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

João Côrte-Real Bivar Sacramento
Department of Engineering and Management, Instituto Superior Técnico
joao.c.sacramento@tecnico.ulisboa.pt

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

1. Introduction

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 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. The dissertation the present rocument refers to focuses on Logoplaste, a Portuguese company which produces plastic containers through plastic injection molding. Logoplaste Santa Iria (LSI), the factory that this paper is concerned with, has multiple machines available to manufacture their products. Different machines have different cycle times and 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.

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.

This research's objectives are to develop a model which will help LSI in the scheduling of FIMA orders taking into account two objective functions: total tardiness and makespan. Also, the model developed must investigate both production strategies, Make to Stock (MTS) and Make to Order (MTO).

The research's structure is as follows. Section 2 presents the problem description. Section 3 shows the literature review of meta-heuristics studied for this problem. Section 4 present's the algorithm's characterization. Section 5 describes the parameter tuning, test instances used, and an analysis of results obtained. Finally, conclusions and future work are presented in Section 6.

2. Problem Description

Logoplaste is a Portuguese company which specializes in plastic containers made of polyethylene terephthalate (PET), polypropylene (PP) and high-density polyethylene (HDPE).

This paper 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.

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.

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 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 references implies setup times. This is the time required to shift from one product to another. There are three types of components, so three types of setup (molds, dyes and labels) are identified. The setups times are symmetrical, meaning that changing from product 1 to product 2, or vice-versa, originates the same setup time.

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

In the JSSP, a set J of jobs $J = \{J_1, \ldots, J_j, \ldots, J_N\}$ jobs is to be processed on a set M of machines $M = \{M_1, \ldots, M_k, \ldots, M_M\}$. A job is an ordered subset of a set O of operations $O = \{O_1, \ldots, O_i, \ldots, O_n\}$. 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.

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

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

by determining the operation sequence in each machine. Solutions can be represented, for example, in a Gantt Chart, such as the one in

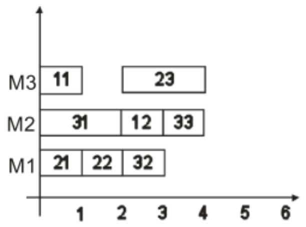

Figure 1 – Gantt chart of a solution to a 3 jobs, 3 machines problem

Figure 1. The x axis represents time, the y axis represents machines and operations O_{ij} represent the j^{th} operation of job J_i.

3. Literature Review

This section aims 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 section is 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 highlighted; finally, the reviewed literature is summarized.

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:

$$(1) \quad f(s^*) = \min f(s), s \in X$$

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 summarizes the three dimensions considered and offers a brief explanation of each.

Table 1 - 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

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. A distinction between flowshop and jobshop is represented in Figure 2.

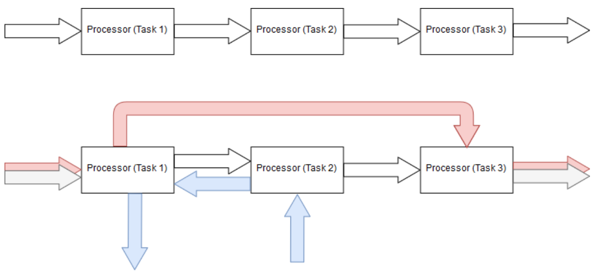

Figure 2 - Flowshop (top) vs Jobshop (bottom)

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.

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 are required to complete a given schedule. Setup times can either be Sequence Dependent or Sequence Independent. Sequence Dependent Setup Times (SDST) 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. The review of literature focuses on this problem type as much as possible, though the classical JSSP as well as some implementations where SDST is not considered are also studied. Both Mono and Multi-Objective implementations are explored, in order to understand how each objective function is tackled as well as how Meta-Heuristics can be adapted to fit more than one objective function.

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.

Exact methods were the first approach used to solve the job shop scheduling problem, and the most common techniques used are branch and bound, Mixed-Integer Programming and Constraint Programming (Graves, 1981).

Both (Bowman, 1959) and (Wagner, 1959) developed Mixed-Integer Linear Programming (MILP) 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.

Given the limitations of exact approaches, 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 based on of two types.

Local search meta-heuristics 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.

Tabu Search

This research proposes to develop a Tabu Search (TS). The 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.

First, an overview of the Mono-Objective problem is presented.

(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. (Brandimarte, 1993) developed a hierarchical TS algorithm for both minimizing makespan and weighted tardiness in the FJSSP using simple neighborhood structures.

(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.

(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. (C. Scrich, V. Armentano, 2004) proposed two TS approaches for tardiness minimization in the FJSSP, namely a hierarchical and a multi-start procedure. 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. (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. In (Shen, Dauzère-Pérès and Neufeld, 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).

Finally, concerning the Multi-Objective FJSSP, the following papers used the TS as an approach.

(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. These criteria were weighted and the objective function for the algorithm developed is a weighted sum of the three criteria. (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.

Literature Review Conclusions

The TS showed to be a promising approach to the FJSSP, even when considering the SDST extension of the problem. 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.

4. Algorithm Characterization

This section focuses on describing the algorithm and its mechanisms. First, 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.
advantages and disadvantages.

Before the procedures are described, an overview of the problem variables 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 $s_{i0} = 0$.

To refer that two operations v and w are processed consecutively on the same machine, one should use (u, v). 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:

$$(2)\quad \begin{cases} T_n = d_n - C_n, \ d_n \leq C_n \\ 0, \ otherwise \end{cases}$$

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

$$(3)\quad T_{total} = \sum_{n=1}^{m} T_n$$

The makespan C_{max} is given by:

$$(4)\quad C_{max} = \max C_n$$

The MOTSA was implemented in Python 3.8 running on a Windows 10 Operating System.

The MOTSA can be applied to both MTS and MTO strategies.

These production strategies dictate how FIMA orders are satisfied: in the MTO strategy, production orders are taken directly from the set of FIMA orders received; in the MTS strategy, inventory's initial, minimum and maxiumum stock levels are considered.

To generate the production orders in the MTS strategy, orders are iterated over by order of ascending due date, i.e. by use of an Earliest Due Date dispatching rule. Each order is fulfilled from the inventory when possible, only calculating production requirements if the available inventory is insufficient. If inventory is insufficient, the present FIMA order translates into a production order for the factory. After allocating FIMA orders to inventory or a specific production order, the final inventory levels are calculated and production orders generated with enough quantities to replenish the inventory levels back to the maximum. This production strategy originates smaller sets of orders, making the problem easier to solve. Also, since orders can be fulfilled from the inventory, meeting delivery deadlines is much easier, thus optimal tardiness is expected to be achieved with more ease than in the MTO case.

Multi-Objective Tabu Search Algorithm (MOTSA)

MOTSA's main components are described below.

Problem Initialization

First and foremost, to extract all of the information required, a data set consisting of processing times, setup times and orders is needed.

Processing times of all products in each machine, also indicating which machines are eligible to manufacture each product.

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.

Orders received from FIMA, namely products, quantities and due dates.

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.

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.

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 appropriate for a good exploration of the solution space. 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.

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:

- **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'.

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.

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.

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.

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.

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.

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.

During an execution of the algorithm, solutions are checked for their Tabu Status when applying to be selected as the new seed solution. Tabu solutions cannot be accepted as new solutions in each iteration, unless they are the only improving solution in the neighborhood.

Diversification Strategy

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 selects 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..

Finally, the general framework of the algorithm is presented. 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 any stop criteron is reached, the algorithm stops and returns the best solution found. Figure 3 describes the main idea of the general framework with a macroscopic view of its components.

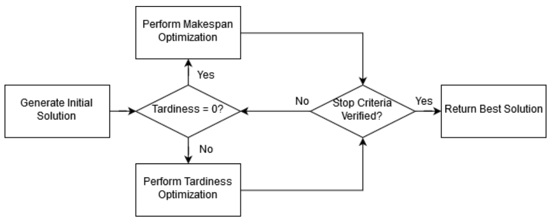
Perform Makespan Optimization
Yes
Generate Initial Solution
Tardiness = 0?
No
Stop Criteria Verified?
Yes
Return Best Solution
No
Perform Tardiness Optimization

Figure 3 - Flowchart of the MOTSA framework

5. Model Tuning and Results Analysis

In this section, the instances created for testing the model built are described. Sensitivity analysis for important parameters of the algorithm is shown. Finally, results of the application of the MOTSA on all instances are presented, considering both the MTS and MTO production strategies.

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.

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.

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. Results showed that a neighborhood size of 3 is ideal for the case study instance.

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 optimal size for the tabu list was shown to be 20. Therefore, each of the Tabu Lists has a maximum size of 20 Tabu elements.

Table 2 - Tuned Model Parameters

Maximum CPU Time	600
Maximum Number of Iterations	200
Neighborhood Size	3
Maximum Non-Improving Iterations	3
Assignment Tabu List Size	20
Sequencing Tabu List Size	20

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. 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 3 summarizes relevant characteristics of the instances created.

Table 3 - Test instances information

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 of the case study.

In total, 6 problem instances are considered in both MTO and MTS production strategies and also with 2 different sets of setup times, making a total of 24 sets of executions. Each of the problem instances above is run 30 times to ensure a statistically significant set of results. See Tables 4, 5, 6 and 7 for a summary of results on MOTSA's MTO Symmetrical and Asymmetrical instances, and MOTSA's MTS Symmetrical and Asymmetrical instances, respectively.

Table 4 - MTO results with Symmetrical Setups

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

Table 5 - MTO results with Asymmetrical Setups

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

Table 6 - MTS results with Symmetrical Setups

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

Table 7 - MTS results with Symmetrical Setups

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

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 much lower than in the MTO case. Nevertheless, the MTO production strategy proved to be effective for the case study instance, despite showing difficulties in the more difficult test instances.

The influence of the asymmetrical setup times is difficult to assess, since it affects each problem instance differently in the MTO and MTS case. For instance, the Case Study presented better results when considering the Asymmetrical setup times in the MTO strategy, while the opposite happened in the MTS strategy.

For a better understanding of each execution and how solution quality evolves over time in a run of the model, Figure 4 shows the evolution of both objective functions over an execution. The data was taken from Execution Number 3 of the neighborhood size sensitivity analysis. From the graph, it is clear that the first few iterations of the algorithm have a lot of room for improvement and, therefore, objective function values vary more in the beginning, whereas they tend to stabilize over the course of an execution. Is this case, the best solution was found quickly and for the following iterations, many more solutions were visited; however, no further improvements were found.

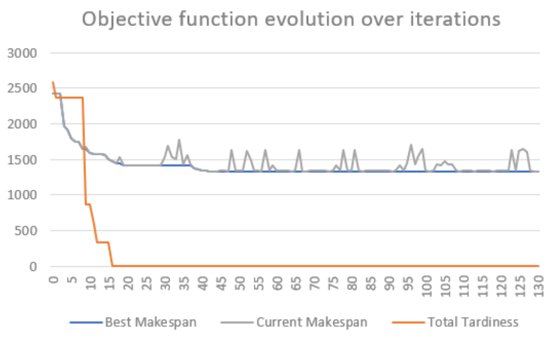

Figure 4 - Objective functions evolution across iterations

Furthermore, for a more visual representation of the model's consistency, a graph showing the results of 30 executions for the case study instance is presented in Figure 5. The executions the graph refers to were made for the MTO production strategy and considering symmetrical setup times. The left axis represents the makespan, while the right-hand axis represents the deviation of each execution's best makespan from the average makespan, as a percentage of the average makespan. This deviation never reaches an absolute value greater than 5%. Also, the 5% value was even achieved for an makespan lower than the average.

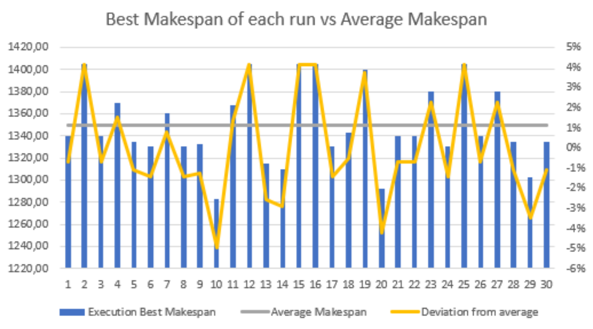

Figure 5 - Comparison of each execution vs average results

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. 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. 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 literature review found that the Mono-Objective Tabu Search had a good performance for this problem, especially considering the makespan. However, applications of a Multi-Objective Tabu Search were scarce. Furthermore, no papers were found implementing a Multi-Objective Tabu Search using the lexicographical method.

To address this gap, the algorithm developed in this paper was a Multi-Objective Tabu Search using the lexicographical method optimizing Total Tardiness first and then the Makespan.

Overall, the model developed in this paper 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. Some future work is identified:

First, while the neighbor functions provide an excellent support for a good exploration of the search space, the neighborhood generation procedure could be faster. Also, the tabu lists implemented are somewhat limitative in the sense that a fixed tabu list size does not adapt to the search's peculiarities. 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.

Bibliograpgy

Abdelmaguid, T. F. (2015) 'A neighborhood search function for flexible job shop scheduling with separable sequence-dependent setup times', *Applied Mathematics and Computation*. Elsevier Ltd., 260, pp. 188–203. doi: 10.1016/j.amc.2015.03.059.

Bowman, E. H. (1959) 'The Schedule-Sequencing Problem', *Operations Research, 7(5), 621–624.*, 7(5), pp. 621–624. doi: https://doi.org/10.1287/opre.7.5.621.

Brandimarte, P. (1993) 'Routing and scheduling in a flexible job shop by tabu search', *Annals of Operations Research*. doi: 10.1007/BF02023073.

C. Scrich, V. Armentano, M. L. (2004) 'Tardiness minimization in a ¯ exible job shop : A tabu search approach', *Journal of Intelligent Manufacturing*, 15, pp. 103–115. doi: https://link.springer.com/article/10.1023/B:JIMS.0000010078.30713.e9.

Dauzère-pérès, S. and Paulli, J. (1997) 'An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search', *Annals of Operations Research*, 70, pp. 281–306.

Dell'Amico, M. and Trubian, M. (1993) 'Applying tabu search to the job-shop scheduling problem', *Annals of Operations Research*. doi: 10.1007/BF02023076.

Garey, M. R., Johnson, D. S. and Sethi, R. (1976) 'The Complexity of Flowshop and Jobshop Scheduling', *Mathematics of Operations Research*, 1(2), pp. 117–129. doi: 10.1287/moor.1.2.117.

Glover, F. (1998) 'Tabu Search !', 3, pp. 621–757.

Graves, S. C. (1981) 'A Review of Production Scheduling', *Operations Research Publication*, 29(4), pp. 646–675.

Jia, S. and Hu, Z. (2014) 'Computers & Operations Research Path-relinking Tabu search for the multi-objective fl exible job shop scheduling problem', *Computers and Operation Research*. Elsevier, 47, pp. 11–26. doi: 10.1016/j.cor.2014.01.010.

Li, J. *et al.* (2010) 'An effective hybrid tabu search algorithm for multi-objective flexible job- shop scheduling problems', *Computers & Industrial Engineering*, 59(4), pp. 647–662. doi: 10.1016/j.cie.2010.07.014.

Manne, A. S. (1960) 'On the Job-Shop Scheduling Problem', *Operations Research Publication*, 8(2). doi: https://doi.org/10.1287/opre.8.2.219.

Mastrolilli, M. and Gambardella, L. M. (2000) 'Effective neighbourhood functions for the flexible job shop problem', *Journal of Scheduling*, 3(1), pp. 3–20. doi: 10.1002/(SICI)1099-1425(200001/02)3:1<3::AID-JOS32>3.0.CO;2-Y.

Saidi-Mehrabad, M. and Fattahi, P. (2007) 'Flexible job shop scheduling with tabu search algorithms', *International Journal of Advanced Manufacturing Technology*. doi: 10.1007/s00170-005-0375-4.

Shen, L., Dauzère-Pérès, S. and Neufeld, J. S. (2018) 'Solving the flexible job shop scheduling problem with sequence-dependent setup times', *European Journal of Operational Research*. Elsevier B.V., 265(2), pp. 503–516. doi: 10.1016/j.ejor.2017.08.021.

Wagner, H. M. (1959) 'An Integer Linear-Programming Model For Machine Scheduling', *Naval Research Logistics*, pp. 131–140. doi: https://doi.org/10.1002/nav.3800060205.