a randomly generated machine different from the current one. The initial population was also randomly generated. Computational results showed that the Genetic Algorithm is suitable for application to the FJSSP.

(Chen et al. 1999) also applied the GA to the FJSSP with respect to makespan minimization. In their algorithm, the representation of an individual consisted of two chromosomes A and B, where A is concerned with the routing of operations and B with operation sequencing on machines. The A chromosome's strings are of the format (T_{ij}, M_{ij}), where i denotes the job and j denotes the operation. The B chromosome is a matrix with M rows (corresponding to the M machines available) where each row consists of a vector containing the operations T_{ij} assigned to each machine. All individuals were selected for reproduction at each iteration; a uniform crossover operator was applied to the A chromosomes, where randomly selected positions of the string are swapped. Mutation on A chromosomes consisted of randomly selecting an operation and reassigning it a machine. Mutation on the B chromosomes consisted of randomly selecting a machine and two operations to switch positions. A replacement mechanism was implemented to select which individuals integrate the new generation where the parents were replaced in case the children were fitter. Computational results showed the algorithm provides high quality solutions, however at the cost of a considerably large running time.

(Pezzella et al. 2008) presents a GA to solve the makespan minimization of the FJSSP. Solutions are represented as a list of triplets (i, j, k) where i is the job, j the j^{th} an operation of job i and k the machine in which it is processed. The sequencing of operations is determined by the order in the which they appear in the list. Three methods of selection for reproduction were evaluated: binary tournament where two individuals are randomly selected from the population and the best one is inserted in the mating pool; n-size tournament, where the tournament features a random number n of individuals; linear ranking, where individuals are ranked by fitness and the probability of selection depends on rank such that the probability of an individual being selected scales with rank. Results showed the best method to be the binary tournament. Assignment crossover consists of swapping the machine assignment of a subset of operations between the two parents. Mutation consists of reassigning a single operation. An intelligent assignment mutation is also used where an operation is moved from the highest workload machine to the minimum workload machine. Sequencing crossover is the Precedence-Preserving Order-based crossover (POX) of (Lee and Yamakawa 1998). This is an operator designed to guarantee feasibility after crossover. In similar fashion, the mutation used is the Precedence-Preserving Shift (PPS). The PPS was only applied in case of improving the fitness. Computational results show that this algorithm outperforms other genetic algorithms and has comparable results to those of the best algorithm at the time, the TS of (Mastrolilli and Gambardella 2000).

More recently, (Azzouz et al. 2016) also used the GA to solve the FJSSP-SDST minimizing the makespan. The encoding used consisted of a binary matrix where rows are all the operations of jobs and columns are machines. Thus, an entry X_{ijk} of the matrix is 1 if operation T_{ij} is assigned to machine k and 0 otherwise.

Since no preemption is allowed, it comes that $\sum_{k=1}^{m} X_{ijk} = 1$. The order in which operations are featured in the matrix is the order in which they are processed and is derived from their start time. Conflicts with operations started at the same time are solved by first starting the operation which is featured in the least number of jobs. With due respect to the importance of the first generation of solutions, this algorithm uses a combination of randomly generated and heuristically constructed solutions. Each iteration, four individuals are randomly chosen from the population and the best are reproduced.

The crossover operator is the one described in (Davis, 1985) and consists of randomly selecting two positions and copying the genes between those two positions from the first parent and the rest, starting the from the farthest position, from the second parent (skipping genes already featured in the offspring). Mutation consists of selecting an operation from the highest workload machine and transferring it to the least workload machine available. Both crossover and mutation are performed according to previously defined probabilities $P_{crossover}$ and $P_{mutation}$. Computational experiments showed that this algorithm outperformed other algorithms in the literature.

3.3.2.3. Mono-Objective Hybrid Algorithms

Hybrid meta-heuristics take ideas from both local search and population-based algorithms to combine local search's efficient and effective intensification capabilities with population-based algorithms' ability to do diversify the search. Since the Tabu Search seemed to be the most appropriate local search meta-heuristic for our problem, as well as the Genetic Algorithm did for population-based meta-heuristics, the focus will be towards hybridizations using these meta-heuristics, though others will be investigated as well.

(C. Y. Zhang et al. 2008) developed a hybrid SA and TS algorithm to solve the JSSP makespan minimization. In this research, SA is used to find promising elite solutions in the Big Valley, and TS is used to intensify the search starting from those solutions. This hybridization allows a good combination of intensification from TS and diversification from SA. The tabu list used in this algorithm does not only store the operation and machine, but also the position of that operation in the machine. The length of the tabu list is dynamic and takes a random value between pre-specified limits. The SA is responsible for updating the list of elite solutions with each iteration and, should the algorithm not find a new best solution after a predetermined number of iterations, the algorithm is restarted using one of the elite solutions as a starting solution. Computational results showed that this simple algorithm can obtain good results, having determined new best solutions for previously unsolved problem instances in numerous benchmarks.

(G. Zhang, Shi, and Gao 2008), (Azzouz et al. 2012) and (Li and Gao 2016) all developed hybrid Genetic Algorithm and Tabu Search algorithms to solve the FJSSP with respect to makespan minimization.

(G. Zhang et al. 2008) employed a GA using TS for the local search procedures to solve the FJSSP makespan minimization. Again, a good choice of representation is key to improve the efficiency of the algorithm by avoiding generating infeasible solutions. The chromosome representation in this algorithm consisted of two components: Machine Selection and Operation Sequence. The first is a list of integers with length L equal to the sum of all operations (operations) of all jobs. The integer is the index for the alternative machines set for that operation. The second component is also a list of length L, where each element is the index of the job and the processing sequence comes from the order in which the index appears in the list. Selection of individuals for reproduction follows a tournament selection where three individuals are randomly chosen, their fitness evaluated and the best moves into the mating pool. The remaining are returned to the population. For machine selection strings, the crossover operator is a two point crossover. For Operation Sequence strings, in order to not violate precedence constraints, the Precedence Preserving Order-Based crossover of (Lee and Yamakawa 1998) is applied. Mutation on machine selection consists of simply changing a randomly chosen machine by a machine in the alternative machines set. Mutation on operation sequence randomly chooses two positions to swap. The Tabu Search's neighborhood is the one proposed by (Mastrolilli and Gambardella 2000). The tabu tenure is also the one used by the same work of (Mastrolilli and Gambardella 2000). Integration of the two meta-heuristics works as follows: first, an initial population is randomly generated. After evaluation, two outcomes may follow depending on a probability applied to each individual: either the individual performs a crossover or Tabu Search is applied to it. The method stops after a given number of iterations or consecutive non-improving iterations. This algorithm proved to be effective in solving the FJSSP with respect to makespan minimization.

Later, (Azzouz et al. 2012) also used GA and TS in a multi-agent system to solve the FJSSP with respect to makespan minimization. A multi-agent system is a system based on a population of agents, designed to cooperate towards common goals as well as work toward individual goals. In this paper, there are multiple Resource agents to perform local optimization with TS and an Interface agent responsible for global optimization using the GA.

(Li and Gao 2016) also developed a hybrid GA and TS for the FJSSP makespan minimization. The algorithm's framework is basically a generic GA where, at each iteration, every individual in the population is improved through a TS based local search. Encoding uses two strings, one for machine assignments and one for operation sequencing. Crossover has an equal probability to be either the POX earlier described or the job-based crossover (JBX) for sequence strings and a two-point crossover operator is used for machine assignments. These operators are shown to maintain feasibility after reproduction. Two mutation operators were also used for the sequence strings, one which simply swaps two operations in the operation sequencing string and one which randomly selects three elements in the string and generate the neighborhood concerning those elements, then randomly selecting one of the neighbors. For machine strings, the mutation operator randomly chooses r positions in the string where r is half the length of the string and changes the machines assigned to those operations. For the TS part of the algorithm, the neighborhood used the one from (Mastrolilli and Gambardella 2000). When applying TS to a given individual, it must first be decoded to a feasible schedule which is used as TS's initial solution. After the TS, the solution obtained is encoded to the GA representation for genetic operators to be applied. This algorithm showed very promising results by obtaining new best solutions in well-known benchmark problem instances.

3.3.2.4. Multi-Objective Meta-heuristics

Finally, this section reviews the multi-objective approaches to the FJSSP. As it is a scarcer area of research, meta-heuristics are not divided by type but rather grouped by objectives being optimized.

(Vilcot and Billaut 2011) proposed a TS algorithm for the a multi-objective FJSSP considering makespan and maximum tardiness as the criteria to be optimized. The tabu list used in the algorithm was the best list discovered by (Dauzère-pérès and Paulli 1997). Two approaches are proposed for generating the set of non-dominated solutions: the first uses the $\epsilon - constraint$ approach, where the maximum tardiness is bounded, with the bound being updated with each iteration, and the makespan is minimized; the second uses a linear combination of both criteria, tested with various combinations of weight for each criterion. The authors concluded that the linear combination is a better approach than the $\epsilon - constraint$ approach.

(Li et al. 2010), (Chiang and Lin 2013) and (Jia and Hu 2014) all developed meta-heuristics to solve the FJSSP considering makespan, total workload and maximum workload as the criteria to be minimized. (Li et al. 2010) proposed a hybrid Tabu Search, (Chiang and Lin 2013) proposed a genetic algorithm and (Jia and Hu 2014) proposed a Tabu Search with path relinking.

(Li et al. 2010) proposed a hybrid TS based algorithm to solve the multi-objective FJSSP with the three criteria chosen being makespan, total workload and maximum workload. The three objectives were considered in the same objective function through a linear combination. This algorithm adopted a hierarchical approach, first using a TS algorithm to solve the assignment component of the problem, and then using a Variable Neighborhood Search (VNS) algorithm to solve the sequencing problem that follows. The starting solution is selected from a population of initial solutions generated by various rules. These individuals are also generated hierarchically, first defining machine assignments and only then sequencing the operations on machines. This study encoded solutions as two vectors, one for machine assignments and the other for operation sequencing. Computational experiments show that this algorithm is superior to other algorithms in the literature approaching the multi-objective FJSSP.

(Chiang and Lin 2013) proposed a GA for the multi-objective FJSSP, considering the minimization of the makespan, total workload and maximum workload, through use of a Pareto method. The algorithm is very simple, in the sense that it features only two parameters: the number of individuals in a population, and the maximum number of generations. The encoding used for individuals is the 3-tuple scheme, where each gene is a tuple (j, i, k) where j is the job, i is the operation and k is the machine where operation i is processed. For the initial population generation, five different routing rules and three sequencing rules were used with equal probability by means of a procedure designed to maintain the diversity of the initial population. Crossover is randomly performed, being either a simple assignment crossover or the POX earlier described. One of a possible five mutation methods is applied whenever two identical individuals are in the population. The computational results comparing this algorithm to other known algorithms on known benchmark instances show that this algorithm has competitive performance, having been able to find at least 70% of the non-dominated solutions in all instances.

(Jia and Hu 2014) developed a TS procedure based on path relinking to solve a multi-objective FJSSP considering the minimization of the makespan, total workload and maximum workload, using the Pareto approach. In this work, an extension of the TSAB algorithm earlier mentioned is made to solve multi-objective problems. This extended TSAB is integrated with path relinking, resulting in a good exploration of the search space. A further extension of the algorithm is done using a dimension-oriented intensification search, and this is the version of the algorithm which achieved the best results, being comparable to those of (Chiang and Lin 2013), who developed an evolutionary algorithm to the multi-objective FJSSP, to be described later in this document.

3.4. Concluding Remarks

An overview of the literature regarding the FJSSP and JSSP shows that the most commonly applied meta-heuristics to this problem are Tabu Search, the Simulated Annealing and the Genetic Algorithm. There is also some research where hybrid algorithms are used, mainly taking elements from GA and TS, which seem to be best algorithms to use for this problem in both Mono-Objective and Multi-Objective applications.

The TS showed the most promising results coping with this type of problems. The SA seems to be the weaker algorithm, being efficient but not sufficiently effective for this type of problem. The GA has also been widely applied to the FJSSP, also showing promising results.

The literature is somewhat lacking for the multi-objective FJSSP, with the most widely adopted approaches being the Pareto method, and the use of a linear combination of both criteria in a single objective function which attributes weights to each of the objective function values. Since one of the objective functions in the present case study has a clear minimum which is very likely to be reached (total tardiness), the Pareto method does not appear to be promising. Weight attribution might also be deficient even with a good set of weights as it is possible that it would not guarantee the optimality of the total tardiness.

Furthermore, the lexicographical method has not been applied in the relevant literature, and it seems to be a good fit for the objective functions being analyzed in this dissertation. In the lexicographical approach, objective functions are optimized one at a time. Because the Total Tardiness has a clear minimum (0) and is the most important criterion for FIMA, it is the first objective to be optimized, followed by the Makespan.

With these considerations in mind, the algorithm to be developed in this dissertation is a Multi-Objective Tabu Search Algorithm, considering the Total Tardiness and the Makespan as objective functions. A lexicographical approach is used, meaning the algorithm will first optimize one of the objective functions and only then the other.

While it would be interesting to develop a Mono-Objective algorithm, there is a limitation:

- On the one hand, optimizing just the makespan would not guarantee the optimality of the total tardiness.
- On the other hand, optimizing just the Total Tardiness could result in very large Makespans.

However, since a lexicographical method is being used, the Multi-Objective problem being solved is effectively an extension of the Mono-Objective problem where a second objective function is optimized after reaching the Total Tardiness's optimum.

4. Algorithm Characterization

This chapter will focus on describing the algorithm and its mechanisms. First, a summary of relevant information gathered in the previous chapters is done, featuring a description of the problem and the algorithm's framework, as well as the criteria chosen to be optimized. Then, the first section focuses on a detailed description of the algorithm and its components. Finally, the second section presents a detailed description of each of the production strategies being analyzed (MTO and MTS), discussing advantages and disadvantages.

Before the algorithms are described, an overview of the model's parameters and solution structure is given.

- $R = \{r_1, \dots, r_n, \dots, r_m\}$ is the set of orders, where m is the number of orders. Orders are composed of:
    - an order ID r_n, which acts as a unique order identifier
    - a product reference (operation will be the term used from this point forward) o_i;
    - a quantity qt_n ;
    - a due date d_n;
- $M = \{m_1, \dots, m_k, \dots, m_{NM}\}$ is the set of machines, where NM is the number of machines
- $O = \{o_1, \dots, o_i, \dots, o_{NO}\}$. is the set of operations, where NO is the number of operations.
- The processing time of operation o_i in machine m_k is p_{ik}.
- The setup time of operation incurred by scheduling operation o_i after operation o_j is $s_{ij} = s_{ji}$. The first operation o_i scheduled on each machine, i.e. the starting point of each machine, is considered to have setup time $s_{i0} = 0$.

The solution representation chosen is inspired on the Disjunctive Graph Representation described in chapter 2.

In this adapted representation, solutions are represented as a dictionary. A dictionary is a set of keys, each holding any type of data structure. Data can be accessed by key. In this case, the keys of the dictionary are the machine IDs, and each key holds a list of the orders processed on a specific machine, in the respective manufacturing sequence.

Using an instance with 8 orders and 4 machines as an example, a solution S could be:

$S = \{1{:}\,(1,3,5), 2{:}\,(2,4,6), 3{:}\,(7), 4{:}\,(8)\}$, where orders 1,3 and 5 are produced on machine m_1; orders 2,4 and 6 on machine m_2; order 7 on machine m_3 and order 8 on machine m_4. Orders are processed by the order in which they appear on the list.

To refer that two operations v and w are processed consecutively on the same machine, one should use (u, v). A given machine k's schedule is given by $S[k]$.

As mentioned in previous chapters, this dissertation will focus on two criteria: Total Tardiness and Makespan. The Total Tardiness is the key criterion, since timely delivery of FIMA orders is the first and foremost concern of LSI. This is the reason why the Total Tardiness is the first objective function to be minimized in the lexicographical approach. Then, the Makespan should be tended to. A lower Makespan has the obvious benefits of cost reduction through the reduction of production time and machine utilization.

Considering 0 as the starting time, let C_n be the completion time of order R_n.

The tardiness T_n of an order R_n is given by:

$$(1) \begin{cases} T_n = d_n - C_n, \ d_n \leq C_n \\ \quad 0, \ otherwise \end{cases}$$

The total tardiness T_{total} is the sum of all orders' tardiness:

$$(2) \ T_{total} = \sum_{n=1}^{m} T_n$$

The makespan C_{max} is given by:

$$(3) \ C_{max} = \max C_n$$

The first section characterizes the Bi-Objective Tabu Search algorithm, from now on referred to as MOTSA, focusing on each of the components of the algorithm. The MOTSA was implemented in Python 3.8 running on a Windows 10 Operating System.

This algorithm can be applied to both MTS and MTO strategies, with the difference being a set of calculations at the start of the algorithm to determine the production orders required to replenish the inventory as needed. The difference between the MTS and MTO strategies as well as a detailed description of the way each of them works will be discussed in a later chapter.

4.1. Multi-Objective Tabu Search Algorithm (MOTSA)

The problem is to schedule orders on machines in order to minimize tardiness and makespan. Since a lexicographical method is being used and it is a scheduling problem, total tardiness is the first objective function to be minimized. Also, since objective functions are being separately optimized, neighbor functions are different for each of them.

First, each of the algorithm's components is detailed, then the general framework is described along with pseudo-code and a diagram for comprehension.

The Multi-Objective Tabu Search algorithm has the following components:

1. Problem initialization
2. Initial solution generation
3. Tardiness Neighbor Function
4. Makespan Neighbor Function
5. Tabu List
6. Diversification strategies

For the description of each of the components, example data is provided where needed. The data used for the description of the algorithm's procedures is representative in the sense that it allows the reader to understand the algorithm's applicability.

4.1.1. Problem Initialization

First and foremost, the problem data must be initialized and loaded into adequate data structures. Problem data consists of:

1. Number of different products
2. Product setup times
3. Number of machines
4. Processing times of products in each machine
5. Number of orders

To extract all of the information required, a data set consisting of processing times, setup times and orders is needed:

1. **Processing times**: this file contains the information regarding processing times of all products in each machine, thus also indicating which machines are eligible to manufacture each product.

Table 2 - Example processing times (minutes)

		Products				
		1	**2**	**3**	**4**	**5**
Machines	**1**	1	2	1.5	3	2
	2	3	1	3	2	1
	3	1	2	4	2.5	3
	4	2	1.5	1	1	1
	5	2.5	3	1	1	2

Values in the table represent the time needed, in minutes, to manufacture 1 unit of the product. For example, it takes 2 minutes to manufacture 1 unit of product 2 on machine 1, while it takes only 1 minute on machine 2.

2. **Setup times** are represented as table with the setup times incurred when switching production between any two products that can be manufactured on the same set of machines.

Table 3 - Example setup times (minutes)

		Products				
		1	**2**	**3**	**4**	**5**
Product	**1**	0	10	50	50	100
	2	10	0	10	50	100
	3	50	10	0	50	50
	4	50	50	50	0	100
	5	100	100	50	100	0

From Table 3, switching production between products 1 and 2 generates a setup time of 10 minutes, for example. In this example, all products can be manufactured in any machine. The setup times are symmetrical, since changing production from product 1 to 2, or from product 2 to 1 requires the same changeovers.

3. **Orders:** all orders received from FIMA, namely products, quantities and due dates.

Table 4 - Example orders (due date minutes)

Order ID	Product	Quantity	Due
0	2	50	140
1	1	50	150
2	3	40	150
3	3	25	150
4	4	55	150
5	5	45	180
6	3	20	180
7	4	70	190
8	1	50	200
9	1	50	200
10	3	20	200
11	5	90	250
12	5	55	250
13	2	70	300
14	1	50	350

Table 4 depicts an example order file. Due Dates are in minutes starting from the beginning of the production schedule. The quantity of each order is in units.

4.1.2. Initial Solution

Local search metaheuristics excel in search intensification, however lack in diversification. To avoid getting trapped in local minima around a bad area of the solution space, it is convenient to use a good quality solution. This helps local search metaheuristics reach better solutions in fewer iterations. For this reason, local search metaheuristics often use heuristically constructed starting solutions, using a convenient rule or set of rules to do so, as opposed to randomly generated solutions. Since total tardiness is the first objective to be minimized, it is convenient to build an initial solution that is tardiness-oriented. To do so, orders are scheduled by ascending due date, that is, using an Earliest Due Date (EDD) dispatching rule. Figure 6 is a flow chart describing the MOTSA Initial Solution generation procedure. Essentially, orders are sequenced by due date. Then, for each order, the algorithm iterates over the eligible machines, starting from the fastest. In each machine, the algorithm checks all possible positions for a timely insertion, that is, an insertion which does not increase tardiness. On a first pass, if there is no timely insertion available, the order in question is skipped. On a second pass, the insertion which leads to the lowest increase in tardiness is selected. Once an order is inserted, it is removed from the list of available orders. Once all orders have been inserted, the initial solution has been generated.

This algorithm is fully deterministic, so the same initial solution will be achieved every time for a given set of orders. However, since a convenient heuristic was developed, the solution produced has sufficient quality for a good exploration of the solution space.

To verify this hypothesis, results of 10 runs, each performing 25 iterations using a randomly generated initial solution, were compared to the results of 10 runs, each performing 25 iterations using a the MOTSA's initial solution generator. Consider the example problem data presented above, i.e. Tables 2, 3 and 4 for a simple illustration of the algorithm. This example will be used throughout this chapter to illustrate the procedures described. The initial solution generated has a Makespan of 300 minutes and a Total Tardiness of 45 minutes.

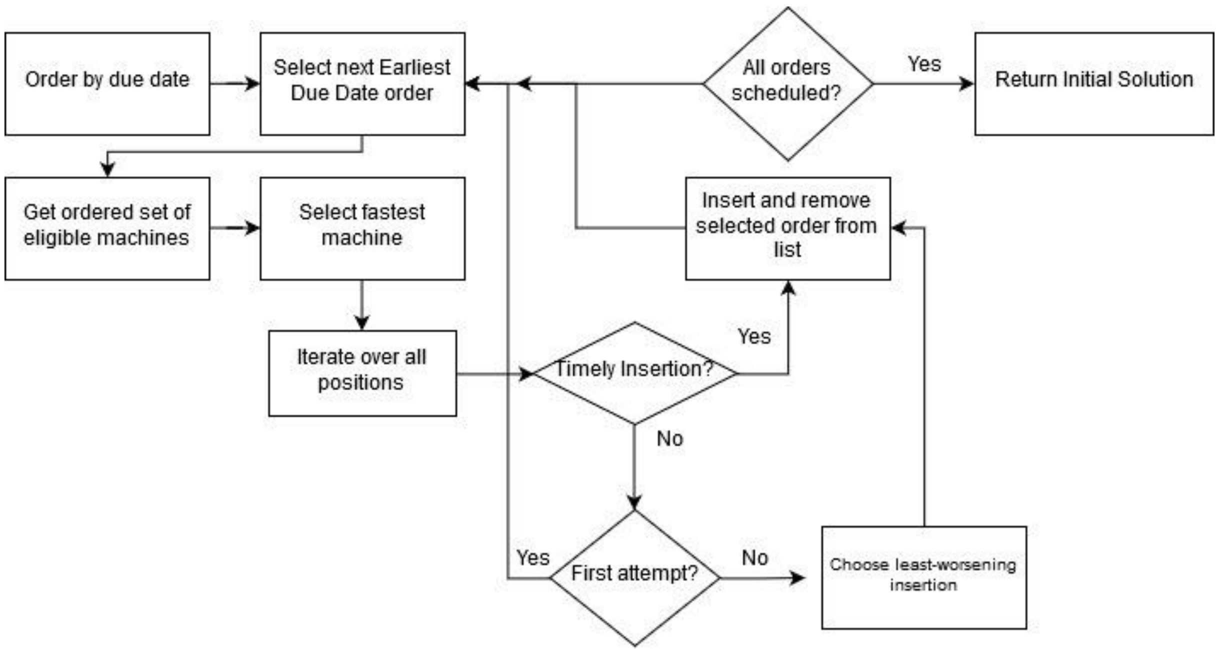

Order by due date
Select next Earliest Due Date order
All orders scheduled?
Yes
Return Initial Solution
Get ordered set of eligible machines
Select fastest machine
Insert and remove selected order from list
Yes
Iterate over all positions
Timely Insertion?
No
Yes
First attempt?
No
Choose least-worsening insertion

Figure 6 – Flow Diagram of the Initial Solution generation algorithm

Figure 7 shows the results of this comparison. The MOTSA Initial Solution clearly improves the search's quality, since it always reaches optimal tardiness and lower makespan than the randomly generated initial solutions.

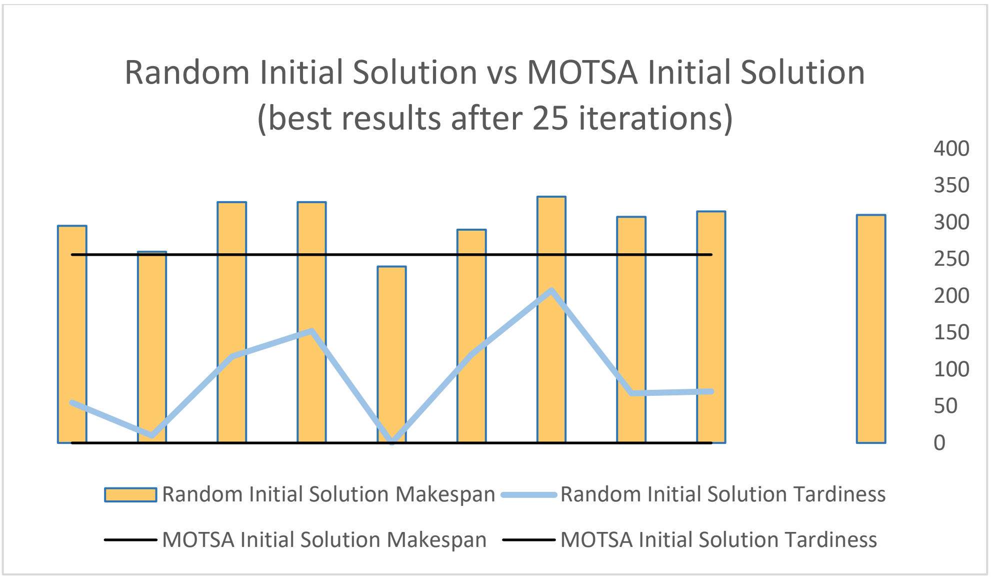


Figure 7 – Comparison of the quality of solutions using a randomly generated initial solution vs the MOTSA Initial Solution

4.1.3. Neighbor Functions

A neighboring solution is any solution S' that can be reached from a seed solution S by applying a move. The neighborhood structure is determined by the move types, so it is convenient to select move types that are sufficiently broad for a good exploration of the solution space, but strict enough such that scanning the neighborhood is not too long of a task. When applying a move, a preliminary evaluation can be done to ensure that the algorithm does not visit non-promising regions of the solution space. Preliminary evaluation is done by a simple check on equations representing each of the objective functions. The equations in question are addressed in each objective function's description.

Since all operations are single stage and preemption is not allowed, each operation is processed on a single machine. As such, the critical path is the list of operations on the machine with the highest completion time, that is, the machine which determines the makespan. Any operation on the critical path is regarded as a critical operation. This notion will be important when developing the neighbor functions.

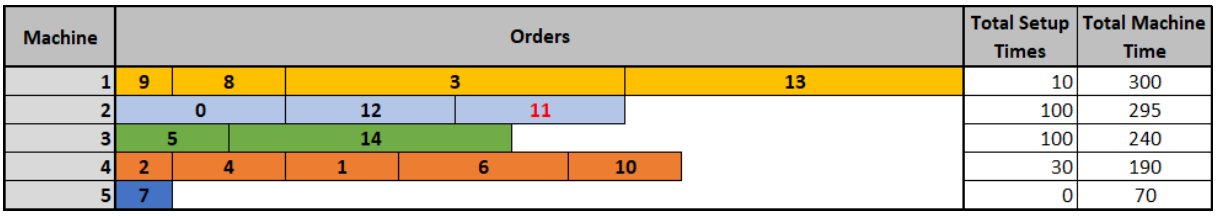
It is also useful to define an operation u's tail r_u, the time between starting the first operation on that machine and the start of operation u, and its head q_u, the time between finishing operation u and the finishing time of the last operation on that machine.

As mentioned previously, the FJSSP is composed of two sub-problems: the assignment problem and the sequencing problem. Thus, move types must entertain both sub-problems. As such, moves can be broadly described as being either:

- **sequencing moves,** which change the sequencing of orders on a machine, by swapping the position of orders on the solution list. A single order is selected and moved to a different position in the same machine k.

- **assignment moves,** which consist of selecting an order and changing the machine where it is processed. An order is deleted from its current machine k and reinserted in another machine k'.

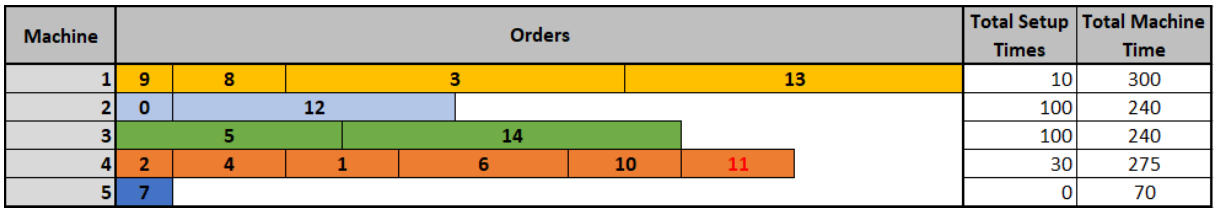
To illustrate each of the move types, consider the Gantt represented in Figure 8. In machine 1, for example, orders 1, 4 and 6 are processed in that sequence. This is the critical machine and its makespan is 300 minutes, having total setup times of 10 minutes. The width of orders is representative of the duration of orders. Tardy orders are in red.



Machine	Orders	Total Setup Times	Total Machine Time
1	9, 8, 3, 13	10	300
2	0, 12, 11	100	295
3	5, 14	100	240
4	2, 4, 1, 6, 10	30	190
5	7	0	70

Figure 8 -Gantt Chart of the Initial Solution for the Example Problem Data (time in minutes)

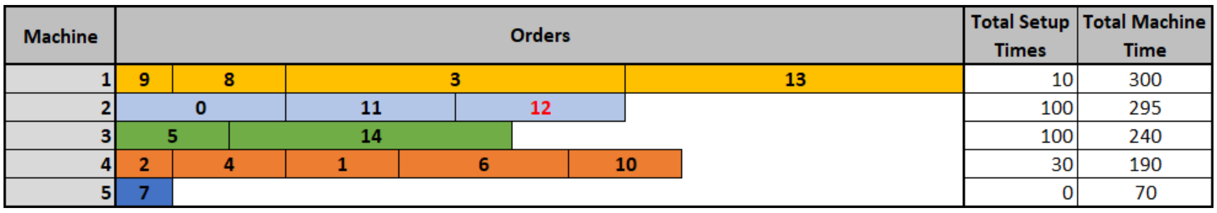
One possible move is to move order 11 to machine 4, that is, an assignment move. A Gantt Chart of this neighbor solution is shown in Figure 9.



Machine	Orders	Total Setup Times	Total Machine Time
1	9, 8, 3, 13	10	300
2	0, 12	100	240
3	5, 14	100	240
4	2, 4, 1, 6, 10, 11	30	275
5	7	0	70

Figure 9 – Gantt Chart of a neighbor generated through an assignment move (time in minutes)

Another possible move would be to move order 11 between orders 0 and 12, which is a sequencing move. The Gantt Chart of this neighbor solution is shown in Figure 10.



Machine	Orders	Total Setup Times	Total Machine Time
1	9, 8, 3, 13	10	300
2	0, 11, 12	100	295
3	5, 14	100	240
4	2, 4, 1, 6, 10	30	190
5	7	0	70

Figure 10 – Gantt Chart of a neighbor generated through a sequencing move (time in minutes)

Every iteration of the algorithm, a neighborhood is generated for the current solution. When generating a neighborhood, the neighbor functions are called several times until the neighborhood has the desired size.

Below follows a detailed description of the moves considered and the evaluation techniques for each objective function.

4.1.3.1. Total Tardiness

Since a lexicographical method is being adopted and on-time delivery of orders is a key concern for LSI, total tardiness is the first objective function to be considered. Since orders being scheduled ahead of time or exactly on time, it is pointless to continue optimizing total tardiness once this value reaches 0.

For tardiness optimization, the selected operations u to be moved are the tardy operations. For notation purposes, assume 0 and $*$ to be *dummy* operations at the start and end of any machine's schedule, which incur in not setups or processing time. Let t_u be the tardiness of operation u in the current solution.

Sequencing

Suppose that $(a, u), (u, b)$ and $(v, w,)$ are in the current solution's critical path. (v, w) is any other pair of consecutive operations on the same machine. A possible sequencing move consists of moving operation u. Let k be the machine where u is scheduled.:

- right before operation a, making $(u, a), (a, b)\ and\ (v, w)$ components of the new solution.
- right after operation b, making $(a, b), (b, u)\ and\ (v, w)$ components of the new solution.
- between v and w, making $(a, b), (v, u)$ and (u, w) components of the new solution.

Before making the move and consequently generating a solution to be later visited, it is possible to check whether said move will result in a makespan improvement with a simple verification. Since a sequencing move only affects setup times, only setup times need to be checked. Each of the previously defined move types require their own equation, verifying that the total tardiness of the new solution will be lower than the current solution S's total tardiness. For such a check, each operation z scheduled after u in both solutions must verify the following equations. Note that it is possible that the operation being moved has a higher tardiness in the new solution while the total tardiness is still improved. First, the tardiness of u in the new solution must be calculated.

$$(4)\ t'_u = d_u - (q_w + s_{uv} + p_u \times qt_u)$$

Then, checking the tardiness of every operation z that is scheduled after u in the new solution. There are four cases for checking every operation z:

1. u was scheduled before z in both the current and new solutions. In this case, only the setups must be considered.

$$(5)\ t'_z = t_z - s_{ua} - s_{bu} - s_{wv} + s_{ba} + s_{uv} + s_{wu}$$

2. u was scheduled before z in the current solution and after z in the new solution.

$$(6)\ t'_z = t_z - s_{ua} - s_{bu} + s_{ba} - p_u \times qt_u$$

3. u was scheduled after z in the current solution and before z in the new solution.

$$(7)\ t'_z = t_z - s_{wv} + s_{uv} + s_{wu} + p_u \times qt_u$$

4. u was scheduled after z in both the current and new solutions. In this case, t_z is the same in both solutions.

$$(8)\ t'_z = t_z$$

For a move to be promising or non-worsening, the sum of the tardiness of operations in the selected machine of the new solution must be lower than or equal to the same in the new solution, thus:

$$(9)\ \sum t'_z,\ z \in S[k] \quad < \quad \sum t_z,\ z \in S[k]$$

Assignment

Suppose that (a,u) and (u,b) are in the current solution's critical path and (v,w) is in any of the machines eligible to process operation u. An assignment move consists of moving operation u between v and w, making $(a,b), (v,u)$ and (u,w) components of the new solution.

As with sequencing moves, preliminary evaluation can also be done for assignment moves to avoid generating and later visiting a solution which is not promising. However, in the case of assignment moves, both machines' operations' tardiness must be calculated.

Calculating the tardiness of u in the new solution is the same as in the sequencing case:

$$(10) \qquad t'_u = d_u - (q_w + s_{uv} + p_u \times qt_u)$$

In the original machine k, only the tardiness of operations z scheduled after u will change:

$$(11) \qquad t'_z = t_z - s_{bu} - s_{ua} + s_{ba} - p_u \times qt_u$$

In the new machine k' as well, only the tardiness of operations z scheduled after u in the new solution will change:

$$(12) \qquad t'_z = t_z - s_{wv} + s_{uv} + s_{wv} + p_u \times qt_u$$

As with sequencing moves, should both of the following equations hold true, the move will result in a lower total tardiness.

$$(13) \qquad \sum t'_z,\ z \in S'[k] \quad < \quad \sum t_z,\ z \in S[k]$$

$$(14) \qquad \sum t'_z,\ z \in S'[k'] \quad < \quad \sum t_z,\ z \in S[k']$$

4.1.3.2. Makespan

When defining the neighbor function for makespan optimization, it is clear that the operation to be moved is any operation in the critical path. Since there are sequence dependent setup times, the sequencing of operations is important.

It should be noted, though, that when optimizing makespan, one must still care for the total tardiness. As such, for the makespan optimization phase, the quality of the move depends on both the makespan and the total tardiness. With this in mind, the makespan optimization neighbor function is an extension of the tardiness optimization neighbor function, where makespan must also be optimized. This makes it much harder to find promising moves, however it is necessary as tardiness must not be tolerated.

Sequencing

Suppose that $(a,u),(u,b)$ and $(v,w,)$ are in the current solution's critical path. (v,w) is any other pair of consecutive operations on the same machine. Note that if two machines have the same makespan, no immediate improvement in the makespan can be found, since a move consists of moving a single operation. A possible sequencing move consists of moving operation u:

- right before operation a, making $(u,a),(a,b)$ and (v,w) components of the new solution.
- right after operation b, making $(a,b),(b,u)$ and (v,w) components of the new solution.
- between v and w, making $(a,b),(v,u,)$ and (u,w) components of the new solution.

Before making the move and consequently generating a solution to be later visited, it is possible to check whether said move will result in a makespan improvement with a simple verification. Since a sequencing move only affects setup times, only setup times need to be checked. Each of the previously defined move types require their own equation, verifying that the new setup times do not exceed the current setup times.

(15) $$s_{au} + s_{ba} + s_{wv} \leq s_{ua} + s_{bu} + s_{wv}$$

(16) $$s_{ba} + s_{ub} + s_{wv} \leq s_{ua} + s_{bu} + s_{wv}$$

(17) $$s_{ba} + s_{uv} + s_{wu} \leq s_{ua} + s_{bu} + s_{wv}$$

Should the equation hold true, the move is promising or non-worsening. Note that such an evaluation does not contemplate tardiness. As mentioned above, the equation presented in the tardiness neighbor function section for sequencing moves, equation (9) must also be verified for this case.

Assignment

Suppose that (a, u) and (u, b) are in the current solution's critical path and (v, w) is in any of the machines eligible to process operation u. An assignment move consists of moving operation u between v and w, making $(a, b), (v, u)$ and (u, w) components of the new solution.

As with sequencing moves, preliminary evaluation can also be done for assignment moves to avoid generating and later visiting a solution which is not promising. Let $mksp_k$ and $mksp_{k'}$ be the makespans of the critical machine and the new machine, respectively. The verification equations are then:

$$(18) \qquad p_{uk'} \times qt_n + s_{uv} + s_{wu} - s_{wv} \leq mksp_k - mksp_{k'}$$

To ensure that the total production time of machine k' in the new solution is not greater than the current solution's makespan.

$$(19) \qquad p_{uk} \times qt_n + s_{ua} + s_{bu} \leq s_{ba}$$

To ensure that the total production time of machine k in the new solution is not greater than the current solution's makespan.

Furthermore, the total tardiness verification equations (13) and (14) must also hold true.

4.1.3.3. Neighborhood Generation

Having defined the neighbor functions oriented toward each of the objective functions, the neighborhood generation procedure must then be defined.

Since the neighborhood, as the set of all solutions that can be reached from the seed solution by applying the move types defined, can be very large, it is convenient to randomize the search process to avoid following a clear path in all executions of the algorithm. Note that, even with randomization, repetitiveness is also a risk if the seed used for the random number generator is fixed. In Python, the programming language used for the implementation of the algorithm, random number generators use the system time as seed by default, to ensure *true* randomization.

For either case, the candidate operations to be moved are selected at random among the list of candidate operations. For each operation, the position the candidate operation is being re-sequenced to is also random, as is the machine in case of assignment, and even the position within the randomly select machine. To avoid scanning through the entire neighborhood for promising solution, when there might not even be one, non-promising solutions can be accepted as neighbors according to a random probability, if within acceptable parameters. A higher Total Tardiness is never accepted, and the Makespan can never exceed double the seed solution's Makespan.

4.1.4. Tabu List

The tabu list is a key component of the Tabu Search. It is the memory structure responsible for restricting the search by storing information about previously visited solutions. Neighborhood solutions then might be accepted or rejected based on their tabu status.

4.1.4.1. Assignment Tabus

One of the tabu lists features **assignment tabus** in the form of pairs (r_n, m_k). Any neighbor solution which has any order r_n in the respective tabu m_k is considered a tabu solution. A pair is added to the Tabu List when order r_n is moved from machine m_k.

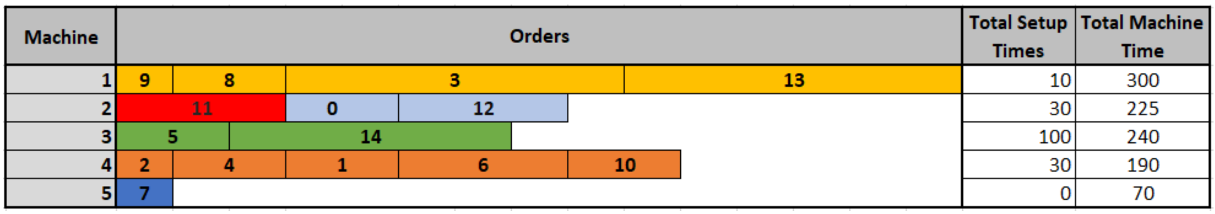
To illustrate how the assignment tabu list works, consider the initial solution and its the neighbor solutions presented in the example. The neighbor solution in Figure 9 is an assignment move, which moves order 11 from machine 2 to machine 4. If this solution is selected as the new current solution by the algorithm, then a tabu element is added to the assignment tabus.

The goal of the tabu list is to prevent the algorithm to return to previously visited solutions, so a pair (11,2) is added. If the tabu list passes the maximum tabu list size (a user-defined parameter), the oldest tabu element in the list is removed.

This means that any candidate neighbor solution which has order 11 scheduled in machine 2 can only be accepted if it is a new best solution.

To check if a solution is tabu solution due to the assignment, the solution is checked for the presence of tabu elements. As an example, consider the move performed that generates the solution in Figure 9 from the solution of Figure 8.

Given that the order being moved, order 11, was originally scheduled in machine 2, the corresponding Tabu List entry being added is the pair (11,2). Suppose, then, that a new candidate solution is the solution shown in Figure 11. This solution consists of moving order 11 back to machine 2. The tabu element (11,2) is present in the solution, making this solution tabu. Tabu elements of a solution are highlighted in red. This solution has a lower total setup time in machine 2, and has achieved optimal Tardiness. Therefore, it can be accepted as an aspiring Tabu Solution.



Machine	Orders					Total Setup Times	Total Machine Time
1	9	8	3	13		10	300
2	11	0	12			30	225
3	5	14				100	240
4	2	4	1	6	10	30	190
5	7					0	70

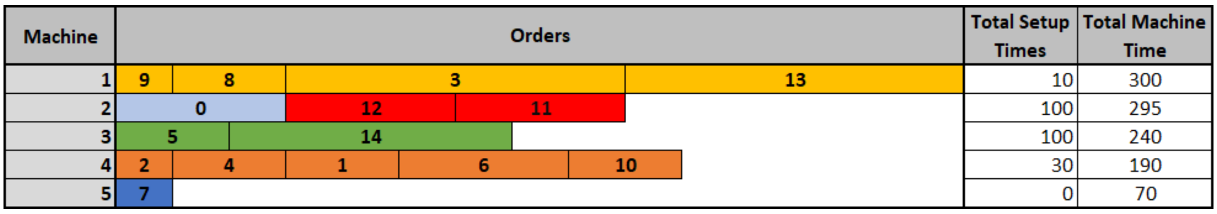
Figure 11 – Gantt Chart of a candidate solution illustrating an assignment Tabu Status.

4.1.4.2. Sequencing Tabus

The other list features **sequencing tabus**, in the form of pairs (r_n, r_m). Any neighboring solution featuring the precedence relation (r_n, r_m) in any machine can not be accepted as a new solution during an iteration. A pair is added to the tabu list when order r_n is moved from before order r_m.

To illustrate how the sequencing tabu list works, consider the initial solution and its the neighbor solution shown in Figure 10. This solution is generated by is a sequencing move, which re-sequences order 11 to before order 12. If this solution is selected as the new current solution by the algorithm, then a tabu element is added to the sequencing tabus. In the this sequencing move, the pair (12,11) is added. If the tabu list passes the maximum tabu list size (a user-defined parameter), the oldest tabu element in the list is removed.

Suppose, then, that a new candidate solution is the solution shown in Figure 12. This solution consists of moving order 11 back to its original position, after order 12. The tabu element (12,11) is present in the solution, since order 12 is scheduled before order 11 in this solution, making it tabu. Tabu elements of the solution are highlighted in red. This solution is not an improvement when compared to the seed solution in Figure 10. As such, it will not be accepted as a new solution.



Machine	Orders					Total Setup Times	Total Machine Time
1	9	8	3	13		10	300
2	0	12	11			100	295
3	5	14				100	240
4	2	4	1	6	10	30	190
5	7					0	70

Figure 12 - Gantt Chart of a candidate solution illustrating a sequencing Tabu Status.

4.1.5. Diversification Strategies

Despite the tabu list which, to a certain degree, ensures diversification of the search, it is still possible that the algorithm will get trapped in local optima. It is not possible to be absolutely certain when it does, though, so a good way to trigger this other diversification mechanism is to keep track of consecutive non-improving iterations.

As mentioned previously, it is possible that the algorithm will accept a non-improving solution. This non-improving solution, however, is not guaranteed to take the search to a new area, meaning that the following iterations are very likely to result in improvements in the current solution's quality, but not necessarily result in improvements when compared to the best solution's quality. For this reason, the algorithm keeps track of the number of consecutive iterations where the best solution is not improved upon. The counter is reset every time a new best solution is found.

When the counter reaches a threshold *MaxNonImprovIter,* a 'random' swap is done. The way this random swap works is it randomly select an operation from the critical machine, as with the neighbor function. The algorithm then randomly selects an operation from any of the eligible machines and checks whether swapping these two operations results in a non-tardy solution. If it does, the current solution is updated to the swapped solution, regardless of the makespan. When the random swap is done, the tabu lists are cleared as though the algorithm was restarted using the swapped solution as a starting solution. The running time and iterations are not reset.

While looking for an acceptable swap, the algorithm performs the same checks described in the tardiness neighbor function, i.e. the following equations must be verified:

$$(13) \qquad \sum t'_z\,,\ z \in S'[k] \ < \ \sum t_z\,,\ z \in S[k]$$

$$(14) \qquad \sum t'_z\,,\ z \in S'[k'] \ < \ \sum t_z\,,\ z \in S[k']$$

If, however, the seed solution already has optimal tardiness, the equations are instead:

$$(15) \qquad \sum t'_z\,,\ z \in S'^{[k]} \ \leq \ \sum t_z\,,\ z \in S[k]$$

$$(16) \qquad \sum t'_z\,,\ z \in S'^{[k']} \ \leq \ \sum t_z\,,\ z \in S[k']$$

To illustrate a random swap of a given solution, take the solution shown in Figure 11. Suppose the algorithm is stuck on this solution and therefore performs a random swap as described above. Since this solution has achieved optimal Tardiness, only a solution with 0 tardiness can be accepted. One example of such a solution is shown in Figure 14.

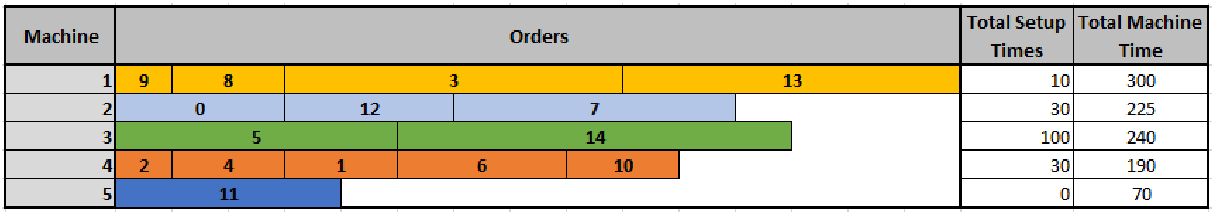


Figure 13 - Gantt Chart of a solution generated by a random swap (swapped solutions are highlighted in green.

4.1.6. General Framework

This section details the general framework for the bi-objective Tabu Search Algorithm.

The algorithm starts by generating an initial solution. Then, the main cycle begins. The first part of the cycle consists of generating the neighborhood of a solution. If the initial solution already has optimal tardiness, then no tardiness optimization is required, so the neighborhood is generated using the makespan neighbor function. Otherwise, the algorithm starts by optimizing tardiness. Once the algorithm's current solution has optimal tardiness, it moves on to makespan optimization. When stop any stop criteria is reached, the algorithm stops and returns the best solution found. Pseudo-code with greater detail is presented for both stages of the algorithm in the following sections.

Figure 14 describes the main idea of the general framework with a macroscopic view of its components.

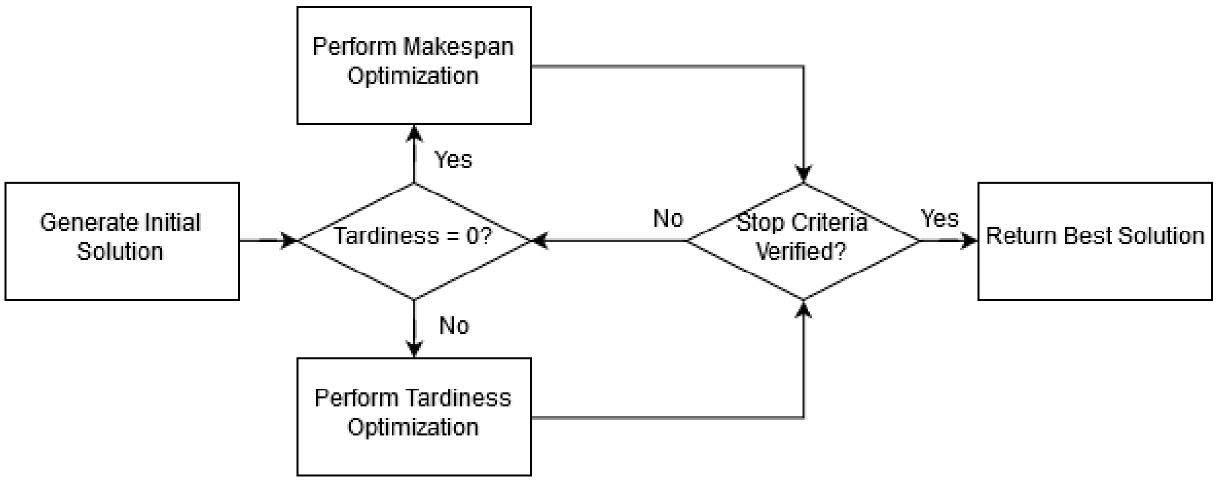

Perform Makespan Optimization
Yes
Generate Initial Solution
Tardiness = 0?
No
Stop Criteria Verified?
Yes
Return Best Solution
No
Perform Tardiness Optimization

Figure 14 – Flowchart of a macroscopic view general framework of the MOTSA

Tardiness optimization

When optimizing tardiness, the algorithm's pseudo code is presented below.

```
generate tardiness neighborhood and order by ascending tardiness.

iterate over neighbors:
   if there is a non-tabu solution which is a new best:
         update solution to this neighbor
         updated tabu list
   if the only improving solution is tabu:
         update solution to this neighbor
         update tabu list

   otherwise select next neighbor.

if the current solution is better than the best solution:
   update the best solution to the current solution
   reset the non-improving iteration counter
otherwise update the non-improving iteration counter.

if non-improving iteration counter <= maximum non-improving iterations:
   apply random swap to the current solution
   if no random swap is found with the restrictions imposed:
      update current solution to the best solution
      compare to the best solution – update if better
      every third swap:
        clear tabu lists

if stop criteria are verified:
      stop and return best solution found
```

An accompanying flowchart is shown in Figure 15 for a more visual representation. If no solution is accepted after this procedure, the current solution remains the same, thus is used as seed solution for the next iterations, unless a random swap is necessary based on the non-improving iterations counter.

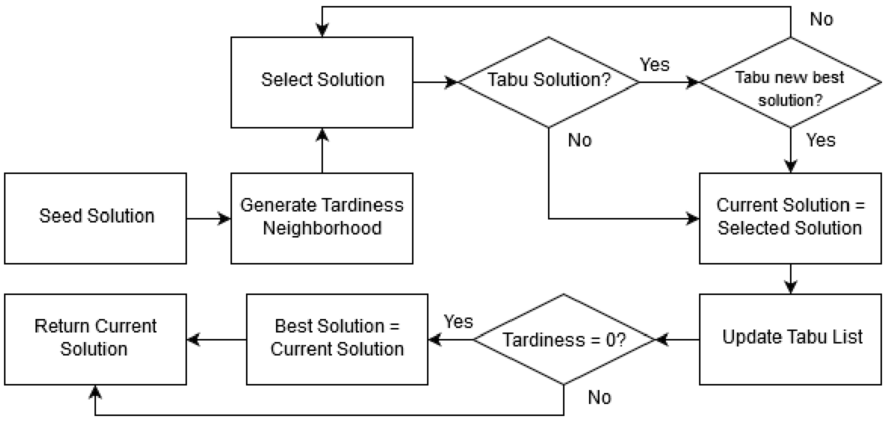


Figure 15 – Flowchart of a tardiness optimization stage's iteration

Makespan optimization

The makespan optimization procedure operates similarly to the tardiness optimization stage, except it generates a makespan oriented neighborhood and only accepts solutions which do not deteriorate the tardiness of the current solution. The pseudocode for each makespan optimization iteration is presented below.

```
generate makespan neighborhood and order by ascending makespan.

iterate over neighbors:
   if there is a non-tabu solution which is a new best:
         update solution to this neighbor
         update tabu list
   if the only improving solution is tabu:
         update solution to this neighbor
         update tabu list

   otherwise select next neighbor.

   if the current solution is better than the best solution:
         update the best solution to the current solution
         reset the non-improving iteration counter
   otherwise update the non-improving iteration counter.

if non-improving iteration counter <= maximum non-improving iterations:
      apply random swap to the current solution
      if no random swap is found with the restrictions imposed:
         update current solution to the best solution
      compare to the best solution – update if better
         every third swap:
         clear tabu lists

if stop criteria are verified:
      stop and return best solution found
```

Figure 16 shows a flowchart of the Makespan optimization stage iteration for a more visual understanding of each iteration.

Finally, refer to Figure 17 for the full flowchart of the algorithm.

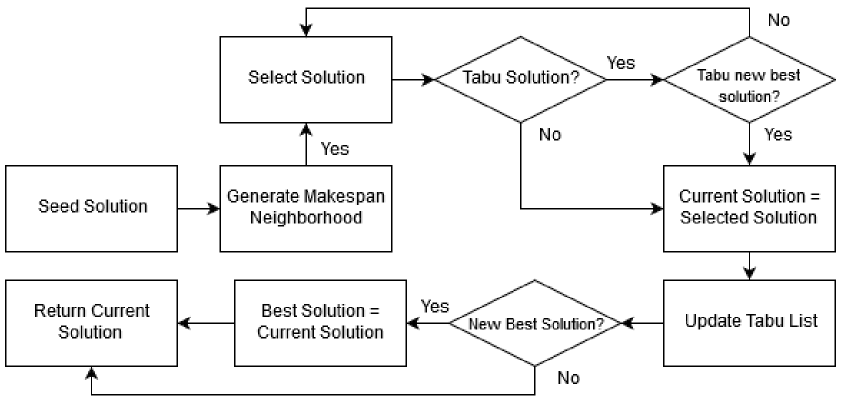


Figure 16 – Flowchart of a makespan optimization stage's iteration

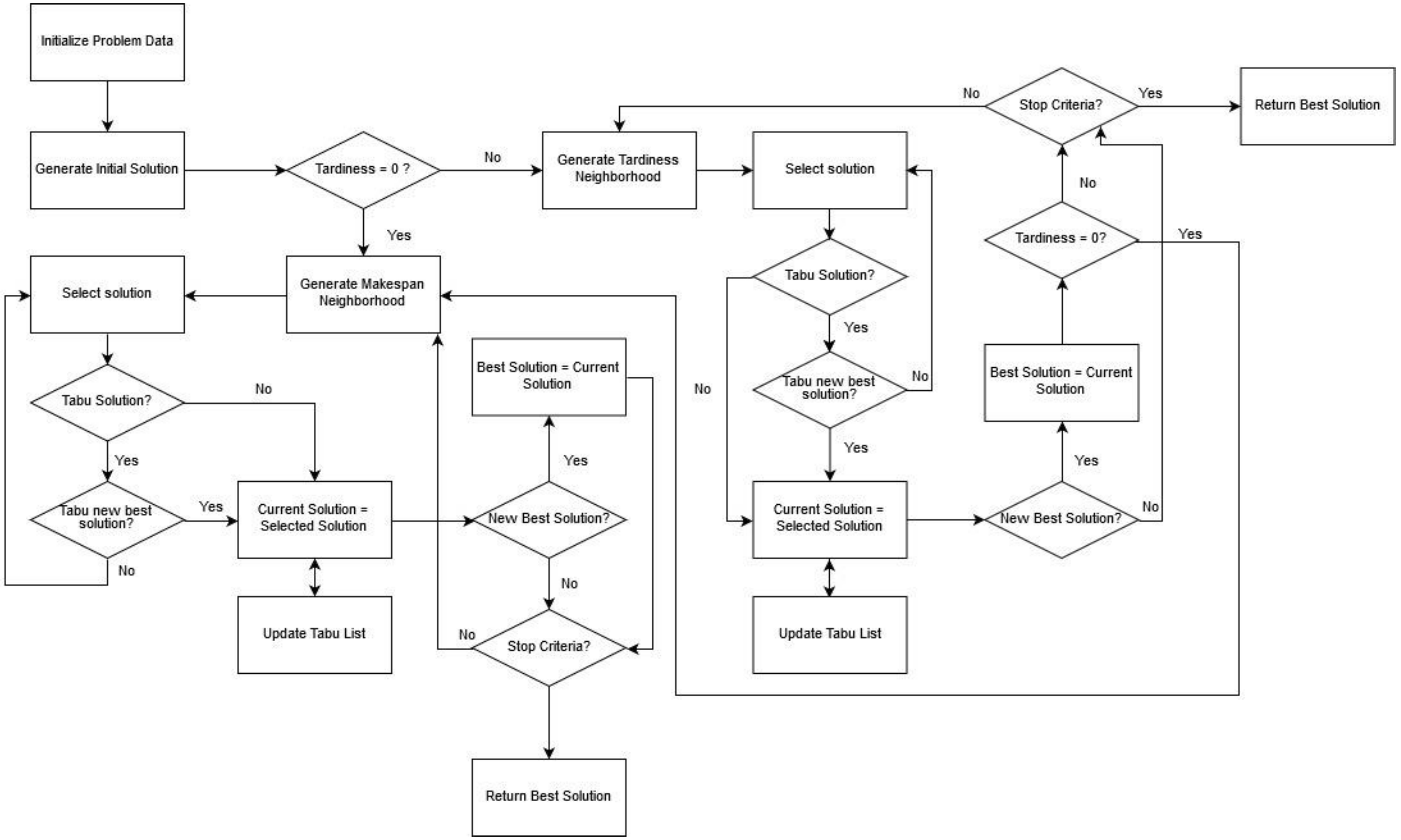


Figure 17 – Flow Diagram of the general framework of the MOTSA

4.2. Production Strategies

In the manufacturing industry, service level is very important for companies, and is assured mostly through keeping inventory. As such, the typical approach to production scheduling adopts the MTS strategy, where production orders are determined based on target stock levels. Fulfilling orders from the inventory is somewhat easier, as deadlines are easier to meet when it is not necessary to manufacture every order.

However, keeping inventory of a large amount of finished product means high inventory costs. For this reason, the MTO production strategy is highly interesting. Should the algorithm, considering the MTO strategy, be capable of dealing with reasonably diverse sets of orders and other situations such as unavailable machines, switching to an MTO approach is preferable, as inventory costs associated with infrastructure and management will likely go down.

4.2.1. Make to Stock

In the MTS approach, production orders are determined based on the inventory necessities. Orders from FIMA are satisfied from the inventory. Should the inventory level be insufficient to deliver the orders received or drop below a pre-established threshold, a production order is generated. This process's pseudocode is presented below:

```
receive set of orders and stock information (current levels, minimum and maximum).

iterate over orders:
   calculate updated stock levels based on order quantity and current stock levels:
      if updated stock level is between the minimum and maximum levels:
         no production orders are generated for the product
         update current stock levels
      if updated stock level is below the minimum level:
         generate production order (enough quantity to satisfy order, due date)
         update current stock level to 0

iterate over stock levels:
   if stock level is below the minimum level:
       generate production order (enough quantity to return to maximum level)
```

A motivating example is given, showing the way the MTS orders are generated:

Consider the example problem which accompanied the previous section. Table 5 presents the example's stock levels. Running the above pseudocode using the data from Tables 5 and 6, results in the production orders shown in Table 6.

In this case, orders 9, 10 and 14 could not be fulfilled by the initial stock levels. Therefore, they each have their own production order in the new order set (orders 0, 1 and 2). In this case, all products' stock levels were taken below the minimum level except for product 3, so 4 production orders were generated - one for each product that needs replenishment.

Table 5 – Example stock levels

Product	Initial	Minimum	Maximum
1	120	100	200
2	160	100	200
3	100	100	200
4	155	100	200
5	200	100	200

From Table 12, the production orders used for the MTS production strategy, the initial order set of 15 has been reduced to a set of only 7 orders, which have a much higher average due date than the initial order set due to the replenishment of stock having no real priority over orders which could not be fulfilled by stock alone, and thus having an arbitrarily large value.

Table 6 – Example set of orders generated for the MTS production strategy

Order ID	Product	Quantity	Due
0	1	50	200
1	3	20	200
2	1	50	350
3	1	180	9999999
4	2	160	9999999
5	4	170	9999999
6	5	190	9999999

4.2.2. Make to Order

The make to Order production strategy is much more direct, as production orders are taken directly from FIMA orders, meaning that in a MTO strategy applied to the example problem, production orders would be represented by Table 6.

As discussed previously, this strategy is preferable if applicable, as it reduces inventory and management costs. However, there can be disadvantages, such as incurring in unnecessary setup times when there are multiple orders referring to the same products.

For example, for the orders in Table 5, it is entirely possible that orders 3 and 4 are scheduled on different machines or in the same machine but not consecutively. This would mean that both orders of product 7 would be incurring in setups. Should they be aggregated, orders of product 7 would only generate one setup time. Though this multiple-setup case is a possibility, it is also possible that the algorithm would schedule 3 and 4 consecutively so as to prevent generating multiple setups for orders the same product.

Further conclusions on the effectiveness of MTO vs MTS will be discussed in the next chapter.

5. Model Tuning and Results Analysis

The present chapter is concerned with detailing the model built for the implementation of the MOTSA. The first section details the model's input data and presents a brief sensitivity analysis which was carried out to tune the model's parameters. The second section describes the different problem instances tested. The final section discusses the results obtained from the application of the model to the problem instances proposed.

5.1. Model Inputs

After having characterized the algorithm's framework, the model needs to be built. To run the model, the necessary inputs are the problem data and the model parameters.

Because meta-heuristics are very problem-dependent, tuning of the model parameters is extremely important. Furthermore, the quality of results produced greatly depends on the problem instance being solved. Therefore, for a thorough test of the model's capabilities, several problem instances should be tested. Since it is impractical to tailor the model's parameters to every instance that is going to be studied, tuning of the parameters should be done on a representative order set. First, the problem data is presented. Then, the model parameters are discussed

The problem data used for testing the model is based on LSI's actual manufacturing system. For all instances, 30 products are considered, namely 15 bottoms and 15 covers. As mentioned in the second chapter, products vary in mold, color and label. Bottoms are manufactured in machines 1 through 5, while covers are manufactured in machines 6 through 8. The processing times table the setup times table are presented in the Annex X and Y, respectively, and are immutable, as the manufacturing context is the same. Slight changes will be done in later test instances for a thorough test of the model's capabilities, such as disabling one of the machines or assuming different setup times.

The order sets must vary from instance to instance since in a real manufacturing environment, this is where most of the uncertainty comes from. The test sets generated will be discussed further.

The parameters that the algorithm requires are:

- Maximum running time
- Maximum number of iterations
- Neighborhood size
- Maximum non-improving iterations
- Maximum size of assignments tabu list
- Maximum size of sequencing tabu list

Tuning of these parameters will be discussed in the following section.

5.2. Parameter Tuning

This section describes the sensitivity analysis performed for the model's parameters. As mentioned previously, meta-heuristics require a somewhat high degree of tailoring to ensure the best results are obtained in every execution of the model and in reasonable CPU time.

Before presenting the sensitivity analysis in detail, it is first convenient to describe the test instance and the initial solution that is generated for it. LSI provided a typical weekly order sheet, consisting of 40 orders. This order sheet can be seen in Annex X. The initial solution for this order sheet was then generated, with a Makespan of 2430 minutes and a Total Tardiness of 430 minutes. This is a good test instance because it allows testing to occur on every component of the algorithm. The production strategy used in the testing instance for tuning is the MTO. This is relevant because the initial solution generation procedure is capable of generating solutions that have 0 total tardiness. Also, the MTO production strategy is more demanding and thus a better test for the model.

Having defined the problem instance used for parameter tuning and describing the quality of the resulting initial solution, the next step is the definition of the model's parameters. The flowchart presented in Figure 18 illustrates the process of parameter tuning.

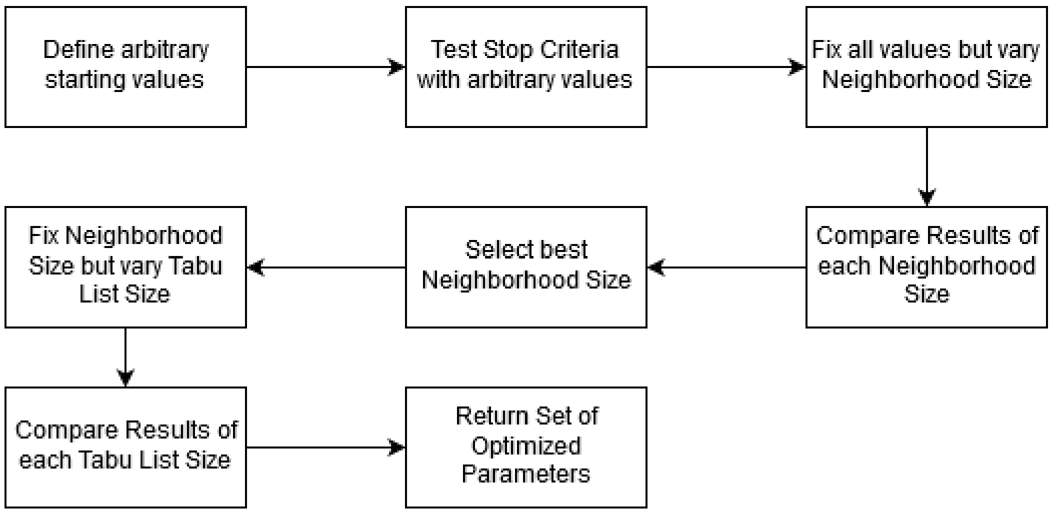

Define arbitrary starting values
Test Stop Criteria with arbitrary values
Fix all values but vary Neighborhood Size
Compare Results of each Neighborhood Size
Select best Neighborhood Size
Fix Neighborhood Size but vary Tabu List Size
Compare Results of each Tabu List Size
Return Set of Optimized Parameters

Figure 18 – Flowchart of parameter tuning process

Stop Criteria

First and foremost, a reasonable CPU time must be defined. Taking into account the fact that scheduling of weekly production is done manually by a team of schedulers, it is clear that even a CPU time that might have been large for a regular meta-heuristic algorithm, such as 10 minutes, for example, would be a great improvement for LSI as it would free the schedulers to perform other tasks. As will be shown briefly, 5 minutes is also enough time for the MOTSA to reach very promising solutions for the instance tested.

It should be noted, however, that CPU time should not be the only stop criterion used. Suppose that a certain instance allows for very fast iterations due to, for example, a smaller size, meaning a much smaller neighborhood to scan. In this scenario, high quality solutions can be reached very quickly, meaning there would no need to run the model for such a long time. Therefore, a combination of both stop criteria offers a good compromise between CPU time and solution quality for diverse problem instances.

Furthermore, the CPU time greatly depends on the problem at hands. Large problem instances (in both number of orders and number of machines) will require a longer time to scan the neighborhood searching for promising solutions, thus resulting in longer CPU times to reach the best solution. For this reason, stop criteria will depend upon the problem instance the model is being applied to. This is relevant because, even though the instance being used to tune the parameters is representative of a typical set of FIMA orders and a good way to tune the model's parameters, the uncertainty LSI operates under makes it possible to have much larger sets of orders from FIMA. Also, since the MTS production strategy is less complex due to having fewer orders to schedule, the CPU time is likely to be much lower. Once the model's parameters have been tuned, a few executions will also be run on the MTS problem and the CPU times will be analyzed. This will prevent the model from running unnecessarily long amounts of time whenever applied to a MTS instance.

Neighborhood Size

Secondly, the next most important parameter to be tuned is the neighborhood size. The neighbor functions developed for both tardiness and makespan optimization are somewhat restrictive, in the sense that non-promising moves are only accepted within a given quality interval relative to the seed solution. Note that the makespan neighbor function is more restrictive than the tardiness neighbor function, given that the makespan neighbor function can never allow a solution that has a non-null total tardiness.

With the restrictive nature of the neighbor functions in mind, it is clear that generating a neighborhood takes a longer CPU time the greater the neighborhood size is. On the other hand, a small neighborhood can also be detrimental to the algorithm's performance, as selecting the first (or first few) acceptable neighbor(s) before scanning for more promising ones can steer the algorithm toward sup-optimal regions of the solution space. There should be a balance between depth (which influences quality) and CPU time.

To determine the most appropriate size for the neighborhood, the test instance was run five times for each neighborhood size, testing sizes of 1, 3, 5, 10 and 15. This procedure should allow for a decent understanding of the impact of the neighborhood size on the solution quality and CPU time.

The values of other parameters of the model were fixed for every execution and were arbitrarily determined:

- The Maximum Non-Improving Iterations was set to 3 because, due to the large size of the entire *observable* neighborhood, randomization was required to guarantee a diverse search of the neighborhood. However, since the neighborhood size might not be enough to hold all promising neighboring solutions, it is possible that no promising solutions are found for a few iterations. 3 iterations is a reasonable value that bridges the compromise between a long search of the neighborhood for the same seed solution and a very fast change of the neighborhood.

- The Tabu lists' sizes were also arbitrarily determined to be 10. For the problem size considered (40 orders and 8 machines), 10 tabu assignments offers a good memory structure forbidding certain orders to be scheduled on bad machines, and 10 tabu assignments offers a good memory structure forbidding consecutive order pairs.

Table 7 summarizes the results of the 25 executions of the model with the intent of determining the optimal neighborhood size. This table is obtained by gathering simple statistical measures from the 5 executions of each neighborhood size. From this table it is not easy to determine whether this is the best neighborhood size, since high quality solutions are reached for different neighborhood sizes. Notably, 3 and 15 . Neighborhood sizes of 1, 5 and 10 returned inconsistent results. It is entirely possible that these executions resulted in worse solutions, and therefore lower consistency, due to the random nature of the algorithm, and not necessarily due to the neighborhood size.

Table 7 - Summary table of neighborhood size sensitivity analysis

Neighborhood Size	**1**	**3**	**5**	**10**	**15**
Minimum Makespan	1380	1330	1330	1330	1315
Average Makespan	1442	1344	1447	1474	1325
Makespan $\frac{Std.Deviation}{Average}$	4%	1%	8%	10%	0%
Minimum Total Tardiness	0	0	0	0	0
Average Total Tardiness	22	0	97	158	0
Tardiness $\frac{Std.Deviation}{Average}$	122%	-	158%	122%	-
Minimum Iterations	118	106	52	51	73
Average Iterations	135.8	121	72	69.6	79.2

For a more thorough analysis, it is convenient to see what is happening inside each execution. Table 8 depicts the results for the 5 executions of the model with the neighborhood size of 3. Figure 19 shows the evolution of the quality of solutions throughout iterations of an execution, for example execution number 3. The left axis refers to the Makespan, while the right axis corresponds to the Total Tardiness. Both axes are in minutes. From the graph it can be seen that, while the algorithm performed 131 iterations, running until the maximum running time (300 seconds), the best solution was found in iteration number 42, after 100 seconds of running.

Table 8 – Neighborhood size sensitivity analysis for neighborhood size = ***3***

Execution Number	Neighborhood Size	Makespan	Tardiness	Run Time	Iterations
1	3	1370	0	301.3041	106
2	3	1360	0	300.6045	115
3	3	1330	0	301.5834	131
4	3	1330	0	300.2662	126
5	3	1330	0	300.3823	127

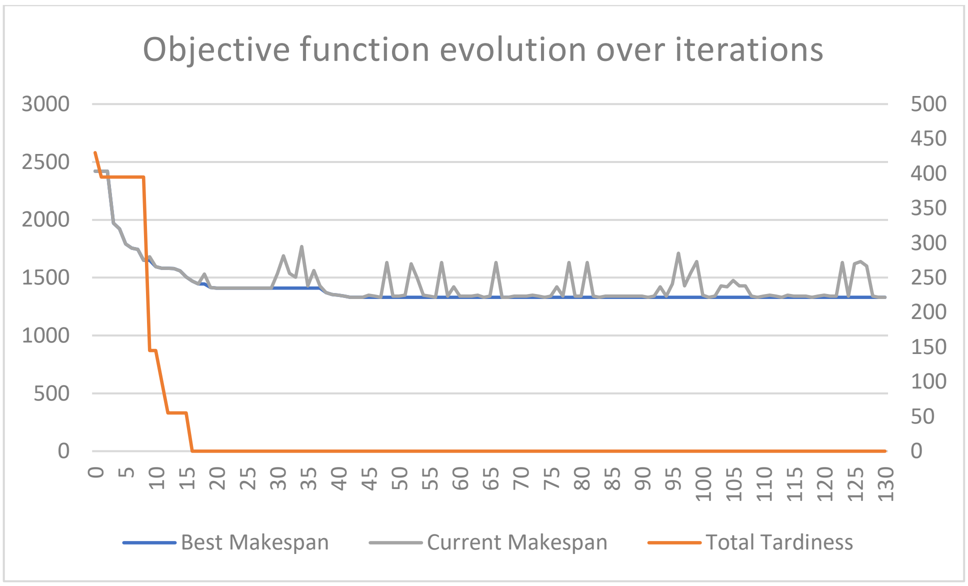


Figure 19 - Evolution of Objective function values over time (Neighborhood Size = 5)

Tabu Lists Sizes

Third, a sensitivity analysis was done to determine the best size for the tabu lists. The size of the tabu lists is also of great importance because it is the memory structure that guides the search by restricting certain moves. A tabu list that is too small will not have a significant impact as it will not be restrictive. On the other hand, a tabu list that is too big will be too restrictive and substantially impair the search.

To determine the optimal size of tabu lists, the same test instance was run 5 times for each size. The size of both tabu lists is equal in all executions. The values being tested are 10, 15, 20 and 25.

The Maximum Non-Improving Iterations is set to 3, for the same reasons mentioned above, and the neighborhood size is 3, which was the value determined in the previous section.

From the observation of Table 9, it can be seen that, as with the neighborhood size, varying the size of the tabu lists does not seem to have a great impact on the solution quality. The results for the Tabu List sensitivity analysis results are very consistent across all neighborhood sizes considered, suggesting that the Tabu List size is not as determining as the neighborhood size. The differences in results between the different Tabu List sizes are not significant, though the average makespan and tardiness varies very slightly between them.

It is possible to see that the best solution found was using a Tabu List size of 15. However, the average makespan was lower for the Tabu List size of 20. Also, standard deviation is 1% lower and the number of iterations is greater in this case. For these reasons, the Tabu List size of 20 was selected as the appropriate value to use in every subsequent execution of the model.

Table 9 – Tabu List size sensitivity analysis results

Tabu List Size	**10**	**15**	**20**	**25**
Minimum Makespan	1330	1290	1302.5	1290
Average Makespan	1348	1340	1331.5	1359
Makespan $\frac{Std.Deviation}{Average}$	3%	3%	2%	6%
Minimum Total Tardiness	0	0	0	0
Average Total Tardiness	0	0	0	21
Tardiness $\frac{Std.Deviation}{Average}$	0%	0%	0%	200%
Minimum Iterations	73	79	102	69
Average Iterations	113.2	122.2	136.8	118.2

Finally, Table 10 summarizes the model's parameters after tuning.

Table 10 – Tuned Model Parameters

Maximum CPU Time*	600
Maximum Number of Iterations*	500
Neighborhood Size	3
Maximum Non-Improving Iterations	3
Assignment Tabu List Size	20
Sequencing Tabu List Size	20

**Maximum CPU Time and Maximum Number of Iterations are set to an arbitrarily high value. However, should CPU time be a concern, a small check can be done on the problem type and determine a more appropriate (lower) value for the stop criteria which still guarantees a consistent solution quality.*

5.3. Problem Instances

Having defined the parameters to be used in the model, it is necessary to decide upon the characteristics that can be present in problem instances to thoroughly test the algorithm. To do this, the instance used for parameter tuning should be analyzed:

- 40 orders
- Average of 8887.5 units per order
- Average due date of 1650 minutes (starting from the beginning of production)
- Approximately uniform distribution of products per order (average product index of 14.075 in a total of 30 different products)

Due to the time limitations in the development of this dissertation, it is very important to define a select few problem instances which can provide a good basis for testing different problem types. Product distribution will be uniform for all different instances, meaning only the number, quantity and due dates of orders are changed. So, the following problem instances were randomly generated within limiting thresholds: a larger problem instance but with lower quantities and more lax due dates; a problem instance with the same size, but smaller orders with tighter due dates; one is the same problem instance used for parameter tuning, but without the availability of machine 5; finally, a much larger problem instance with similar quantities and due dates. Table 11 below summarizes relevant characteristics of the instances created.

Table 11 – Instance descriptions

Instances	Number of Orders	Average Quantity	Average Due Date (minutes)	Number of Machines
1	70	2161.49	1765.7	8
2	30	9796.67	853.3	8
3	40	5000	880	8
4	40	8887.5	1765.7	7
5	100	7189	2091	8

Furthermore, it would be interesting to see how the algorithm would behave when faced with asymmetrical setup times as opposed to the symmetrical setup times presented earlier. The asymmetrical setup times table is presented in Appendix 4. Therefore, each of the problem instances is also run the same number of times with the asymmetrical setup times for both MTO and MTS production strategies.

In total, 6 problem instances (the case study and the 5 presented above) are considered in both MTO and MTS production strategies and also with 2 different sets of setup times, making a total of 6 x 2 x 2

= 24 sets of executions. Each of the problem instances above is run 30 times to ensure a statistically significant set of results. The results are discussed in the following section.

5.4. Results Analysis

In this section, the model's results for the problem instances previously presented are discussed and conclusions are drawn regarding the effectiveness of the algorithm when facing different problem characteristics.

Table 12 summarizes the results obtained for each instance in the MTO production strategy considering the original, symmetrical setup times.

The algorithm dealt well with a higher number of (smaller and laxer) orders, always reaching the Total Tardiness optimum (zero) and achieving a reasonable Makespan for the volume in demand.

The algorithm also produced good results with a lower number of orders, where orders are larger and have tighter due dates, being very quick to achieve very low Makespan solutions and optimal Total Tardiness.

Despite achieving the optimal Total Tardiness in most iterations, some executions of the model for the instance with equal quantities and tighter due dates did not reach this optimum, which is curious since the due dates are, on average, higher than in instance 2 and the average quantity is almost as low as half that of instance 2. The most likely explanation for such a different behavior in a seemingly less complex problem instance is the problem size, since instance 3 has 10 more orders (+33%) than instance 2.

When solving the instance used for parameter tuning with the restriction of machine 5 being unavailable, all solutions found have a high tardiness and relatively high makespan. This was expected, as machine 5 has been used extensively in solutions for other problem instances.

Finally, the largest problem instance was indeed a great challenge for the MOTSA. All executions of the model on this problem instance had a very high Total Tardiness, and the Makespan is also very high when compared to that of other problem instances. Again, this result was expected since this problem instance in specific was built to exhaustively test the model. Note that this result is obtained even with a CPU time that is very high when compared to the CPU times required for other problem instances.

Table 12 – Summarized results for all instances with the Symmetrical Setup Times (MTO)

Symmetrical Setup Times - MTO						
Instance Number	**Minimum Makespan**	**Average Makespan**	**Minimum Total Tardiness**	**Average Total Tardiness**	**0 Tardiness Frequency**	**Average CPU Time**
Case Study	1282.5	1349.5	0	0	100%	267
1	850	994.36	0	0	100%	300
2	880	887.33	0	0	100%	301
3	910	931.33	0	4	87%	304
4	1830	1833.3	395	395	0%	303
5	2747	2910.4	302	425.53	0%	663

To illustrate the model's consistency, Figure 20 shows the best objective makespan values for each run compared with the average. Tardiness is not shown because it always reaches the optimum. The makespan however, does not always reach the lowest possible value. Therefore there is some variability in the solution quality, however it is very low. In fact it is never larger than 5% in module and the average is 0%.

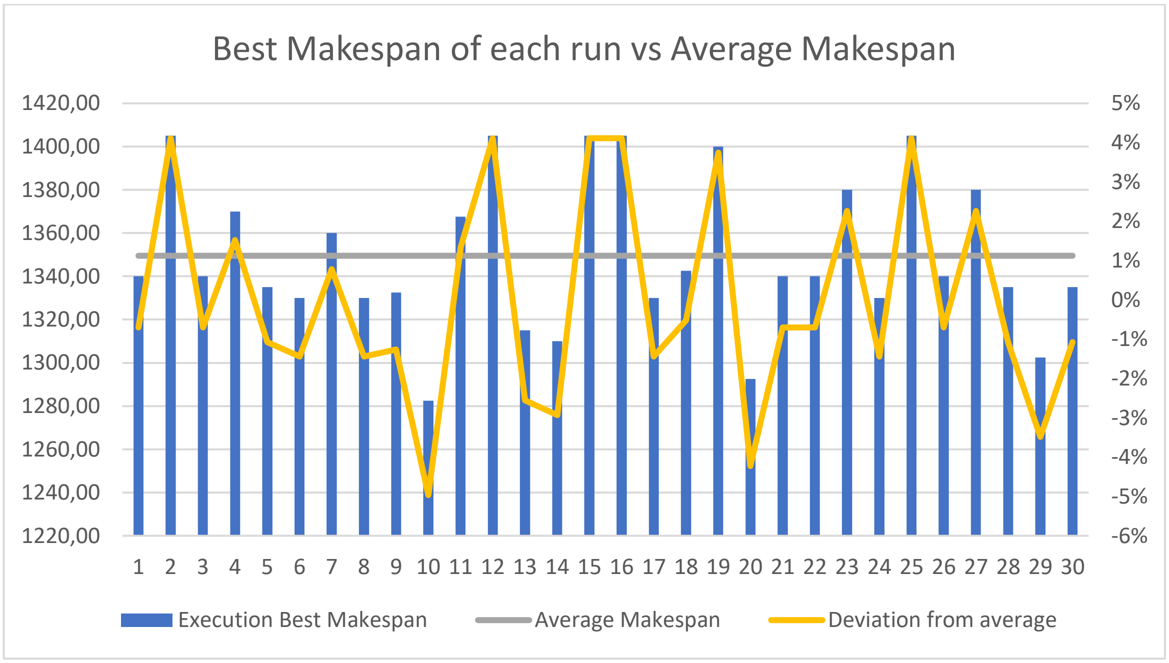


Figure 20 - Comparison of each execution's results with the average results for the Case Study instance in the MTO strategy

Table 13 presents the same information as Table 12, but for the Asymmetrical Setup Times case. Upon observing the table, it can be concluded that the algorithm's performance when considering the Asymmetrical Setup Times is actually superior to its performance when considering the Symmetrical Setup Times in some cases. For example, for instances 1, 3 and 5, the best solution's Makespan is lower than in the symmetrical case. Furthermore, in instance 5, the instance for which the model presented the worst results with symmetrical setup times, the asymmetrical case even reaches the optimal tardiness in all executions, where it never reached the optimal Total Tardiness in the symmetrical case. When applied to the case study instance, as well as instances 2 and 4, however, both objective functions are worse than those of the symmetrical setup times case.

Table 13 - Summarized results for all instances with the Asymmetrical Setup Times (MTO)

Asymmetrical Setup Times - MTO						
Instance Number	**Minimum Makespan**	**Average Makespan**	**Minimum Total Tardiness**	**Average Total Tardiness**	**0 Tardiness Frequency**	**Average CPU Time**
Case Study	1602.7	1602.7	1390	156	0%	303
1	906	970.13	0	0	100%	302
2	1340	1340	40	40	0%	301
3	710	896.67	0	0	100%	303
4	1820	1848	420	420	0%	307
5	2944	2944	0	0	100%	603

Next, the results of the MTS production strategy are presented in Table 14. As expected, the MTS version of the problems is easier to solve, due to the lower number of production orders and much longer due dates, if even applicable. In the MTS case, optimal Total Tardiness is achieved in every single execution of the model and the Makespan is lower than in the MTO case across all instances.

Table 14 - Summarized results for all instances with the Symmetrical Setup Times (MTS)

Symmetrical Setup Times - MTS						
Instance Number	**Minimum Makespan**	**Average Makespan**	**Minimum Total Tardiness**	**Average Total Tardiness**	**0 Tardiness Frequency**	**Average CPU Time (seconds)**
Case Study	1170	1201	0	0	100%	56.9
1	532	532.9	0	0	100%	18.3
2	880	880	0	0	100%	42.4
3	700	703.3	0	0	100%	54.9
4	1710	1722	0	0	100%	57.8
5	2344	2344.3	0	0	100%	278.5

Finally, the results of the Asymmetrical Setup Times are shown in Table 15. In the MTS production strategy, the asymmetrical setup times affect the instances differently when compared to the symmetrical setup times. In the MTS case, only instance 2 has a worse solution in the asymmetrical case when compared to the symmetrical case.

Table 15 - Summarized results for all instances with the Asymmetrical Setup Times (MTS)

Asymmetrical Setup Times - MTS						
Instance Number	**Minimum Makespan**	**Average Makespan**	**Minimum Total Tardiness**	**Average Total Tardiness**	**0 Tardiness Frequency**	**Average CPU Time**
Case Study	1120	1131.75	0	0	100%	52
1	510	516.7	0	0	100%	14.9
2	970	970	0	0	100%	39.4
3	600	614.3	0	0	100%	23.8
4	1450	1450	0	0	100%	54.8
5	2634	2634	0	0	100%	302.6

For a consistency analysis of the MTS strategy, consider, for example, the results of the case study instance with asymmetrical setup times. Figure 22 shows the best objective makespan values for each run compared with the average. By observing Figure 21, again, the conclusion is that the results are very consistent. The lowest makespan is found in many executions and the deviation from the average is never higher than 5% of the average.

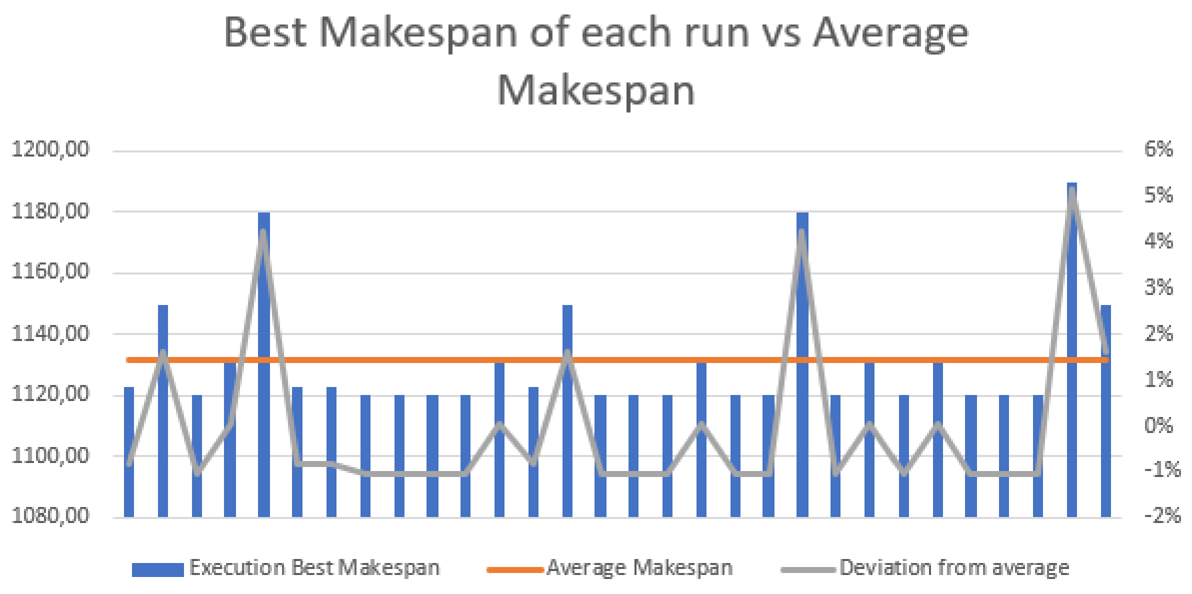


Figure 21 - Comparison of each execution's results with the average results for the Case Study instance in the MTS strategy

6. Conclusions and Future Work

Logoplaste is a Portuguese plastic container manufacturer. Given the rising competitiveness, efficient production scheduling became a priority for Logoplaste's unit LSI. The scheduling problem for the type of manufacturing environment used in LSI is a Flexible Job Shop Scheduling Problem (FJSSP) with Sequence Dependent Setup Times (SDST).

In the FJSSP, a set of jobs needs to be scheduled in a set of machines with the intent of minimizing a given objective function(s). The FJSSP is an extension of the classical JSSP, where there are multiple eligible machines for each operation. This extension increases the complexity of the problem, making the problem effectively a set of two sub-problems: the assignment problem, that is, deciding which machine each order is manufactured on, and the sequencing problem, which deals with determining the best production sequence for a given machine.

The Sequence Dependent Setup Times arise from the fact that products in LSI have three different types of changeover: mold, dye and label. Each of these changeovers incur in a different setup time. Products may share any number of these characteristics. Therefore, the setup times generated by changing production between two products depend on the products in question. The SDST aspect of the problem further increases the complexity of the problem.

Since LSI is under a supply contract with FIMA, fulfilling orders on time is crucial. Therefore, the Total Tardiness is the key criterion in this case study. However, optimizing just the Total Tardiness can lead to high production times. Therefore, the Makespan was also considered as a criterion.

The FJSSP is NP-hard, meaning an exact algorithm can only solve very small problem instances. Since LSI deals with relatively big problem instances, a Meta-Heuristic Algorithm is the most promising approach to solve this case study.

Chapter 3, the literature review, studied both local search-based and population-based Meta-Heuristics, as well as hybrid Meta-Heuristics, for both Mono-Objective and Multi-Objective algorithms. Among local search-based metaheuristics, the Tabu Search and the Simulated Annealing were studied. The only population-based meta-heuristic studied was the Genetic Algorithm due to being the most used and successful. The literature review found that the Mono-Objective Tabu Search had a good performance for this problem and applications of a Multi-Objective Tabu Search were scarce; specifically, no Tabu Search implementations were found considering both Total Tardiness and Makespan. Furthermore, no studies were found implementing a Multi-Objective Tabu Search using the lexicographical method. To address this gap, the algorithm developed in this thesis is a Multi-Objective Tabu Search using the lexicographical method optimizing Total Tardiness first and then the Makespan.

Chapter 4 describes the algorithm's framework and details the key components of the Tabu Search algorithm: initial solution, neighbor function, tabu lists, diversification strategies and finally, the general structure of the algorithm. In chapter 5, the model's inputs are described, and a sensitivity analysis is done to tune the model's parameters using a representative problem instance. Furthermore, instances used for testing the algorithm were also created and described. Finally, the model is applied to the all of the instances developed to thoroughly test the algorithm and the results are discussed.

Overall, the model developed in this dissertation achieved satisfactory results, being able to find the optimal solution in terms of Total Tardiness and also achieving a low Makespan, thus validating the Tabu Search as a good Meta-Heuristic to apply to the FJSSP-SDST in a Multi-Objective perspective using the lexicographical method. Both MTS and MTO strategies appear to be viable for LSI, considering the weekly orders do not deviate too much from the typical. In a MTO strategy, though large instances or a machine breakdown prove to be problematic for LSI, it is possible for LSI to coordinate with FIMA to maintain the service level. This is considering FIMA can be predict their demand appropriately and accordingly update the yearly production plans that are sent to LSI. The MTS strategy is superior in terms of both total tardiness and makespan achieved, for all instances. It could be argued that this strategy would incur in higher warehousing costs as well as inventory management costs. However, since LSI is setup near FIMA and stock is only meant to cover weekly orders, warehousing costs tend to be acceptable for LSI.

Due to the time limitations of this dissertation, further work can be identified which would improve the quality of the model:

- The neighbor functions provide an excellent support for a good exploration of the search space; however, the neighborhood generation procedure could be faster. The tabu lists implemented are somewhat limitative in the sense that a fixed tabu list size does not adapt to the search's peculiarities: some areas of the solution space need to be more broadly searched, while others need a more restricted neighborhood.
- It would also be interesting to implement this algorithm within a population-based algorithm such as the Genetic Algorithm, which would help in diversifying the search and finding good areas of the solution space for the Tabu Search to intensify the search on.
- Furthermore, a direct comparison with a GA implementation for this problem could also be done, to understand which of the two most promising Meta-Heuristics for the Multi-Objective FJSSP is effectively the best.

Finally, it should be mentioned that the present dissertation's work has been accepted for submission of an extended article in the international conference ESCAPE Milano. This extended article will be submitted on November, 30 of the present year. Also, an extended article is also being developed for submission to quality journals in the relevant areas of interest.

Bibliography

Aarts, E. H. L., P. J. M. van Laarhoven, J. K. Lenstra, and N. L. J. Ulder. 1994. “A Computational Study of Local Search Algorithms for Job Shop Scheduling.” *ORSA Journal on Computing* 6(2):118–25.

Abdelmaguid, Tamer F. 2015. “A Neighborhood Search Function for Flexible Job Shop Scheduling with Separable Sequence-Dependent Setup Times.” *Applied Mathematics and Computation* 260:188–203.

Applegate, David and William Cook. 1991. “A Computational Study of the Job-Shop Scheduling Problem.” *ORSA Journal on Computing Publication* 3(2).

Artigues, Christian and Dominique Feillet. 2008. “A Branch and Bound Method for the Job-Shop Problem with Sequence-Dependent Setup Times.” *Annals of Operations Research* 135–59.

Azzouz, A., M. Ennigrou, and L. B. Said. 2016. “Flexible Job-Shop Scheduling Problem with Sequence-Dependent Setup Times Using Genetic Algorithm.” *ICEIS 2016 - Proceedings of the 18th International Conference on Enterprise Information Systems* 2(Iceis):47–53.

Azzouz, Ameni, Meriem Ennigrou, Boutheina Jlifi, and Khaléd Ghédira. 2012. “Combining Tabu Search and Genetic Algorithm in a Multi-Agent System for Solving Flexible Job Shop Problem.”

B. J. Lageweg, J. K. Lenstra, A. H. G. Rinnooy Kan. 1977. “Job-Shop Scheduling by Implicit Enumeration.” *Management Science Publication* 24(4):441–50.

Battiti, Roberto and Giampietro Tecchiolli. 1994. “The Reactive Tabu Search.” (December 2014).

Beck, J. Christopher and T. K. Feng. 2011. “Combining Constraint Programming and Local Search for Job-Shop Scheduling.” *INFORMS Journal on Computing* 23(1):1–14.

Blazewicz, Jacek, Erwin Pesch, and Malgorzata B. Sterna. 2000. “Disjunctive Graph Machine Representation of the Job Shop Scheduling Problem.” *European Journal of Operational Research* 127(2):317–31.

Bowman, Edward H. 1959. “The Schedule-Sequencing Problem.” *Operations Research, 7(5), 621–624.* 7(5):621–24.

Brandimarte, Paolo. 1993. “Routing and Scheduling in a Flexible Job Shop by Tabu Search.” *Annals of Operations Research*.

Brucker, Peter, Bernd Jurisch, F. B. Mathematik, Universitiit Osnabriick, and D. Osnabriick Germany. 1994. “A Branch and Bound Algorithm for the Job-Shop Problem *.” *Discrete Applied Mathematics* 49.

C. Scrich, V. Armentano, M. Laguna. 2004. “Tardiness Minimization in a ¯ Exible Job Shop : A Tabu Search Approach.” *Journal of Intelligent Manufacturing* 15:103–15.

Černý, V. 1985. “Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm.” *Journal of Optimization Theory and Applications* 45(1):41–51.

Chen, Haoxun, Jiirgen Ihlow, and Carsten Lehmann. 1999. “A Genetic Algorithm for Flexible Job-Shop Scheduling.” Pp. 1120–25 in *Proceedings of the 1999 IEEE International Conference on Robotics & Automation*.

Chiang, Tsung-che and Hsiao-jou Lin. 2013. “Int . J . Production Economics A Simple and Effective Evolutionary Algorithm for Multiobjective Flexible Job Shop Scheduling.” *Intern. Journal of Production Economics* 141(1):87–98.

Dauzère-pérès, Stéphane and Jan Paulli. 1997. “An Integrated Approach for Modeling and Solving the General Multiprocessor Job-Shop Scheduling Problem Using Tabu Search.” *Annals of Operations Research* 70:281–306.

Dauzere. 2002. “A Modified Simulated Annealing Method for Flexible Job Shop Scheduling Problem.”

Dell’Amico, Mauro and Marco Trubian. 1993. “Applying Tabu Search to the Job-Shop Scheduling Problem.” *Annals of Operations Research*.

Garey, M. R., D. S. Johnson, and Ravi Sethi. 1976. “The Complexity of Flowshop and Jobshop Scheduling.” *Mathematics of Operations Research* 1(2):117–29.

Glover, Fred. 1998. “Tabu Search !” 3:621–757.

Graves, Stephen C. 1981. “A Review of Production Scheduling.” *Operations Research Publication* 29(4):646–75.

Hillier, Frederick S. 2001. *INTRODUCTION TO OPERATIONS RESEARCH*. 7th ed.

Holland, John H. 1984. “No Title.” 317–33.

J. Carlier, E. Pinson. 1989. “Carlier1989.Pdf.” *Management Science* 35(2):164–76.

Jia, Shuai and Zhi-hua Hu. 2014. “Computers & Operations Research Path-Relinking Tabu Search for the Multi-Objective Fl Exible Job Shop Scheduling Problem.” *Computers and Operation Research* 47:11–26.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi. 1983. “Optimization by Simulated Annealing S.” *Science* 220:671--680.

Ku, Wen-yang and J. Christopher Beck. 2016. “Computers & Operations Research Mixed Integer Programming Models for Job Shop Scheduling : A Computational Analysis.” *Computers and Operation Research* 73:165–73.

Laarhoven, Peter J. M. Van, Emile H. L. Aarts, and Jan Karel Lenstra. 1992. “Job Shop Scheduling by Simulated Annealing.” (August 2015).

Li, Junqing, Liaocheng Universtiy, Quan-ke Pan, and Yun-chia Liang. 2010. “An Effective Hybrid Tabu Search Algorithm for Multi-Objective Flexible Job- Shop Scheduling Problems.” *Computers & Industrial Engineering* 59(4):647–62.

Li, Xinyu and Liang Gao. 2016. "Int . J . Production Economics An Effective Hybrid Genetic Algorithm and Tabu Search for Fl Exible Job Shop Scheduling Problem." *Intern. Journal of Production Economics* 174:93–110.

Manne, Alan S. 1960. "On the Job-Shop Scheduling Problem." *Operations Research Publication* 8(2).

Mastrolilli, Monaldo and Luca Maria Gambardella. 2000. "Effective Neighbourhood Functions for the Flexible Job Shop Problem." *Journal of Scheduling* 3(1):3–20.

Mesghouni. 1997. "Evolution Scheduling." 0–5.

Naderi, B., S. M. T. Fatem. Ghomi, and M. Aminnayeri. 2010. "A High Performing Metaheuristic for Job Shop Scheduling with Sequence-Dependent Setup Times." *Applied Soft Computing Journal* 10(3):703–10.

Nowicki, Eugeniusz and Czeslaw Smutnicki. 1996. "A Fast Taboo Search Algorithm for the Job Shop Problem." (September 2014).

Nowicki, Eugeniusz and Czesław Smutnicki. 2005. "An Advanced Tabu Search Algorithm for the Job Shop Problem." *Journal of Scheduling*.

Pezzella, F., G. Morganti, and G. Ciaschetti. 2008. "A Genetic Algorithm for the Flexible Job-Shop Scheduling Problem." 35:3202–12.

Raaymakers, W. H. M. 2000. "Scheduling Multipurpose Batch Process Industries with No-Wait Restrictions by Simulated Annealing." 126.

Saidi-Mehrabad, Mohammad and Parviz Fattahi. 2007. "Flexible Job Shop Scheduling with Tabu Search Algorithms." *International Journal of Advanced Manufacturing Technology*.

Shen, Liji, Stéphane Dauzère-Pérès, and Janis S. Neufeld. 2018. "Solving the Flexible Job Shop Scheduling Problem with Sequence-Dependent Setup Times." *European Journal of Operational Research* 265(2):503–16.

Vilcot, Geoffrey and Jean-charles Billaut. 2011. "A Tabu Search Algorithm for Solving a Multicriteria Flexible Job Shop Scheduling Problem." *International Journal of Production Research ISSN:* 49(23):6963–80.

Wagner, H. M. 1959. "An Integer Linear-Programming Model For Machine Scheduling." *Naval Research Logistics* 131–40.

Zhang, Chao Yong, Peigen Li, Yunqing Rao, and Zailin Guan. 2008. "A Very Fast TS / SA Algorithm for the Job Shop Scheduling Problem." *Computers & Operations Research* 35:282–94.

Zhang, Guohui, Yang Shi, and Liang Gao. 2008. "A Genetic Algorithm and Tabu Search for Solving Flexible Job Shop Schedules." 369–72.

Zhu, Zhiwei and Ronald B. Heady. 2000. “Minimizing the Sum of Earliness / Tardiness in Multi-Machine Scheduling : A Mixed Integer Programming Approach.” *Computers & Industrial Engineering* 38:297–305.

Appendix

Appendix 1 – Problem Data

Problem Size Parameters

Number of Machines	8
Number of Orders	40
Number of Products	30

Processing times of products on each eligible machine (minutes)

		Product														
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Machine	**1**	1	1	1	1	1	1	1	1	1	1	0.04	0.04	0.04	0.04	0.04
	2	0.02	0.02	0.02	1	1	1	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
	3	1	1	0.015	1	0.015	0.015	1	0.015	0.015	1	1	1	1	1	1
	4	0.02	0.02	0.02	0.02	0.02	1	0.02	0.02	0.02	0.02	1	1	1	1	1
	5	0.025	0.025	0.025	0.025	0.025	1	1	1	1	1	0.025	0.025	0.025	0.025	0.025

		Product														
		16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Machine	**6**	1	1	1	0.02	0.02	0.02	1	1	1	1	1	0.02	1	1	1
	7	1	1	1	1	1	1	0.007	0.007	0.007	0.007	0.007	1	1	1	1
	8	0.01	0.01	0.01	1	1	1	1	1	1	1	1	1	0.01	0.01	0.01

Types of Setup Times

	Label	**Dye**	**Mold**
Time (minutes)	10	100	300

Setup Times matrix (minutes)

Products	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	10	10	300	300	300	300	300	300	300	300	300	300	300	300
2	10	0	10	300	300	300	300	300	300	300	300	300	300	300	300
3	10	10	0	300	300	300	300	300	300	300	300	300	300	300	300
4	300	300	300	0	10	10	300	300	300	300	300	300	300	300	300
5	300	300	300	10	0	10	300	300	300	300	300	300	300	300	300
6	300	300	300	10	10	0	300	300	300	300	300	300	300	300	300
7	300	300	300	300	300	300	0	10	10	10	300	300	300	300	300
8	300	300	300	300	300	300	10	0	10	10	300	300	300	300	300
9	300	300	300	300	300	300	10	10	0	10	300	300	300	300	300
10	300	300	300	300	300	300	10	10	10	0	300	300	300	300	300
11	300	300	300	300	300	300	300	300	300	300	0	10	10	10	10
12	300	300	300	300	300	300	300	300	300	300	10	0	10	10	10
13	300	300	300	300	300	300	300	300	300	300	10	10	0	10	10
14	300	300	300	300	300	300	300	300	300	300	10	10	10	0	10
15	300	300	300	300	300	300	300	300	300	300	10	10	10	10	0

Products	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
16	0	100	100	300	300	300	300	300	300	300	300	300	300	300	300
17	100	0	10	300	300	300	300	300	300	300	300	300	300	300	300
18	100	10	0	300	300	300	300	300	300	300	300	300	300	300	300
19	300	300	300	0	100	10	300	300	300	300	300	100	300	300	300
20	300	300	300	100	0	100	300	300	300	300	300	10	300	300	300
21	300	300	300	10	100	0	300	300	300	300	300	100	300	300	300
22	300	300	300	300	300	300	0	10	100	100	300	300	300	300	300
23	300	300	300	300	300	300	10	0	100	100	300	300	300	300	300
24	300	300	300	300	300	300	100	100	0	100	300	300	300	300	300
25	300	300	300	300	300	300	100	100	100	0	300	300	300	300	300
26	300	300	300	300	300	300	300	300	300	300	0	300	300	300	300
27	300	300	300	100	10	100	300	300	300	300	300	0	300	300	300
28	300	300	300	300	300	300	300	300	300	300	300	300	0	100	100
29	300	300	300	300	300	300	300	300	300	300	300	300	100	0	10
30	300	300	300	300	300	300	300	300	300	300	300	300	100	10	0

Case Study set of Orders from FIMA

Order ID	1	2	3	4	5	6	7	8	9	10
Product	2	3	4	1	5	3	4	6	2	5
Quantity	10000	6000	8000	6000	2000	12000	10000	1000	4500	1000
Due Date	1400	1500	1000	1600	2000	2100	1200	1850	1000	1400

Order ID	11	12	13	14	15	16	17	18	19	20
Product	7	8	9	10	10	12	13	14	11	15
Quantity	20000	30000	10000	8000	30000	10000	6000	8000	6000	2000
Due Date	700	1200	1350	800	1400	3500	2500	950	3000	2700

Order ID	21	22	23	24	25	26	27	28	29	30
Product	13	14	16	12	15	17	18	19	20	20
Quantity	12000	10000	1000	4500	1000	20000	30000	10000	8000	20000
Due Date	900	2000	1200	5600	900	1450	1400	1100	1700	800

Order ID	31	32	33	34	35	36	37	38	39	40
Product	22	30	29	28	25	26	27	21	23	24
Quantity	5000	7000	5500	6000	2500	1000	2500	8000	9000	2000
Due Date	2000	1500	800	1500	1750	1900	2500	900	1450	1500

Case Study Stock Levels

Product	1	2	3	4	5	6	7	8	9	10
Initial	30000	25000	10000	15000	12000	25000	13000	5000	20000	50000
Minimum	25000	24000	9000	15000	10000	24000	10000	4000	18000	50000
Maximum	32000	30000	11000	20000	14000	28000	15000	10000	30000	60000

Product	11	12	13	14	15	16	17	18	19	20
Initial	30000	25000	10000	15000	12000	25000	13000	5000	20000	50000
Minimum	25000	24000	9000	15000	10000	24000	10000	4000	18000	50000
Maximum	32000	30000	11000	20000	14000	28000	15000	10000	30000	60000

Product	21	22	23	24	25	26	27	28	29	30
Initial	15000	12000	25000	13000	5000	20000	50000	30000	25000	50000
Minimum	15000	10000	24000	10000	4000	18000	50000	25000	24000	50000
Maximum	20000	14000	28000	15000	10000	30000	60000	32000	30000	60000

Appendix 3 – Sensitivity Analysis Results

Neighborhood Size Sensitivity Analysis Results

Execution Number	Neighborhood Size	Maximum Non-Improving Iterations	Assignment Tabu List Size	Sequencing Tabu List Size	Best Makespan	Best Tardiness	Run Time	Iterations
1	1	3	10	10	1510	55	301	154
2	1	3	10	10	1520	55	301	156
3	1	3	10	10	1420	0	303	118
4	1	3	10	10	1380	0	303	125
5	1	3	10	10	1380	0	303	126
6	3	3	10	10	1370	0	301	106
7	3	3	10	10	1360	0	301	115
8	3	3	10	10	1330	0	302	131
9	3	3	10	10	1330	0	300	126
10	3	3	10	10	1330	0	300	127
11	5	3	10	10	1330	0	306	88
12	5	3	10	10	1495	90	300	68
13	5	3	10	10	1420	0	304	54
14	5	3	10	10	1650	395	301	52
15	5	3	10	10	1340	0	302	98
16	10	3	10	10	1650	395	304	51
17	10	3	10	10	1360	0	301	73
18	10	3	10	10	1650	395	304	56
19	10	3	10	10	1380	0	302	87
20	10	3	10	10	1330	0	302	81
21	15	3	10	10	1320	0	303	75
22	15	3	10	10	1330	0	305	76
23	15	3	10	10	1330	0	305	79
24	15	3	10	10	1315	0	303	93
25	15	3	10	10	1330	0	301	73

Tabu List Size Sensitivity Analysis Results

Execution Number	Neighborhood Size	Maximum Non-Improving Iterations	Assignment Tabu List Size	Sequencing Tabu List Size	Best Makespan	Best Tardiness	Run Time (seconds)
1	3	3	10	10	1330	0	303
2	3	3	10	10	1330	0	302
3	3	3	10	10	1330	0	300
4	3	3	10	10	1330	0	300
5	3	3	10	10	1420	0	303
1	3	3	15	15	1330	0	304
2	3	3	15	15	1290	0	301
3	3	3	15	15	1420	0	301
4	3	3	15	15	1330	0	300
5	3	3	15	15	1330	0	302
1	3	3	20	20	1330	0	303
2	3	3	20	20	1330	0	300
3	3	3	20	20	1315	0	302
4	3	3	20	20	1302.5	0	300
5	3	3	20	20	1380	0	303
1	3	3	25	25	1290	0	300
2	3	3	25	25	1315	0	303
3	3	3	25	25	1330	0	303
4	3	3	25	25	1520	105	303
5	3	3	25	25	1340	0	304

Appendix 4 – Test Instances

Instance 1

Order ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Product	1	27	13	14	10	29	25	15	24	12
Quantity	1200	2600	500	2200	1800	3100	700	3200	900	3000
Due Date	2400	2700	1400	2300	2400	1100	700	1800	2200	1100

Order ID	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Product	25	7	25	7	30	23	12	11	24	11
Quantity	3700	1200	2500	1500	2400	600	3400	3500	1400	3200
Due Date	2000	500	2200	2500	600	800	2300	2800	1200	600

Order ID	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Product	29	27	14	22	13	28	7	6	17	17
Quantity	3200	600	500	3500	3400	1800	3200	2600	800	2300
Due Date	2200	600	1900	800	2300	1800	2500	800	500	2200

Order ID	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**
Product	22	15	9	3	20	14	26	18	21	10
Quantity	1800	1800	2700	2200	1900	3800	1200	1000	2300	1500
Due Date	1800	2100	2300	2200	2400	2900	700	900	1900	2900

Order ID	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**
Product	16	15	20	26	3	20	30	4	11	21
Quantity	600	500	3500	1900	3900	1800	3600	2100	3300	2400
Due Date	2500	2200	1500	1300	2600	1800	2900	800	1000	500

Order ID	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
Product	12	11	28	7	21	29	8	26	4	28
Quantity	1200	1700	3600	2000	3800	1400	600	1500	1100	2500
Due Date	1100	1700	1200	1900	2500	1900	700	2200	1600	2400

Order ID	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**	**70**
Product	30	29	19	14	13	3	20	25	11	9
Quantity	2500	3200	2700	900	3100	1600	3200	2500	1600	2300
Due Date	600	1700	2400	1300	2700	2900	1600	2800	2500	1500

Instance 2

Order ID	1	2	3	4	5	6	7	8	9	10
Product	11	18	25	10	23	10	13	23	23	21
Quantity	19800	9400	9500	4600	17600	14900	15700	13800	1300	11700
Due Date	1300	1200	1000	500	1000	900	500	300	300	300

Order ID	11	12	13	14	15	16	17	18	19	20
Product	14	9	10	13	21	19	4	7	25	19
Quantity	2500	11000	5300	6100	4300	10800	2900	1900	3200	16700
Due Date	300	300	1300	1000	1300	1200	700	700	1000	900

Order ID	21	22	23	24	25	26	27	28	29	30
Product	23	9	25	13	6	22	11	17	12	17
Quantity	10600	18200	7200	3500	6700	16600	18600	17900	7400	4200
Due Date	1200	900	500	500	1200	1200	900	800	1100	1300

Instance 3

Order ID	1	2	3	4	5	6	7	8	9	10
Product	16	13	8	12	5	19	29	12	14	25
Quantity	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000
Due Date	800	500	500	900	700	600	1000	1200	1000	700

Order ID	11	12	13	14	15	16	17	18	19	20
Product	3	4	20	3	14	22	3	14	10	6
Quantity	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000
Due Date	800	1000	1300	1100	600	1000	900	700	1000	500

Order ID	21	22	23	24	25	26	27	28	29	30
Product	14	23	8	10	8	11	7	23	17	13
Quantity	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000
Due Date	1300	1100	500	700	700	900	1000	800	1100	600

Instance 4 is the same as the Case Study Order presented in Appendix 1, except machine 5 is unavailable for manufacturing.

Instance 5

Order ID	1	2	3	4	5	6	7	8	9	10
Product	22	10	30	6	29	23	6	25	7	15
Quantity	13400	6900	8000	10100	6200	1700	4600	3000	7500	12000
Due Date	2700	500	2200	1500	2100	3900	3500	1600	1700	3600

Order ID	11	12	13	14	15	16	17	18	19	20
Product	22	9	14	14	16	24	16	3	24	19
Quantity	7800	9700	1400	2200	7700	8500	3200	3700	13800	12200
Due Date	2500	3400	2700	900	900	3100	3800	2400	3800	2100

Order ID	21	22	23	24	25	26	27	28	29	30
Product	19	23	13	16	28	24	29	5	18	12
Quantity	14800	5800	3200	2800	9400	3200	13800	8600	5500	5700
Due Date	3600	1100	1800	600	800	3300	3200	2600	2300	700

Order ID	31	32	33	34	35	36	37	38	39	40
Product	15	24	25	5	1	8	11	19	1	10
Quantity	1600	3700	3500	8800	11200	6600	9800	10200	11900	14200
Due Date	3800	3600	700	2600	3200	900	1500	2500	1600	900

Order ID	41	42	43	44	45	46	47	48	49	50
Product	5	4	6	30	7	27	15	12	28	15
Quantity	8800	2000	4700	9600	8000	7600	6400	1500	3200	11100
Due Date	2000	1900	3200	3000	3400	3000	3900	1400	1800	2700

Order ID	51	52	53	54	55	56	57	58	59	60
Product	17	21	9	10	3	10	23	16	23	15
Quantity	8900	10300	2000	1200	12200	2500	10400	5800	5100	10700
Due Date	900	2400	1600	1200	2700	3800	1700	1300	1600	2700

Order ID	61	62	63	64	65	66	67	68	69	70
Product	1	3	7	28	10	9	12	23	21	3
Quantity	7500	2600	5400	4300	8700	2700	5000	2100	5800	10400
Due Date	3500	900	3400	1900	2200	1200	1800	1800	3800	1300

Order ID	71	72	73	74	75	76	77	78	79	80
Product	30	30	26	9	12	20	29	10	20	26
Quantity	3100	7100	8500	1700	14900	13400	6100	3300	13500	3800
Due Date	2200	800	800	700	1900	600	1000	2700	1600	700

Order ID	81	82	83	84	85	86	87	88	89	90
Product	23	28	24	18	6	14	9	28	27	30
Quantity	2300	7500	1300	6300	11100	3000	1300	14400	14800	8300
Due Date	1300	1500	900	1600	1600	1000	1800	2100	1000	3700

Order ID	91	92	93	94	95	96	97	98	99	100
Product	30	21	6	8	3	1	8	4	11	14
Quantity	9500	8600	9400	11400	6800	9500	11300	10800	2900	8600
Due Date	1600	3400	2100	3500	2200	1100	3100	500	1600	2300

Finally, the asymmetrical setup times matrixes are shown below. These matrixes are applied to all instances described.

Products	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	300	300	300	10	300	300	300	300	300	300	300	300	300	300
2	10	0	10	300	10	300	300	300	300	100	300	300	10	300	10
3	300	300	0	10	100	300	300	300	300	300	10	300	300	300	300
4	300	300	300	0	300	300	300	300	300	300	100	300	100	300	300
5	300	300	300	300	0	300	300	300	10	300	300	300	300	10	100
6	300	300	10	10	300	0	300	100	300	300	100	10	300	300	100
7	300	300	100	10	300	10	0	300	300	100	100	300	300	10	300
8	100	300	10	300	300	300	300	0	100	10	300	10	300	300	300
9	300	300	10	300	300	300	300	300	0	300	300	300	10	100	300
10	300	300	10	300	300	300	300	100	300	0	10	10	300	300	100
11	300	10	300	100	300	300	300	10	300	300	0	10	300	300	300
12	10	300	300	300	300	300	10	300	300	10	300	0	300	10	300
13	300	300	300	10	300	300	100	300	10	300	300	300	0	10	10
14	300	300	300	300	300	300	10	10	300	300	10	300	100	0	300
15	300	300	300	300	300	300	10	100	300	100	300	300	10	300	0

Products	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
16	0	300	100	100	300	300	300	300	100	300	300	300	300	300	10
17	300	0	300	300	10	10	300	100	100	300	300	300	10	300	300
18	300	300	0	300	300	10	300	300	10	10	10	300	300	300	300
19	300	300	300	0	300	300	10	300	300	300	300	300	300	300	100
20	10	100	300	300	0	300	100	300	10	10	300	300	300	300	300
21	300	100	300	100	100	0	300	300	300	100	10	300	300	300	300
22	300	300	10	300	300	300	0	300	100	300	10	300	300	10	300
23	300	100	300	100	300	10	300	0	300	10	300	300	300	300	100
24	300	10	300	300	10	300	300	300	0	300	300	300	300	300	100
25	300	10	300	100	10	300	300	10	300	0	300	300	300	10	300
26	300	100	10	100	10	300	300	300	100	10	0	300	10	10	100
27	300	300	300	300	10	10	10	100	300	10	300	0	100	300	300
28	100	300	300	300	10	300	300	300	300	300	300	300	0	10	100
29	300	100	300	300	300	10	10	100	300	300	300	300	300	0	300
30	10	300	300	300	300	100	100	300	300	300	300	300	300	10	0

Stock Levels for the Test

Product	1	2	3	4	5	6	7	8	9	10
Initial	5100	30000	11000	20000	14000	28000	15000	10000	30000	60000
Minimum	5100	24000	9000	15000	10000	24000	10000	4000	18000	50000
Maximum	8000	30000	11000	20000	14000	28000	15000	10000	30000	60000

Product	11	12	13	14	15	16	17	18	19	20
Initial	32000	30000	11000	20000	14000	28000	15000	10000	30000	60000
Minimum	25000	24000	9000	15000	10000	24000	10000	4000	18000	50000
Maximum	32000	30000	11000	20000	14000	28000	15000	10000	30000	60000

Product	21	22	23	24	25	26	27	28	29	30
Initial	20000	14000	28000	15000	10000	30000	60000	32000	30000	60000
Minimum	15000	10000	24000	10000	4000	18000	50000	25000	24000	50000
Maximum	20000	14000	28000	15000	10000	30000	60000	32000	30000	60000

Appendix 6 – MOBSA MTO Results

MOTSA MTO Case Study Instance – Symmetrical Setup Times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	1340	0	258	150
2	1405	0	327	150
3	1340	0	253	150
4	1370	0	244	150
5	1335	0	283	150
6	1330	0	278	150
7	1360	0	242	150
8	1330	0	269	150
9	1333	0	277	150
10	1283	0	201	150
11	1368	0	205	150
12	1405	0	303	150
13	1315	0	282	150
14	1310	0	252	150
15	1405	0	302	150
16	1405	0	293	150
17	1330	0	258	150
18	1343	0	215	150
19	1400	0	265	150
20	1293	0	241	150
21	1340	0	300	150
22	1340	0	316	150
23	1380	0	269	150
24	1330	0	221	150
25	1405	0	219	150
26	1340	0	276	150
27	1380	0	265	143
28	1335	0	233	150
29	1303	0	320	150
30	1335	0	310	150

MOTSA MTO Instance 1 – Symmetrical Setup Times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	850	0	303	82
2	1040	0	303	76
3	940	0	305	65
4	850	0	306	68
5	930	0	302	63
6	1140	0	305	56
7	1140	0	303	50
8	930	0	305	67
9	1140	0	305	49
10	930	0	303	55
11	940	0	301	51
12	1140	0	302	52
13	1040	0	304	44
14	940	0	302	53
15	965	0	303	60
16	850	0	303	82
17	1040	0	305	76
18	940	0	306	65
19	850	0	303	68
20	930	0	305	63
21	1140	0	306	50
22	930	0	306	67
23	1140	0	302	49
24	930	0	303	55
25	940	0	303	51
26	1040	0	301	44
27	940	0	303	53
28	965	0	301	60
29	850	0	304	83
30	930	0	302	63

MOTSA MTO Instance 2 – Symmetrical Setup Times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	902	0	303	76
2	880	0	302	170
3	902	0	302	80
4	902	0	302	78
5	902	0	302	81
6	880	0	300	161
7	880	0	301	169
8	880	0	301	163
9	880	0	301	161
10	880	0	300	164
11	902	0	302	78
12	880	0	300	165
13	880	0	301	154
14	880	0	300	164
15	880	0	301	166
16	880	0	304	127
17	880	0	301	153
18	880	0	304	117
19	880	0	303	118
20	880	0	301	155
21	902	0	302	123
22	880	0	302	143
23	880	0	302	126
24	880	0	302	120
25	902	0	303	157
26	880	0	305	156
27	880	0	300	141
28	902	0	303	135
29	880	0	300	146
30	880	0	302	119

MOTSA MTO Instance 3 – Symmetrical Setup Times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	920	0	302	62
2	920	0	303	109
3	910	0	303	85
4	1020	35	305	70
5	920	0	303	64
6	920	0	301	83
7	920	0	306	62
8	920	0	307	65
9	910	0	305	90
10	920	0	302	71
11	910	0	308	91
12	920	0	300	121
13	1020	35	304	67
14	920	0	301	66
15	920	0	304	64
16	920	0	304	81
17	920	0	304	67
18	910	0	302	89
19	1020	35	300	90
20	920	0	305	80
21	920	0	303	88
22	920	0	302	79
23	920	0	302	82
24	910	0	301	89
25	920	0	305	84
26	910	0	303	86
27	920	0	300	72
28	1020	35	300	87
29	920	0	303	67
30	920	0	303	79

MOTSA MTO Instance 4 – Symmetrical Setup Times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	1830	395	301	48
2	1830	395	303	43
3	1830	395	302	46
4	1830	395	301	43
5	1830	395	305	44
6	1830	395	303	44
7	1850	395	307	48
8	1850	395	303	46
9	1830	395	307	45
10	1840	395	301	50
11	1830	395	301	45
12	1830	395	303	49
13	1830	395	300	51
14	1830	395	303	41
15	1830	395	303	51
16	1830	395	302	51
17	1830	395	302	47
18	1830	395	300	46
19	1830	395	303	50
20	1830	395	302	46
21	1830	395	302	44
22	1830	395	302	50
23	1830	395	305	50
24	1830	395	305	46
25	1830	395	303	46
26	1830	395	304	51
27	1830	395	301	50
28	1830	395	300	48
29	1830	395	304	41
30	1830	395	303	46

MOTSA MTO Instance 5 – Symmetrical Setup Times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	2847	302	614	27
2	2847	302	609	41
3	2847	302	610	45
4	2994	565	615	29
5	2847	302	607	36
6	2847	427	613	44
7	2927	360	608	31
8	2886	302	605	28
9	2874	360	619	23
10	2998	554	611	30
11	2847	384	629	30
12	3168	524	622	29
13	2923	360	631	22
14	3057	740	644	25
15	2747	599	647	30
16	2998	554	611	30
17	2847	384	629	30
18	3168	524	622	29
19	2923	360	631	22
20	3057	740	644	25
21	2747	599	647	30
22	2886	302	609	30
23	2847	302	612	29
24	2847	302	613	28
25	2847	384	612	29
26	2847	427	618	32
27	2998	554	616	31
28	2994	565	600	28
29	2747	599	620	30
30	3057	740	601	28

MOTSA MTO Case Study Instance – Asymmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	1680	60	453	102
2	1680	60	453	103
3	1680	60	453	104
4	1390	420	454	63
5	1680	60	454	103
6	1680	60	451	101
7	1390	420	452	59
8	1390	420	457	61
9	1680	60	454	104
10	1680	60	452	100
11	1680	60	450	103
12	1390	420	454	63
13	1390	420	453	57
14	1680	60	450	101
15	1390	420	451	58
16	1680	60	453	104
17	1680	60	453	98
18	1680	60	453	100
19	1680	60	455	105
20	1680	60	452	102
21	1680	60	452	99
22	1680	60	451	101
23	1680	60	450	100
24	1390	420	458	59
25	1680	60	451	104
26	1680	60	452	101
27	1390	420	454	62
28	1680	60	453	103
29	1680	60	451	101
30	1680	60	452	99

MOTSA MTO Instance 1 – Asymmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	906	0	302	40
2	906	0	305	30
3	906	0	305	50
4	906	0	301	46
5	1140	0	305	35
6	1006	0	304	32
7	906	0	305	45
8	1140	0	305	30
9	906	0	304	41
10	906	0	305	40
11	1066	0	306	37
12	906	0	304	44
13	1140	0	303	32
14	906	0	303	29
15	906	0	302	51
16	906	0	304	40
17	906	0	305	30
18	906	0	302	50
19	906	0	301	46
20	1140	0	302	35
21	1006	0	301	32
22	906	0	303	45
23	1140	0	303	30
24	906	0	301	41
25	906	0	304	40
26	1066	0	306	37
27	906	0	306	44
28	1140	0	301	32
29	906	0	303	29
30	906	0	305	51

MOTSA MTO Instance 2 – Asymmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	1340	40	301	137
2	1340	40	302	138
3	1340	40	301	137
4	1340	40	301	138
5	1340	40	302	137
6	1340	40	302	132
7	1340	40	301	130
8	1340	40	301	132
9	1340	40	302	130
10	1340	40	302	123
11	1340	40	300	124
12	1340	40	302	123
13	1340	40	301	121
14	1340	40	300	128
15	1340	40	300	128
16	1340	40	301	137
17	1340	40	302	138
18	1340	40	301	137
19	1340	40	301	138
20	1340	40	302	137
21	1340	40	302	132
22	1340	40	301	130
23	1340	40	301	132
24	1340	40	302	130
25	1340	40	302	123
26	1340	40	300	124
27	1340	40	302	123
28	1340	40	301	121
29	1340	40	300	128
30	1340	40	300	128

MOTSA MTO Instance 3 – Asymmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	1000	0	302	67
2	1000	0	302	69
3	740	0	304	138
4	1000	0	305	65
5	810	0	312	110
6	750	0	301	123
7	1000	0	303	71
8	1000	0	304	69
9	730	0	302	158
10	1000	0	302	68
11	1000	0	304	71
12	710	0	301	143
13	710	0	305	101
14	1000	0	302	63
15	1000	0	303	65
16	1000	0	301	96
17	710	0	302	99
18	730	0	305	93
19	1000	0	301	87
20	1000	0	304	88
21	810	0	300	87
22	1000	0	300	100
23	710	0	305	96
24	1000	0	302	93
25	1000	0	302	95
26	740	0	300	87
27	710	0	303	95
28	750	0	303	93
29	1000	0	302	95
30	1000	0	302	82

MOTSA MTO Instance 4 – Asymmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	1870	420	304	40
2	1900	420	304	45
3	1820	420	315	49
4	1920	420	306	40
5	1820	420	305	44
6	1820	420	302	44
7	1900	420	305	51
8	1880	420	305	45
9	1820	420	308	43
10	1820	420	317	46
11	1820	420	309	50
12	1820	420	304	45
13	1820	420	309	47
14	1870	420	301	41
15	1820	420	308	44
16	1820	420	306	40
17	1820	420	302	40
18	1820	420	300	41
19	1820	420	301	48
20	1870	420	302	46
21	1820	420	305	46
22	1870	420	304	45
23	1900	420	305	48
24	1820	420	300	41
25	1920	420	302	47
26	1820	420	302	40
27	1820	420	304	41
28	1900	420	300	45
29	1870	420	304	40
30	1820	420	301	43

MOTSA MTO Instance 5 – Asymmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	2944	0	605	36
2	2944	0	602	40
3	2944	0	603	33
4	2944	0	605	30
5	2944	0	602	50
6	2944	0	605	20
7	2944	0	602	35
8	2944	0	604	39
9	2944	0	603	30
10	2944	0	606	18
11	2944	0	607	46
12	2944	0	605	44
13	2944	0	605	26
14	2944	0	607	17
15	2944	0	605	20
16	2944	0	603	35
17	2944	0	607	41
18	2944	0	607	22
19	2944	0	604	56
20	2944	0	607	17
21	2944	0	602	40
22	2944	0	607	31
23	2944	0	606	23
24	2944	0	606	27
25	2944	0	606	25
26	2944	0	603	34
27	2944	0	606	55
28	2944	0	607	54
29	2944	0	604	36
30	2944	0	607	22

MOTSA MTS Case Study Instance – Symmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	1170.0	0	52	43
2	1170.0	0	59	49
3	1170.0	0	63	62
4	1170.0	0	55	49
5	1170.0	0	67	51
6	1170.0	0	60	52
7	1170.0	0	57	58
8	1170.0	0	58	47
9	1480.0	0	42	48
10	1170.0	0	62	53
11	1170.0	0	61	61
12	1170.0	0	46	54
13	1170.0	0	61	59
14	1170.0	0	51	45
15	1170.0	0	54	47
16	1170.0	0	57	59
17	1170.0	0	59	47
18	1170.0	0	55	45
19	1170.0	0	47	58
20	1170.0	0	64	54
21	1170.0	0	53	54
22	1170.0	0	58	51
23	1480.0	0	43	51
24	1170.0	0	62	62
25	1170.0	0	54	46
26	1170.0	0	56	53
27	1170.0	0	72	61
28	1480.0	0	48	62
29	1170.0	0	56	55
30	1170.0	0	58	56

MOTSA MTS Instance 1 – Symmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	533	0	18	100
2	533	0	19	100
3	532	0	17	100
4	534	0	19	100
5	532	0	19	100
6	532	0	18	100
7	532	0	17	100
8	532	0	19	100
9	533	0	20	100
10	534	0	19	100
11	534	0	18	100
12	534	0	19	100
13	540	0	18	100
14	532	0	18	100
15	532	0	18	100
16	533	0	18	100
17	532	0	17	100
18	532	0	19	100
19	534	0	18	100
20	534	0	19	100
21	532	0	20	100
22	532	0	19	100
23	532	0	18	100
24	534	0	19	100
25	534	0	19	100
26	532	0	19	100
27	532	0	19	100
28	532	0	17	100
29	532	0	16	100
30	532	0	18	100

MOTSA MTS Instance 2 – Symmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	880	0	44	100
2	880	0	42	100
3	880	0	42	100
4	880	0	42	100
5	880	0	41	100
6	880	0	42	100
7	880	0	40	100
8	880	0	41	100
9	880	0	44	100
10	880	0	42	100
11	880	0	43	100
12	880	0	42	100
13	880	0	40	100
14	880	0	42	100
15	880	0	43	100
16	880	0	42	100
17	880	0	42	100
18	880	0	42	100
19	880	0	41	100
20	880	0	42	100
21	880	0	43	100
22	880	0	43	100
23	880	0	42	100
24	880	0	45	100
25	880	0	42	100
26	880	0	46	100
27	880	0	46	100
28	880	0	45	100
29	880	0	46	100
30	880	0	44	100

MOTSA MTS Instance 3 – Symmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	700	0	55	100
2	700	0	57	100
3	700	0	54	100
4	700	0	57	100
5	700	0	54	100
6	700	0	67	100
7	700	0	69	100
8	700	0	68	100
9	700	0	60	100
10	700	0	53	100
11	700	0	52	100
12	700	0	50	100
13	700	0	58	100
14	700	0	54	100
15	700	0	53	100
16	700	0	56	100
17	700	0	54	100
18	700	0	53	100
19	700	0	58	100
20	700	0	55	100
21	700	0	57	100
22	700	0	52	100
23	700	0	50	100
24	700	0	55	100
25	700	0	54	100
26	700	0	56	100
27	700	0	56	100
28	700	0	55	100
29	700	0	41	100
30	800	0	37	100

MOTSA MTS Instance 4 – Symmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	1720	0	66	100
2	1720	0	58	100
3	1720	0	56	100
4	1780	0	56	100
5	1710	0	57	100
6	1720	0	56	100
7	1710	0	57	100
8	1710	0	57	100
9	1720	0	63	100
10	1710	0	57	100
11	1760	0	63	100
12	1720	0	60	100
13	1710	0	59	100
14	1720	0	53	100
15	1760	0	56	100
16	1710	0	56	100
17	1720	0	56	100
18	1710	0	54	100
19	1720	0	62	100
20	1710	0	57	100
21	1710	0	54	100
22	1720	0	62	100
23	1710	0	58	100
24	1720	0	58	100
25	1720	0	53	100
26	1720	0	55	100
27	1710	0	57	100
28	1720	0	60	100
29	1710	0	59	100
30	1760	0	59	100

MOTSA MTS Instance 5 – Symmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	2344	0	170	100
2	2344	0	222	100
3	2344	0	271	100
4	2344	0	305	90
5	2344	0	237	100
6	2344	0	282	100
7	2344	0	182	100
8	2344	0	305	93
9	2344	0	306	91
10	2344	0	302	89
11	2344	0	306	90
12	2344	0	306	87
13	2344	0	280	100
14	2344	0	301	84
15	2344	0	306	82
16	2344	0	303	93
17	2344	0	291	100
18	2344	0	303	97
19	2344	0	282	100
20	2344	0	302	90
21	2352	0	124	100
22	2344	0	305	100
23	2344	0	306	94
24	2344	0	304	95
25	2344	0	304	80
26	2344	0	300	95
27	2344	0	302	85
28	2344	0	303	88
29	2344	0	304	96
30	2344	0	240	100

MOTSA MTS Case Study Instance – Asymmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	1132	1123	0	47
2	1132	1150	0	50
3	1132	1120	0	57
4	1132	1133	0	55
5	1132	1180	0	55
6	1132	1123	0	49
7	1132	1123	0	57
8	1132	1120	0	50
9	1132	1120	0	51
10	1132	1120	0	54
11	1132	1120	0	61
12	1132	1133	0	62
13	1132	1123	0	50
14	1132	1150	0	52
15	1132	1120	0	51
16	1132	1120	0	49
17	1132	1120	0	42
18	1132	1133	0	52
19	1132	1120	0	51
20	1132	1120	0	50
21	1132	1180	0	52
22	1132	1120	0	53
23	1132	1133	0	56
24	1132	1120	0	55
25	1132	1133	0	55
26	1132	1120	0	50
27	1132	1120	0	46
28	1132	1120	0	55
29	1132	1190	0	51
30	1132	1150	0	49

MOTSA MTS Instance 1 – Asymmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	510	0	15	100
2	510	0	14	100
3	524	0	15	100
4	528	0	14	100
5	510	0	17	100
6	528	0	14	100
7	528	0	14	100
8	520	0	15	100
9	510	0	15	100
10	510	0	16	100
11	510	0	16	100
12	520	0	14	100
13	528	0	13	100
14	510	0	16	100
15	524	0	14	100
16	520	0	15	100
17	520	0	14	100
18	520	0	15	100
19	510	0	16	100
20	520	0	14	100
21	510	0	14	100
22	524	0	16	100
23	524	0	16	100
24	524	0	16	100
25	510	0	16	100
26	510	0	15	100
27	510	0	16	100
28	510	0	14	100
29	510	0	15	100
30	510	0	15	100

MOTSA MTS Instance 2 – Asymmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	970	0	40	100
2	970	0	40	100
3	970	0	40	100
4	970	0	39	100
5	970	0	40	100
6	970	0	37	100
7	970	0	40	100
8	970	0	40	100
9	970	0	39	100
10	970	0	40	100
11	970	0	39	100
12	970	0	39	100
13	970	0	40	100
14	970	0	40	100
15	970	0	40	100
16	970	0	39	100
17	970	0	39	100
18	970	0	38	100
19	970	0	40	100
20	970	0	39	100
21	970	0	40	100
22	970	0	39	100
23	970	0	36	100
24	970	0	40	100
25	970	0	40	100
26	970	0	40	100
27	970	0	40	100
28	970	0	40	100
29	970	0	39	100
30	970	0	40	100

MOTSA MTS Instance 3 – Asymmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	610	0	25	100
2	710	0	25	100
3	600	0	24	100
4	600	0	24	100
5	600	0	25	100
6	600	0	24	100
7	600	0	24	100
8	710	0	25	100
9	635	0	22	100
10	610	0	24	100
11	610	0	23	100
12	600	0	23	100
13	635	0	25	100
14	600	0	24	100
15	635	0	22	100
16	610	0	22	100
17	600	0	25	100
18	600	0	24	100
19	600	0	24	100
20	600	0	24	100
21	600	0	26	100
22	600	0	23	100
23	600	0	23	100
24	610	0	23	100
25	610	0	22	100
26	600	0	26	100
27	610	0	23	100
28	600	0	23	100
29	635	0	23	100
30	600	0	24	100

MOTSA MTS Instance 4 – Asymmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	1450	0	54	100
2	1450	0	56	100
3	1450	0	54	100
4	1450	0	55	100
5	1450	0	54	100
6	1450	0	60	100
7	1450	0	58	100
8	1450	0	52	100
9	1450	0	56	100
10	1450	0	55	100
11	1450	0	56	100
12	1450	0	58	100
13	1450	0	55	100
14	1450	0	58	100
15	1450	0	58	100
16	1450	0	56	100
17	1450	0	58	100
18	1450	0	46	100
19	1450	0	54	100
20	1450	0	58	100
21	1450	0	57	100
22	1450	0	56	100
23	1450	0	50	100
24	1450	0	57	100
25	1450	0	47	100
26	1450	0	53	100
27	1450	0	50	100
28	1450	0	54	100
29	1450	0	51	100
30	1450	0	57	100

MOTSA MTS Instance 5 – Asymmetrical Setup times

Execution Number	Best Makespan	Best Tardiness	Run Time (seconds)	Iterations
1	2634	0	301	81
2	2634	0	302	78
3	2634	0	301	85
4	2634	0	307	77
5	2634	0	301	77
6	2634	0	302	81
7	2634	0	300	82
8	2634	0	304	71
9	2634	0	304	68
10	2634	0	305	74
11	2634	0	301	79
12	2634	0	306	73
13	2634	0	302	77
14	2634	0	300	77
15	2634	0	300	76
16	2634	0	303	78
17	2634	0	301	75
18	2634	0	301	83
19	2634	0	306	79
20	2634	0	304	78
21	2634	0	304	83
22	2634	0	300	82
23	2634	0	303	78
24	2634	0	304	78
25	2634	0	303	80
26	2634	0	305	80
27	2634	0	301	81
28	2634	0	300	72
29	2634	0	302	82
30	2634	0	304	77