Master thesis
Industrial Engineering Master Degree

Flowshop Scheduling Problem with TCIT minimization

Author: Paula Sánchez de los Reyes

Tutor: Paz Pérez González

Dept. of Industrial Organisation and Business Management
School of Engineering
University of Seville

Sevilla, 2023

Master thesis
Industrial Engineering Master Degree

Flowshop Scheduling Problem with TCIT minimization

Author:
Paula Sánchez de los Reyes

Tutor:
Paz Pérez González

Dept. of Industrial Organisation and Business Management
School of Engineering
University of Seville

Seville, 2023

Acknowledgements

To my family, especially my parents, my sister and my couple, thank you for the unconditional support you have always given me. To my tutor, for accompanying me and giving me all her help.

Sevilla, 2023

Resumen

El Trabajo Fin de Máster presentado aborda la minimización del tiempo total de espera de las máquinas (Total Core Idle time) dentro de un problema determinista de programación del taller de flujo permutado. El interés de la minimización de este objetivo viene estrechamente ligado con la minimización del consumo energético del problema.

En este Trabajo Fin de Máster se presenta la adaptación de los métodos más eficientes desarrollados para objetivos clásicos (makespan y total completion time o total flowtime), para la minimización del Total Core Idle time.

En cuanto a las heurísticas, NEH, NEH$_M$, LR-NEH y PFH-NEH son las heuristicas adaptadas a nuestro problema. Por otro lado, IG, IGALL y VBIH son las metaheurísticas adaptadas. Además, se aplica una búsqueda local basada en una primera mejora con intercambios generales en todas las heurísticas y metaheurísticas que la contienen.

El primer objetivo del documento es la calibración de cada uno de los métodos adaptados, eligiendo así cada uno de los parámetros en cuestión para cada una de las heuristicas y metaheuristicas. Para ello se utilizan las instancias de Taillard.

A continuación, una vez calibrados los métodos, se comparan las heurísticas y metaheurísticas utilizando las instancias de Vallada. Finalmente, la conclusión que se obtiene de dicho estudio es la elección de la heuristica construcitva NEH$_M$ como las más eficientes dentro de las estudiadas y la VBIH dentro de las metaheuristicas.

Por último, cabe señalar que este documento se ha redactado en Latex, el código del método se ha escrito en C, el tratamiento de los datos en Excel y el análisis estadístico en SPSS.

Abstract

This document addresses deterministic permutation flowshop scheduling problem with a new objective function, the total core idle time. The interest of this objective is related to reduce the energy consumption of the system, taking into account the machine energy consumption.

In this master thesis, we present the adaptation of the most efficient methods presented for classical objectives (makespan and total completion time or total flowtime), for the minimization of the Total Core Idle time.

About the heuristics, NEH, NEH$_M$, LR-NEH and PFH-NEH are adapted for our problem. On the other hand, IG, IGALL and VBIH are the metaheuristics adapted. In addition, a local search based on a first improvement with general swaps is applied in all heuristics and metaheuristics that contain this type of operators.

The first objective of the document is to calibrate the parameters of each method, selecting the best one. For this purpose the Taillard benchmark is used.

Then, once the methods are calibrated, the heuristics and metaheuristics are compared using the Vallada's benchmark. Finally, the most efficient method are the NEH$_M$ related to the heuristics and the VBIH about the metaheuristics.

Finally, it should be noted that this document has been written in Latex, the method code has been C, the processing of data in Excel and the statistical analysis in SPSS.

Contents

1 Introduction

1.1 Scheduling

In order to frame the problem developed in the master thesis, a brief introduction of the manufacturing scheduling should be done. Scheduling is understood by [16] as the assignment of the various resources of a company to the manufacturing of a range of products.

The first step to a good understanding is to contextualize the manufacturing scheduling. As [16] explained, production management is the process in which manufacturing decisions must be taken in order to ensure the delivery of goods with maximum quality, minimum cost, and minimum lead time. These decisions depend on the impact on the company and the scope among others. Depending on the timing and the actors involved in these decisions, there are different levels in which are classified. As [16] classifies, the first group is the long-term decisions, also called strategic decisions. The scope is to establish major decisions as the plants to build, the products, and the capacity among others. These decisions have a great impact on all areas of the company and are often reviewed . The next decisions mentioned by [16] are related to how these goods are used on a day-to-day basis. These are divided into two groups, medium-term decisions (tactical decisions), and short-term decisions (operating decisions). When it is necessary to plan the quantity of product to be produced for a long time and the information is unknown, the only option is by estimation. To facilitate this process it is usually to make the aggregated production plan. This contemplates the estimation of the group of products in weeks or months. According to these decisions it is possible to establish material quantity, capacity... As [16] mentioned, the last decisions are short-term decisions, establishing tasks to be done and the resources needed for doing them. These are daily and repetitive decisions where manufacturing scheduling is located.

The scheduling decisions made by the companies are different, but there are some common aspects that are needed by the majority. One of the common features is the importance of decisions due to the direct impact on delivery times. As

[16] explained, it is necessary to understand that scheduling does not it a problem, is a process that is conducted by people to make decisions. Sumarizing, scheduling manufacturing is the decision-making process consisting on assigning the set of operations/tasks required to manufacture a set of products to the existing resources in the shop floor, as well as the time periods to initiate these operations/tasks.

Due to the complexity of the scenary, a mathematical model must be used to make decisions. For this purpose, it is necessary to develop scheduling models, methods to solve scheduling problems, and tools and systems to facilitate this. Then, an abstraction of reality to the solved model is assumed. For example, we will assume that it is possible to identify units of resources that can perform these operations/tasks and we will refer to them simply as machines.

To the best understanding of the method developed to the manufacturing scheduling, as [16] mentioned, some concepts must be introduced. The first one is the the definition of the unit to be executed previously mentioned. It is a task, that can be a product, part of a product, or others and is named as job. Some characteristics of these jobs can be whether jobs have the order of operations fixed in advance, the possibility of preemption in operations, or if a precise machine must do a job or can do the job at a different speed than the others. These characteristics that constrain the model cause certain schedules to be unfeasible and therefore, the aim of scheduling is to find at least one feasible schedule. Then, among the feasible schedules, a way of finding the best on must be defined by selecting a criterion or various criteria to compare them. About the criteria, the most common is the non-decreasing function of the completion times of the jobs, as makespan or total flow time. It is a brief intruduction which will be explained with more details throughout the document.

Finally, about the representation of this model, is usually represented by a Gantt diagram. This Gantt diagram can be machine-oriented or job-oriented. While in both cases time is represented on the x-axis, machines (first one) or jobs are represented on the vertical axis. Either one of the two represents the schedule of jobs in each one of the machines. For a better understanding, we can look at the figures 1.1.

1.2 Notation

As previously introduced, scheduling is a decision process to solve a real problem by a model. Through a model we achieve focus on a certain problem of reality or a simplification of reality that can be solved.
There are several types of models:

- Mathematical model.

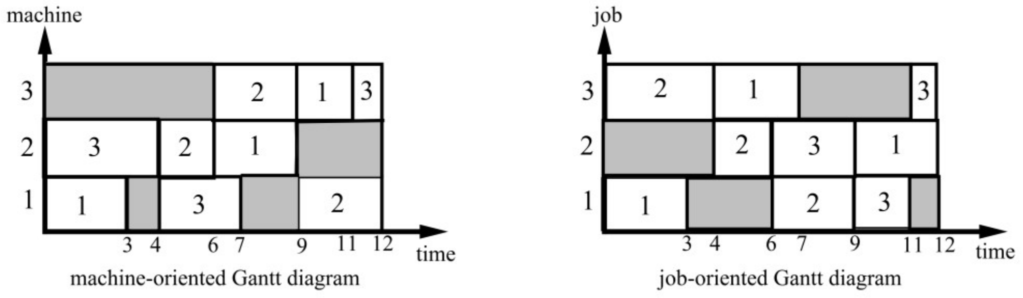

Figure 1.1 Machine-oriented and job-oriented Gantt diagrams [16].

- Graphical models.
- Statistical models.
- Simulation models.
- Algorithmic models.

A scheduling model is a formal abstraction of a (manufacturing) scheduling decision-making problem which is described by considering the system of tasks/operations, the processing constraints and the criteria.

It is assumed that a scheduling model contains several numbers of jobs and machines. More specifically we assume that there are N jobs and M machines ($N = 1,2,...,n$ $M = 1,2,...,m$).

These jobs and machines are independent and all the following data is deterministic:

- Task or Operation (O_{ij}): The number of tasks or operations for each job that must be scheduled. According to the scheduling, there are a star time (SO_{ij}) and a finish time (EO_{ij}) .
- Processing route(R_j): This is the predefined route that each of the jobs must follow, in other words, the machines order for each job.
- Processing time(p_{ij}): The time that needs each machines to process each job. $p_{ij} \leq E_{Oij}$-SO_{ij}.
- Ready dates (r_j): The instant of time in which the job is available to schedule. $\forall\ j \in N, i \in M\ SO_{ij} \geq r_j$. If is necessary to use a ready date r_{ij} $\forall\ j \in N, i \in M$, $SO_{ij} \geq r_{ij}$.
- Due dates (d_j): The instant of time in which ideally must be processed.
- Mandatory due dates or deadlines ($\overline{d_j}$): The instant of time it is mandatory to be processed.

- Due windows s $[[d_j^-, d_j^+]]$: A time interval where a job should be delivered.
- Costs, priorities, importance, or weights (w_j): Is used by models who need to prioritize or give importance to a job.
- Release dates for machines (rm_i): The time in which the machine is available. $\forall\, i \in M,\, j \in N,\, SO_{ij} \geq rm_i$

The triplet $\alpha|\beta|\gamma$ represents the type shop floor, the restrictions, and the criteria. The α describes the machine environment, which for the case of study is flowshop and is denoted as Fm. The β field represents the constraints. At last, γ field represents the objective to minimize. These three classifications will be explained deeply in the following sections.

1.3 Flowshop

The literature classifies the scheduling problems according the machine layout, and according to how the different job processing routes R_j are specified.

Flow Shop or flowshop is the layout where every job has a number of operations to be processed on a set of machines. For the best understanding, we can see the Figure 1.2. In this layout there are some assumptions:

- The machines are arranged in series, each one with a different purpose.
- Each job must visit each machine, in the same specific order.
- Each machine can only process a job at the same time. For this reason, there are several jobs waiting for the same machine.
- Each job can be processed by one machine at the same time.
- The transportation time between stages is considered null.
- The order of the jobs in each machine could be different. In many cases, for reasons of simplicity in scheduling it is assumed that the processing sequence is the same for all machines. In a permutation flow shop model all jobs must follow the same machine order (i.e. the first machine is the first one for every job). Solving the non-permutation problem is also more difficult as the possible schedules are $(n!)^m$ towards the n! possible sequences for the permutation problem. In order to get a visual idea of the difference, we can see the following figures 1.3 and 1.4

1.4 Problem description

Summarising the considered problem involves is the classical PFSP (permutation flowshop scheduling problem), which considers n jobs to be scheduled on m machines arranged in series, and all jobs will follow the same sequence

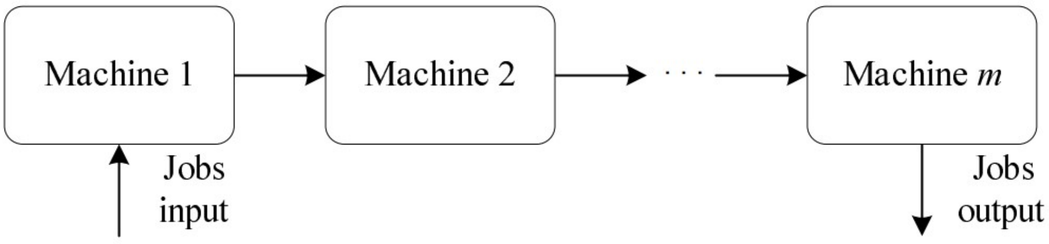

Figure 1.2 Flow layout of flowshop [16].

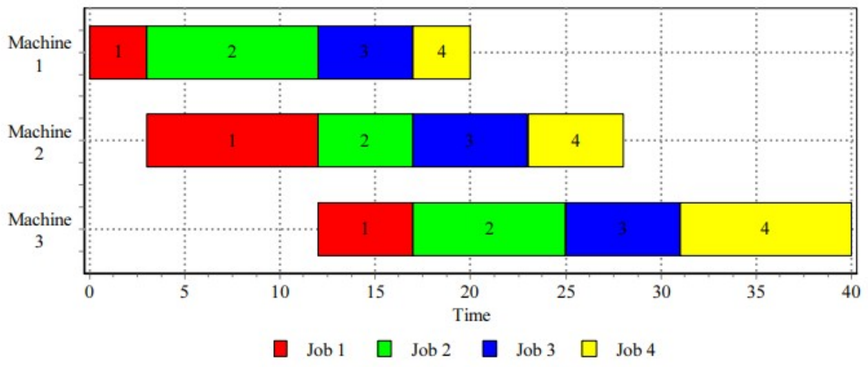

Figure 1.3 Flow layout of flowshop [16].

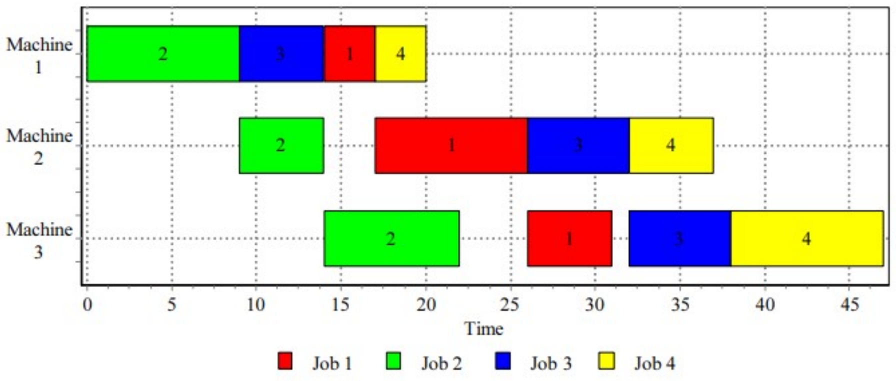

Figure 1.4 Flow layout of flowshop [16].

(permutation assumption). Jobs and machines are available at time zero. Preemption of jobs is not allowed. The objective function is to minimize the Total Core idle time (TCIT). The Total Core Idle Time is defined by [22] as the idle time between two consecutive jobs in a flowshop schedule (see Figure 1.5). It is demonstrated by [22], who shows that problem PFSP with TCIT is NP-hard in the strong sense. In this master thesis, the interest of this objective function is that, the core idle time is the time that the machine should be in standby status, consuming less energy than during the processing time of the jobs.

The core idle time minimization in the PFSP opens a new line of research in the literature about Energy Efficiency PFSP, due to the relation of this objective with the energy consumption of the system when the machines are idle (stand-by status).

As in the classical PFSP, the speed of the machine is not configurable, and the processing times are constant. Additionally, machines may be turned-on once the first job is going to be processed, and turned-off when the last job is finished, so during front and back idle times machines are off instead of the standby mode.

Before the definition of the objective function, the components of idle time must be presented:

- Front Idle Time of machine i, $FIT_i, 2 \leq i \leq m$: time before machine i processes the job in the first position.
- Core Idle Time of job in position k of machine i, $CIT_{i,[k]}, 2 \leq i \leq m, 2 \leq k \leq n$: time between the completion time of job in position k 1 and the starting time of job in position k in machine i.
- Back Idle Time of machine i, $BIT_i, 1 \leq i \leq m-1$: time after the machine i has finished the last job before the completion of the overall schedule.

Formally, the core idle time of machine *i* is defined as

$$CIT_i = \sum_{k=2}^{n} (C_{i,[k]} - p_{i[k]} - C_{i,[k-1]}) \tag{1.1}$$

Where, $p_{i[k]}$ is the given processing time of job scheduled in position *k* in machine *i*, and $C_{i,[k]}$ is the variable providing the completion time of job scheduled in position *k* in machine *i*.

Therefore, Total Core Idle Time is defined as

$$TCIT = \sum_{i=1}^{m} CIT_i \tag{1.2}$$

The relation between TCIT and makespan is deeply analysed by [22]. Taking into account the conclusions from that study, the optimal solution provided by makespan minimization is not the optimal solution for TCIT. However, the relation between both measures is that TCIT minimization provides better results for makespan than the opposite (makespan minimization is not efficient

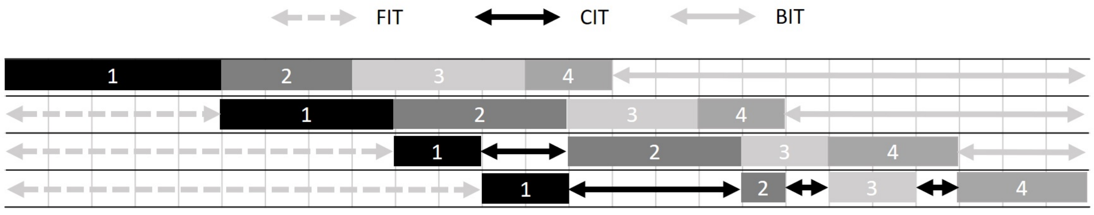

Figure 1.5 Front/Core/Back Idle Times (FIT/CIT/BIT) in a PFSP [5].

for TCIT). This means that specific approximate methods from the literature developed for makespan may not provide directly good solutions for TCIT, and specific methods for this new objective should be developed.

1.5 Objective of the master thesis

The deterministic permutation flowshop scheduling problem (PFSP) has been widely studied considering different (single) objective functions, usually makespan (see e.g. [6, 24, 2, 10, 12, 17]), total completion time (see e.g. [11, 9]), and due-date related objectives (see e.g. [32, 11, 3, 13]).

As, to the best of our knowledge, approximate methods have not been applied to the PFSP with TCIT minimization, in this master thesis, the main idea is to test methods proposed in the literature for related problems, in order to check their efficiency for this objective. Related problems can be PFSP with makespan and total completion time, due to the relation between these objectives and machine idle times [22]. We have adapted the following approximate methods in the literature developed for makespan to TCIT: the constructive heuristic NEH [25], and the metaheuristics IG ([30]), IGALL [7] and VBIH [34]. It has not been possible to include in this work the state-of-the-art method by [10], due to the high computational effort needed by this method in the adaptation, since the accelerations and critical path methods applied to the makespan are not adaptable to the TCIT with similar complexity to the total completion time, see e.g. [11]. Additionally, we have adapted one approximate methods developed for total completion time: the LR-NEH, since, according to [29], it is the best constructive heuristic with respect to the quality of the solution.

1.6 Document structure

As we have already seen, the document starts with an Introduction (Chapter 1). It is composed of an introduction of the main characteristics of Scheduling

(Section 1.1), contextualizing the notation used (Section 1.2), and following with a description of Flow shop layout (Section 1.3). The next section is the definition of the problem (Section 1.4), and finally, this section in which the document structure is summarized.

After this, a literature review is presented (Chapter 2). It is composed by a introduction of the same in which manufacturing scheduling en general terms is presented. After that's the literature review of the classical problems is developed. Finally, a more specific review of the considered problem is presented.

The next chapter details the construcitve heuristics (Chapter 3). More specifically the presented method are NEH, NEH$_M$, PFH-NEH and LR-NEH. The metaheuristics are presented in Chapter 4 . The metaheuristics are IG, IGALL and VBIH.

In Chapter 5, the computational results of these heuristics and metaheuristics are presented. The chapter starts with a description of the testbed used as set of instances (Section 6.1), following with the calibration of the heuristics (Section 6.2) and the comparison of the same (Section 6.3). The two following sections are the calibration and comparison of the metaheuristics (Section 6.3 and Section 6.4).
Finally, the last chapter of the document presents the conclusions (Chapter 6).

2 Literature Review

2.1 Introduction

Manufacturing scheduling has been widely studied in recent times due to its applicability to real cases. Can be found a large number of papers about this topic. More specifically flow shop scheduling has great importance within the scope. The most studied problems are permutation flowshop minimizing makespan ($Fm|prmu|C_{max}$) and permutation flowshop minimizing total flowtime ($Fm|prmu|\sum_{j=1}^{n} C_j$). For this reason, we can find a lot of heuristics and metaheuristics proposed for these two problems. Another problem studied in the literature and which is related to the objective of our problem is the permutation flowshop minimizing total energy consumption.

With regard to the problem addressed in the master thesis, the ($Fm|prmu|TCIT$), the scenario is totally different. Idle times-related objectives have been scarcely considered in the literature.

It is clear that the minimization of the makespan is directly related to the minimization of the idle times (see [20]). However, [22] analyze the relation between total core idle time (TCIT), defined as the idle times between jobs, avoiding including the front idle time generated by the first job in a given schedule, and the makespan. Taking into account the conclusions from that study, the optimal solution provided by makespan minimization is not the optimal solution for TCIT. However, the relation between both measures is that TCIT minimization provides better results for makespan than the opposite (makespan minimization is not efficient for TCIT).

2.2 $Fm|prmu|C_{max}$ and $Fm|prm|\sum C_j$

In this section, we focus on some of the main methods developed for the classical objectives of makespan and total flow time.

[25] developed the most influential algorithm in the literature for the permutation flowshop problem with makespan minimization i.e. $Fm|prmu|C_{max}$, the NEH. This has marked the entire scheduling investigation. Many papers have been based on improving the proposed algorithm. It should be noted that one of the most important algorithms of the master thesis is the NEH. It was [33] was improved the complexity of the NEH algorithm, reducing it from O(n^3m) to O(n2m). [15]) studied the implementation of different initial sequences for the NEH. It showed that the best initial sequence is the one used in the classical NEH approach (i.e. no increasing order of the sum of the processing).

Another heuristic studied in this master thesis is the NEH$_M$ proposed by [27]. It is one of the clear examples of the influence of the NEH in the rest of the heuristics proposed. This method is applied to a hybrid flow shop (multiple parallel machines per stage) minimizing the Total Energy Consumption (TCE).

Focused on the total flow time minimization, the LR-NEH proposed by [29] is implemented. As is noted in this paper is a good trade-off between CPU time and quality. It is a constructive heuristic based on the NEH and the LR proposed by [21]. Following the total flow time minimization, the PFH-NEH proposed by [27] is implemented. As the LR-NEH, it is based on the combination of the PF [23] and the NEH already mentioned.

Metaheuristic algorithms for the PFSP appeared much later than the heuristic. Among the earliest approaches, we find the simulated annealing algorithm of [26], which is fairly simple in the sense that it uses a constant temperature and an insertion neighborhood. About the metaheuristics developed in the master thesis the first and more important is the Iterated greedy proposed by [30] to the makespan objective. IG generates a sequence of solutions by iterating over greedy constructive heuristics using two main phases: destrution and construction. Another metaheuristic developed in this master thesis with Core idle time minimization is the IGALL proposed by [7]. It is based on the same concept of the Iterated Gready with optimization of partial solutions.

The last one method adapted from the PFSP with Total Flow time minimization is the VBIH. It is developed by [34] based on a variable block insertion.

To summarize all the methods applied, the article of reference and the adaptation made is the table 2.1

Table 2.1 Summarize of the methods applied.

Method	Reference	Problem	Adaptation
NEH	[25]	$Fm\|prmu\|C_{max}$	Objective function
NEH$_M$	[28]	$HFm\|prmu\|C_{max},TCE$	Objective function and type of shop floor
LR-NEH	[29]	$Fm\|prmu\|\sum C_j$	Objective function
PFH-NEH	[27]	$Fm\|prmu\|TCE,\sum C_j$	Objective function
IG	[30]	$Fm\|prmu\|C_{max}$	Objective function
IGALL	[7]	$Fm\|prmu\|C_{max}$	Objective function
VBIH	[34]	$Fm\|prmu\|\sum C_j$	Objective function

Along these studies, the most widely used benchmark has been the one proposed by [33]. Lately [35] carried out a study of problem instances to develop a new hard benchmark with difficult problems, which shows significant differences between algorithms. This master thesis uses these two benchmarks, the first one for the calibration and the second one for the comparison.

2.3 Core idle time

As already mentioned, since the first work published by [19] a lot of studios are published about makespan and total flow time objectives. However, as we already mentioned, other measures are more suitable in some industrial environments, such as idle and waiting times.

In flow shop scheduling, waiting time of jobs and idle time of machines are often considered because of their impact on many manufacturing environments and applications. In the literature, no-wait and no-idle constraints are considered for problems minimizing C_{max} mainly.

Core Idle time is found in the literature introduced in the objective function, in some heuristics method and as constraints. Related with the objective function, idle time is usually found in the minimization of the total machine completion time.

[18] developed efficient solutions method for the minimization of the

completion time for two set o dominates machines. [14] discussed the minimization of weighted sum of machine completion times with minimal and maximal time lags and provided the np-hard nature of the problem.

[31] considered the problem of minimizing C_{max} and $\sum C_j$ with assignment flexibility of tasks. They concluded that there is no solution found which simultaneously minimizes both objectives. Additionally, some multi-criteria approaches have been considered, for example [36]. As [22] summarize to the best of our knowledge, $\sum CIT_i$ have only been rarely discussed as objective functions, for this reason the scope of our master thesis must be developed.

About the latest contributions related on the core idle time we found [4]. In the same line that we previously mentioned, this paper aimed to develop four new MILP models to optimize the total idle and waiting times in the permutation flowshop problem.

We found [1] that studies four single-objective variants of the permutation flowshop scheduling problem, where two objectives are considered: the weighted sum of the makespan and the core waiting time, and the weighted sum of the makespan and the core idle time. For each objective, both the problem with the assumption of semi-active solution and the one without it are considered. Two Mixed Integer Linear Programming (MILP) formulations and one Constraint Programming (CP) formulation are presented.

To conclude, in the literature review is observed that the TCIT minimization is not adressed by aproximated methods. It is therefore proposed to adapt these heuristics and metaheuristics and to be able to obtain this analysis.

3 Constructive heuristics

In this section, we describe the simple heuristic NEH, and constructive heuristics NEH$_M$, PFH-NEH, and LR-NEH, previously mentioned in Section 1, and adapted for the TCIT objective.

3.1 NEH

The NEH was developed by [25] for the PFSP with the makespan objective. We use the same procedure with the TCIT criterion. The pseudo-code of the method is presented in 1.

Previously we shall define the data needed to compute this heuristic. The only data needed are the number of jobs, the number of machines, and the processing times.

The first step is to define the initial sequence. The NEH behold sorts all jobs in non-increasing order of the sum of their processing times. Previously the sum of the processing times is calculated as:

$$P_j = \sum_{i=1}^{m} (p_{ij}) \tag{3.1}$$

Once is obtained the initial sequence starts the body of the algorithm. First, a one-size sequence with the first job of the non-increasing order sequence is composed. Next step consists in insert the following job from the initial sequence on all possible positions of the partial sequence selected in the previous step. Then, in the position in which suppose the minimum core idle time the job is inserted.This step is repeated until the last job of the partial sequence is inserted. When the last job is inserted we obtained a sequence with size n.

Related to the complexity of the problem, [33] proposed an improvement to reduce in orther to run in time $O(n^2m)$ instead of $O(n^3m)$ for the minimization of makespan as obejctive function. This acceleration consists on calculate:

- The earliest completion time of each job on each machine of the partial sequence of size k-1.

$$e_{ij} = \max\{e_{i,j1}, e_{i1,j} + p_{ij}\}, j = 1...k1; i = 1...m \tag{3.2}$$

- The time between the starting time of the job and the final of the operations.

$$q_{ij} = \max\{q_{i,j+1}, q_{i+1,j}\} + p_{ij}, j = 1...k1; i = 1...m \tag{3.3}$$

- The third step is to calculate the earliest completion time in all machines for all jobs due to the insertion of the job k:

$$f_{ij} = \max\{f_{i-1,j}, e_{i,j} - 1 + p_{ik}\}, j = 1...k; i = 1...m \tag{3.4}$$

Finally, the makespan of the partial sequence with the job k inserted on the i_{th} position is:

$$C_{mj} = \max\{f_{ij} + q_{ij}\} \tag{3.5}$$

Doing this process is not necessary to compute the complete makespan for each partial solution. However, the acceleration cannot be applied for other objectives such as the total completion time [8] and it cannot be applied to the total core idle time either.

3.2 NEH$_M$

In addition, many NEH-derived methods are implemented in order to improve the original NEH. Some of these variants are in the line of change the initial sequence.

In this document, the modification of the NEH takes into account the strong impact of the first job of the sequence in the idle time of each machine. The NEH-M(X) heuristic was proposed by [28] to the hybrid flowshop scheduling problem minimizing total energy consumption and total flow time. The reason to implement this heuristic for our problem (still being developed for another problem) is the already mentioned relationship with the core idle time. The pseudo-code of the method is presented in 12.

Related to the input needed to schedule the heuristic, the data needed are the number of jobs, the number of machines, the processing times. X determine the number of times that the NEH algorithm is repeated changing the first job of the initial sequence. The details of the heuristic are explained below. Initially, the jobs are sorted in the same way as in the NEH. The obs are sorted in non-increasing order of their processing time: Previously the sum of the processing times is calculated as:

$$P_j = \sum_{i=1}^{m} (p_{ij}) \tag{3.6}$$

Once this initial sequence is ordered by the criterion previously mentioned start the body of the algorithm. The first iteration is the same as the original NEH, a one-size sequence with the first job of the non-increasing order sequence composed. The next step consists in insert the following job from the initial sequence on all possible positions of the partial sequence selected in the previous step. Then, in the position in which suppose the minimum core idle time the job is inserted.

Then, the remainder of iterations are generated by choosing different jobs as the first job and implementing this procedure. In the h-th iteration, the job in position h in the initial sequence is chosen as the first job of the partial solution. Once the first job of this h-th solution is selected, the NEH procedure is applied.

Finally, X n-size solutions generated by applying x times the NEH algorithm with different first jobs are obtained. The last step is to choose the one with the minimum value of the objective function. A summary of this heuristic is presented by.

3.3 LR-NEH

The next heuristic implemented for our problem is the LRNEH(X) heuristic developed by [29] to the minimization of the total flow time. Should be noted that takes into account the main influence of the first job in the objective function. It is a composition of the classical NEH and the LR developed by [21] for the minimization of the total flow time. Related to the input needed to schedule the heuristic, the data needed are the number of jobs, the number of machines, the processing times, and the parameter d.

First, the LR(x) algorithm represented in the pseudo-code 12 is explained. For the best understanding of the heuristic, previously should be defined the index function that will be the main important concept of the algorithm.

To calculate the index function, we start to having a partial sequence formed by k already scheduled jobs (π) and a partial sequence formed by n-k unscheduled jobs (U). This index function is composed by:

- The weighted total machine idle time, calculated for each job j $\in$ U and each position k $\in$ π that can be occupied:

$$IT_{j,k} = \sum_{i=2}^{m} \frac{m \max\left(C_{i-1,j}, C_{i,[k]}\right)}{i + k(m-i)/(n-2)} \tag{3.7}$$

- The total flowtime of jobs j and λ, where λ is a single artificial job considering the other jobs in U and its processing time is the average of the processing times of these jobs:

$$AT_{j,k} = C_{m,j} + C_{m,\lambda} \tag{3.8}$$

Summarizing this two terms:

- $C_{i,[k]}$: The completion time of the job in the k$-$th position at machine i.
- $C_{m,j}$: The completion time of the job j in the last machine.
- $C_{m,\lambda}$: The completion time of the artificial job λ.

Then, the index function is computed as:

$$\xi_{j,k} = (n-k-2) IT_{j,k} + AT_{j,k} \tag{3.9}$$

About procedure, the LR heuristic begins calculating the index function for the n jobs. This initial sequence in generated sorting this n jobs in ascending order of their index function.

Then, the body of the procedure is repeated until generate x n$-$size sequence. For each iteration l $\in$ x, remove the l job of the previous sequence, in which initially, jobs are sorting in ascending order of their index function to generate a one-size sequence with the remove job. The next step is to calculate the index function for the remaining jobs. The job with the minimum value of the index function is inserted in the one$-$size sequence. This procedure based on calculate the index function of the remaining jobs and inserted in the sequence mentioned is repeated until generate a n-size sequence. Should be noted that ties are broken by selecting the one with the minimum weighted total machine idle time. This procedure is repeated x times, generating x sequence. The sequence with the minimum objective function is selected as final sequence.

Once the LR(x) heuristic is explained, we will now to understand the LR−NEH(x) constructive heuristic. The pseudo-code of the method is presented in 15.

As mentioned above, the heuristic depend on the parameter d. It determines the criteria of change to apply the LR to apply the NEH heuristic. The LR−NEH(x) is based on generate x sequences applying both algorithm. Next, this procedure is explained.

The constructive heuristic begins applying the LR method. Then, the index function is calculated to the n jobs. The job with the minimum value of the index function is remove and insert in the one-size sequence. Then, a d-size sequence is generated applying the LR method.

Starting from a d−size sequence, the NEH algorithm is apply. The n−d jobs are sorting in non-decreasing order of their processing time. Then, the final sequence is generated inserting all the remaining jobs in all positions, selecting the one with the minimum value of the objective function. The final sequence of this iteration is obtained. As already mentioned, this procedure is repeated x times, generating x different sequence. Each with the minimum value of the objective function is selected.

3.4 PFH-NEH

The next heuristic implemented for our problem is the PFH-NEH(X) heuristic developed by [28] to the minimization of the total flow time and total energy consumption. Should be noted that focuses on the main influence of the first job in the objective function.

Related to the input needed to schedule the heuristic, these are the number of jobs, the number of machines, the processing times, a initial sequence and a job index.

First, the PF(x) algorithm is explained. The pseudo-code of the method is presented in 28. For the best understanding of the heuristic, previously should be defined the cost function that will be the main important concept of the algorithm.

This index function is composed by:

– A weight calculates as:

$$w_i = \frac{m}{i + j(m-i)/(n-2)} \tag{3.10}$$

– The weighted sum of the idle times of job π_j on all machines:

$$IT_j = \sum_{i=2}^{m} w_j \max\left(C_{\pi_j,i-1} - C_{\pi_{j-1},i}, 0\right) \tag{3.11}$$

Then, the cost function is computed as:

$$cF_j = (n-j-1)IT_j + C_{\pi_j,m} \tag{3.12}$$

About procedure, the PF heuristic part of a initial sequence (π) and a job index. The first step is to remove of the π sequence the job in the position indicate by the job index. Then, the cost function is calculated for the remaining jobs. The job with the minimum value of the cost function is removed and inserted in the final sequence (Π). These steps are repeated until a n−size sequence is formed.

Once the PF heuristic is explained, let us now to understand the PFH−NEH(x).The pseudo-code of the method is presented in in 13. The PFH−NEH(x) is based on generate x sequences applying both algorithm. Next, this procedure is explained.

The initial order is determined by calculating the fD_i values as given below:

$$fD_j = \frac{2}{m-1}\sum_{i=1}^{m}(m-i)p_{\pi_j,i} + \sum_{i=1}^{m} p_{\pi_j,i} \tag{3.13}$$

Then the body of the algorithm is repeated x times. In the $h-th$ iteration, the PF heuristic is applied based on the input of this initial order and the h_th position as job index. Once a $n-size$ sequence is obtained applying the PF algorithm, the NEH heuristic is applied. The NEH starts by the sequence formed by the PF method. Then a one-size sequence with the first job is generated. The remaining jobs are inserted in the position in which suppose the minimum core idle time.

Then the sequence with the minimum value of the objective function is selected as output of the method. The last step is to apply the insertion local search with general swaps to the complete solution as long as it is improved. The pseudo-code of the local search is presented in10]

3.5 Local search

About the local search, a first improvement with general swaps is applied in all heuristics and metaheuristics that contain it.

It starts with a initial sequence previously generated. Then the body of the algorithm is applied. It is based on insert a determinate job in all position until a improvement of the objective function is found.

The procedure stars by removing the first job of the sequence and inserting in all possible position on the sequence. This procedure is repeated for all the jobs in all positions.

The algorithm ends when one of these two options occurs:

- The algorithm ends when a better solution is reached, even if it has not been tested with the remaining works
- When all the jobs in the sequence have been inserted in all possible position of the sequence.

Then, the best solution is selected.

Algorithm 1: NEH [27]

```
Input: instance data
Output: $\Pi_b$, $obj_b$
1 begin
2     $P_j$:=total processing time of each job j ;
3     $\Pi^0$:=sort the jobs in descending order of $P_j$;
4     $\pi_1$:=$\pi_1^0$;
5     for $k = 2$ to $n$ do
6         $\Pi$=Insert $\pi_k^0$ in the best position of $\Pi$;
7     $\Pi_b$:=$\Pi$;
8     $obj_b$:=$Obj(\Pi_b)$;
9     return $\Pi_b$, $obj_b$
```

Algorithm 2: NEH-M [28]

```
Input: instance data, $x$
Output: $\Pi_b$, $obj_b$
1  begin
2      $P_j$:=$total\ processing\ time\ of\ each\ job\ j$ ;
3      $\Pi^0$:=$sort\ the\ jobs\ in\ descending\ order\ of\ P_j$;
4      $\Pi'$:=$\Pi^0$;
5      for $h = 1$ to $x$ do
6          Swap the positions of jobs $\pi'_1$ and $\pi'_h$;
7          $\pi^h_1$:=$\pi'_1$;
8          for $k = 2$ to $n$ do
9              $\Pi^h$=Insert $\pi'_k$ in the best position of $\Pi^h$;
10     $\Pi_b$:=The best solution among ($\Pi^1$,$\Pi^2$...$\Pi^x$);
11     $obj_b$:=$Obj(\Pi_b)$;
12     return $\Pi_b$, $obj_b$
```

Algorithm 3: LR

```
Input: instance data,x
Output: $\Pi_b$, $obj_b$
1  begin
2      $\alpha^0$:=sort the jobs in ascending order of $\xi_{j0}$;
3      for $l = 1$ to $x$ do
4          $\pi_1^l$:=$\alpha_l$;
5          $U$:=Remove $\alpha_l$ from $\alpha$;
6          for $k$=2 to $n$ do
7              Calculate $\xi_{jk}$ for all the jobs from $U$;
8              $\pi_k^l$:=Take the job with minimum $\xi_{jk}$;
9              Remove job $j$ from $U$;
10     $\Pi_b$:=The best solution among ($\Pi^1$,$\Pi^2$...$\Pi^x$);
11     $obj_b$:=$Obj(\Pi_b)$;
12     return $\Pi_b$, $obj_b$
```

Algorithm 4: LR-NEH

```
Input: instance data,x
Output: $\Pi_b$, $obj_b$
1  begin
2      $\alpha^0$:=sort the jobs in ascending order of $\xi_{j0}$;
3      for $l = 1$ to $x$ do
4          $\pi_1^l$:=$\alpha_l$;
5          $U$:=Remove $\alpha_l$ from $\alpha$;
6          for $k$=2 to $d$ do
7              Calculate $\xi_{jk}$ for all the jobs from $U$;
8              $\pi_k^l$:=Take the job with minimum $\xi_{jk}$;
9              Remove job $j$ from $U$;
10         $\beta$:=sort $U$ in ascending order of total processing times;
11         for $k$=1 to $n-d$ do
12             $\Pi^l$=Insert $\beta_k$ in the best position of $\Pi^l$;
13     $\Pi_b$:=The best solution among ($\Pi^1$,$\Pi^2$...$\Pi^x$);
14     $obj_b$:=$Obj(\Pi_b)$;
15     return $\Pi_b$, $obj_b$
```

Algorithm 5: PF [27]

```
   Input: instance data, $\Pi^*$, $Index$
   Output: $\Pi_b$, $obj_b$
1  begin
2      $U$:=$\Pi^*$;
3      $\pi_1$:=$u_{index}$;
4      $U$:=$Remove\ u_{index}\ from\ U$;
5      for $j = 2\ to\ n$ do
6          for $k = 1\ to\ n - j$ do
7              $\pi_j$=$U_k$;
8              $Calculate\ C_{\pi_{j,i}}\ \forall\ i$;
9              $It_j$=$\sum_2^m w_j max(C_{\pi_{j,i-1}} - C_{\pi_{j-1,i}}, 0)$;
10             $cFj$=$(n - j - 1)$x$It_j + C_{\pi_{j,m}}$;
11             if $k$=$1$ then
12                 $job$=$\pi_j$;
13                 $BestCoef$=$cFj$;
14                 $BestI$=$It_j$;
15             if $cFj \lhd BestCoef$ then
16                 $job$=$\pi_j$;
17                 $BestCoef$=$cFj$;
18                 $BestI$=$It_j$;
19             if $cFj$=$BestCoef$ then
20                 if $It_j \lhd BestI$ then
21                     $job$=$\pi_j$;
22                     $BestCoef$=$cFj$;
23                     $BestI$=$It_j$;
24         $\pi_j$=$job$;
25         $Remove\ pi_j\ from\ U$;
26     $\Pi_b$:=$\Pi$;
27     $obj_b$:=$Obj(\Pi_b)$;
28     return $\Pi_b$, $obj_b$
```

Algorithm 6: PFH-NEH [27]

```
   Input: instance data
   Output: $\Pi_b$, $obj_b$
1  begin
2      $\Pi^0$:=$sort\ the\ jobs\ in\ ascending\ order\ of\ fD_j$;
3      if $n \lhd = 100$ then
4          $x = n$;
5      else
6          $x = 1$;
7      for $k = 1$ to $x$ do
8          $\Pi'$=PF($\pi^0_{,k}$);
9          $\Pi''$=NEH($\Pi'$);
10         $\Pi^k$=LocalSearch($\Pi''$);
11     $\Pi_b$:=The best solution among ($\Pi^1$,$\Pi^2$...$\Pi^x$);
12     $obj_b$:=$Obj(\Pi_b)$;
13     return $\Pi_b$, $obj_b$
```

Algorithm 7: Local search

```
   Input: instance data, $\Pi_0$, $obj_0$
   Output: $\Pi_b$, $obj_b$
1  begin
2      $\Pi_b$:=$\Pi_0$;
3      $obj_b$:=$obj_0$ for $j = 1$ to $n$ do
4          for $k = 1$ to $n$ do
5              if $k! = j$ then
6                  $\Pi_V$=Insert $\pi^0_j$ in the k position of $\Pi_0$;
7                  if $Obj\Pi_V \lhd obj_b$ then
8                      $k := n\ j =: n\ \Pi_b$:=$\Pi_V$;
9                      $obj_b$:=$Obj(\Pi_b)$;
10     return $\Pi_b$, $obj_b$
```

4 Metaheuristics

The next step, with the purpose of obtaining better solutions than with the previous heuristics and enrich the problem evaluation, the implementation of a metaheuristic seems appropriate. In this section, IG, IGALL and VBIH are presented as the metaheuristics adapted for our problem.

Note that, the main adaptation besides the objective function, is the initial sequence. Has been decided to tested each of the metaheuristics with all the heuristics as initial sequence. The objective is to make a complete analysis of the problem.

4.1 Iteraty Greedy

The first metaheuristic adapted is the original Iterated Greedy (IG) algorithm. The pseudo-code of the method is presented in 22.The algorithm is composed of two phases. The destruction phase removes some jobs from a previously input sequence and the construction phase consists on the application of a greedy constructive heuristic to complete the sequence generated by the destruction phase. Then, a selection criterion must be applied to decide to apply the iteration to the sequence recently formed or to the previous one. This procedure is repeated until a stopping criterion is met.

To a best understanding, should be explained the original method developed by [30] for $Fm|prmu|C_{max}$. This original IG starts with a initial solution obtained by the NEH heuristic. The destruction phase is based on eliminate d jobs of the sequence π, generating a partial sequence π_D with n-d jobs and another sequence π_R with these d jobs removed.

Then the construction phase is applied. It is based on insert the removes jobs applying the NEH mechanism. Each job of the π_p partial sequence is inserted in all positions of the π partial sequence. The position in which the best objective function is obtained, is definitively inserted. The job inserted in the π_D sequence is removed from the π_p sequence. It is repeated until there are no any jobs in the π_p sequence and the pi_D reaches to be a complete solution.

Additionally, a local search method is added to improve the solution quality, choosing a local search algorithm based on searching a neighborhood by insertion. This is, a job is removed of the sequence and inserted into every possible position on the sequence from the previous solution, when a improvement of the objective function is reached the job is inserted in that position, else no insertion is applied. This step is applied on every job with a randomly order of until an improvement occurs or every job had inserted.

The last step an acceptance criterion to decide whether or not the new sequence is accepted as the initial solution for the next iteration. [30] decides to implement the temperature extracted from the simulated annealing procedure, specifically the temperature that [26] proposed for their simulated annealing algorithm to solve the problem $Fm|prmu|C_{max}$:

$$Temperature = T\frac{\sum_{i=1}^{m}\sum_{j=1}^{n} P_{ij}}{nm10} \tag{4.1}$$

T was empirically tested and decided to be $T = 0.5$. The stopping criterion has been decide to be time-based, related with the size of the instance. It is computed as:

$$t = n\ (m/2)\ 20\ milliseconds. \tag{4.2}$$

Once the original IG is explained should be indicated the adaption on our problem. As mentioned above, as the initial sequence has been tested all the heuristic adapted in our master thesis (NEH, NEH$_M$, LR$-$NEH and PFH$-$NEH). The comparison will be shown in the next chapter. Another adaptation has been the local search applied. Has been compared some local search and has been decided what provides the best results for our problem. The local search implemented is the based on first improvement with general swaps. Should be noted that it is the local search applied in all heuristics and metaheuristics that contain each.

Finally, the main adaption is the objective function computed, being the already mentioned $Fm|prmu|TCIT$ problem.

About the d parameter, is will be decided in the calibration phase.

4.2 IGALL

The next metaheuristic adapted is the IGALL developed by [7]. The pseudo-code of the method is presented in 23. It is a variation of the original IG added a local search to the partial sequence. The first step is the destruction phase, d jobs are removed by the initial sequence and stored in a partial sequence.

It is time to apply the modification of the Iteraty Greedy. The next step is to implement a local search to the partial solution. The application of this phase can lead to different search directions.

Then the construction phase based on insert the removes jobs applying the NEH mechanism is apply to the improved partial sequence. Each job of the π_R improve partial sequence is inserted in all positions of the π partial sequence. The position in which the best objective function is obtained, is definitively inserted.

Additionally, as the IG, the local search [10] is applied to improve the quality of the solution. As mentioned above, a local search based on first improvement with general swaps is implemented.

The last step an acceptance criterion to decide whether or not the new sequence is accepted as the initial solution for the next iteration. The same criteria based on temperature extracted from the simulated annealing procedure is used.

$$Temperature = T\frac{\sum\limits_{i=1}^{m}\sum\limits_{j=1}^{n}P_{ij}}{nm10} \tag{4.3}$$

T was empirically tested in the next chapter and the stopping criterion is the same as is explained.

4.3 VBIH

The third and last metaheuristic adapted is the VBIH algorithm developed by [34] for the PFSP with total completion time and blocking constraint. Worth mentioning that it is similar to the IG and IGALL already explained. The pseudo-code of the method is presented in 25.

In the original VBIH, the initial sequence considered is obtained by the application of the PFH-NEH heuristic. In our adaptation will be tested with all the heuristics developed in the master thesis. About the body, as in the previous two method a destruction phase is applied.

The main difference between VBIH and the IG is the number of jobs removed in the destruction phase. The VBIH method is based on removed a block of a variable number of jobs in each iteration. Block moves with different block sizes can be employed because the block size b changes during the algorithm, with $b \in (bmin, bmax)$.

The next step is similar to the IGALL, a local search 10 is applied to the partial sequence of size b. As above mentioned, the local search is based on first improvement with general swaps. After that, the destruction phase is implemented. The improve block of job is inserted in all possible position of the partial sequence of size $n-b$. The position in which the best objective function is obtained, is definitively inserted. Then, a local search based on first improvement with general swaps is applied to the complete sequence.

Once the block is inserted, if the new solution of the construction phase is better than the previous best solution, it is selected as the new best solution. Then, the same block size is kept as long as the solution improves. Otherwise, the block size is changed incrementally $(b = b+1)$.

If the solution is worse, an acceptance criterion to decide whether or not the new sequence is accepted as the initial solution for the next iteration is implemented. The same criteria based on temperature extracted from the simulated annealing procedure is used. T was empirically tested in the next chapter and the stopping criterion is the same as is explained.

Algorithm 8: IG [27]

```
Input: instance data, $T$, $k$
Output: $\Pi_b$, $obj_b$
begin
    $\Pi_0 := (\pi_1, \ldots, \pi_n)$ initial sequence;
    $obj_0 := Obj(\Pi)$;
    $\Pi_b := \Pi_0$;
    $obj_b := obj_0$;
    $\Pi^d := \varnothing$, $\Pi^p := \varnothing$;
    while Stopping criterion is not satisfied do
        $\Pi^d := Destruction(\Pi_0, k)$;
        $\Pi' := Construction(\Pi^d, \Pi^p)$;
        $\Pi'' := LocalSearch(\Pi')$;
        $obj'' := Obj(\Pi'')$;
        if $obj'' \leq obj_0$ then
            $\Pi_0 := \Pi''$;
            $obj_0 := obj''$;
            if $obj'' \lhd obj_b$ then
                $\Pi_b := \Pi''$;
                $obj_b := obj''$;
        else
            if $r \lhd exp(-(obj'' - obj_0)/T)$ then
                $\Pi_0 := \Pi''$;
                $obj_0 := obj''$;
    return $\Pi_b$, $obj_b$
```

Algorithm 9: IGALL [7]

```
   Input: instance data, $T$, $k$
   Output: $\Pi_b$, $obj_b$
1  begin
2      $\Pi_0 := (\pi_1, \ldots, \pi_n)$ initial sequence;
3      $obj_0 := Obj(\Pi)$;
4      $\Pi_b := \Pi_0$;
5      $obj_b := obj_0$;
6      $\Pi^d := \varnothing$, $\Pi^p := \varnothing$;
7      while Stopping criterion is not satisfied do
8          $\Pi^d = Destruction(\Pi_0, k)$;
9          $\Pi^p$=$Localsearch(\Pi^p)$;
10         $\Pi^{'}$=$Construction(\Pi^d, \Pi^p)$;
11         $\Pi^{''}$=$LocalSearch(\Pi^{'})$;
12         $obj''$:=$Obj(\Pi'')$;
13         if $obj'' \lhd = obj_0$ then
14             $\Pi_0$:=$\Pi''$;
15             $obj_0$:=$obj''$;
16             if $obj'' \lhd obj_b$ then
17                 $\Pi_b$:=$\Pi''$;
18                 $obj_b$:=$obj''$;
19         else
20             if $r \lhd exp(-(obj'' - obj_0)/T)$ then
21                 $\Pi_0$:=$\Pi''$;
22                 $obj_0$:=$obj''$;
23     return $\Pi_b$, $obj_b$
```

Algorithm 10: VBIH [27]

```
Input: instance data, $T$, $bmin$, $bmax$
Output: $\Pi_b$, $obj_b$
begin
    $\Pi_0 := (\pi_1, \ldots, \pi_n)$ initial sequence;
    $obj_0$:=$Obj(\Pi)$;
    $\Pi_b$:=$\Pi_0$;
    $obj_b$:=$obj_0$;
    while Stopping criterion is not satisfied do
        $b$:=$b_{min}$;
        while $b \lhd b_{max}$ do
            $\Pi^p$:=Remove a block of a jobs from $\Pi_0$;
            $\Pi^{p\prime}$:=$LocalSearch(\Pi^p)$;
            $\Pi'$:=Insert the block into the best position of $\Pi^{p\prime}$;
            $\Pi''$:=$LocalSearch(\Pi')$;
            $obj''$:=$Obj(\Pi'')$;
            if $obj'' \lhd = obj_0$ then
                $\Pi_0$:=$\Pi''$;
                $obj_0$:=$obj''$;
                if $obj'' \lhd obj_b$ then
                    $\Pi_b$:=$\Pi''$;
                    $obj_b$:=$obj''$;
            else
                $b = b + 1$;
                if $r \lhd exp(-(obj'' - obj_0)/T)$ then
                    $\Pi_0$:=$\Pi''$;
                    $obj_0$:=$obj''$;
    return $\Pi_b$, $obj_b$
```

5 Computational results

5.1 Testbeds description

In order to compare the heuristics and metaheuristics adapted for our problem, a testbed should be selected. If the optimally solution could be obtained, the result of the method would be with this optimal solution. In the reality, a comun testbed should be used to compare the proposed method with others method implemented. To have the capacity to compare with method proposed by others studies an existing testbed is used.

The most used beachmarks nowdays is the one proposed by [33]. However, [35] proposed a beachmarks with the objective to be amenable for statistical analysis and discriminant when several algorithms are compared. Both consists in generate a set of instances with processing times between 1 and 99.

Althought this testbed are proposed to minimixing the makespan, we evaluate these sequences for our problem in our study in order to compare the Total Core Idle time obtained by the different algorithm. In order to used the two testbed presented, first, the well-known Taillard testbed [33] has been used to calibrate. It is distributed between $n \in \{20,500\}$ and $m \in \{5,50\}$. The combination of number of jobs and machines and the number of instance for each size is represented in 5.1. About the processing times, are uniformly

Table 5.1 Number of instances by size of Taillard instances.

n x m	**5**	**10**	**20**	**50**
20	10	10	10	
50	10	10		10
100	10	10	10	
200		10	10	
500			10	

generated between 1 and 99. Each size has 10 instances, in total 120 instances. The benchmark set of instances from Vallada has been used to compare all the heuristics and metaheuristics. It consists of 240 large instances and 240 small instances.

Small instances are a set of 240 with the following combinations of number of jobs (n) and number of machines (m): $n = \{10, 20, 30, 40, 50, 60\}$, $m = \{5, 10, 15, 20\}$. It could be seen in 5.2. For each combination 10 instances are generated, so in total, we have $6 \times 4 \times 10 = 240$ small instances. Regarding the large instances, they are also a set of 240 where $n = \{100, 200, 300, 400, 500, 600, 700, 800\}$ and $m = \{20, 40, 60\}$. It could be seen in 5.3

Table 5.2 Number of instances by size of Vallada's small instances.

n x m	**5**	**10**	**15**	**20**
10	10	10	10	10
20	10	10	10	10
30	10	10	10	10
40	10	10	10	10
50	10	10	10	10
60	10	10	10	10

Table 5.3 Number of instances by size of Vallada's big instances.

n x m	**20**	**40**	**60**
100	10	10	10
200	10	10	10
300	10	10	10
400	10	10	10
500	10	10	10
600	10	10	10
700	10	10	10
800	10	10	10

The reason why we selected Vallada to compare the different methods is that Taillard's instances are not equidistant, which means, the difference between the number of jobs/machines between two consecutive instances is not the same. In order to make a best statistical analysis the equidistant set of instances generated by [35] is selected.

5.2 Evaluation of the results

To calibrate using the Taillard instances and to compare using the Vallada instances the Relative Deviation Index (RDI) has been selected as the performance measure. It is a indicator used to size the result obtained compared to the best and worst results reached. It is computed by the following equation:

$$RDI_i = \frac{TCIT_{im} - TCIT_{imin}}{TCIT_{imax} - TCIT_{imin}} \tag{5.1}$$

with $TCIT_{im}$ the objective value for instance i obtained by method m, $TCIT_{imax}$ the maximum value obtained for instance i across all the methods, and $TCIT_{imin}$ the minimum.

5.3 Heuristics calibration

In this section, the constructive heuristics (NEH$_M$, LR-NEH, and PFH-NEH) have been calibrated. To calibrate the parameter of the constructive heuristics, we carry out a design of experiments. For this purpose, we use Taillard's benchmark set.

As mentioned above, due to the importance of the first job of the sequence on the objective function, all of the constructive heuristics is based on repeated some procedures changing their initial job. In each iteration, when the first the job is selected the body of the algorithm is applied. This process is repeated x times.

In the design of experiments, the proposed heuristics have one parameter: the number of solutions generated (x). The main characteristic of the parameter x is the dependence of the number of jobs of the instance. If x is set as a function of n, as n grows, so does the number of iterations. Is clearly remarkable that as n increases, x increases then the CPU inverted increases. For this reason, the decision of the value of the parameter x should be made checking if the CPU inverted compensates or not. Regarding these considerations, the cases are shown in Table 5.4.

Table 5.4 Cases of constructive heuristics.

CASE 1	CASE 2	CASE 3	CASE 4
$X = n$ if $n \leq 100$	$X = n$ if $n \leq 200$	$X = n$ if $n \leq 100$	$X = n$ if $n \leq 200$
$X = 1$ if $n > 100$	$X = 1$ if $n > 200$	$X = 20$ if $n > 100$	$X = 20$ if $n > 200$

To the best knowledge of the parameter, the 2 criteria to be taken into account will be explained below:

- The number of job from which the number of job change.
- The value of x that assume since the change.

Then, each of the cases will be explained:

- CASE 1: x is set as n for all the instances with a number of jobs less than 100. At the moment the number of jobs exceeds 100, x is set as 1.
- CASE 2: x is set as n for all the instances with a number of jobs less than 200. At the moment the number of jobs exceeds 200, x is set as 1.
- CASE 3: x is set as n for all the instances with a number of jobs less than 100. At the moment the number of jobs exceeds 100, x is set as 20.
- CASE 4: x is set as n for all the instances with a number of jobs less than 200. At the moment the number of jobs exceeds 200, x is set as 20.

It should be noted that is the combination of two variation of the two criteria:

- The number of job from which the number of job change can be 100 or 200.
- The value of x that assume since the change can be 1 or 20.

In order to compare the cases, we calculate the ARDI (Average Relative Deviation Index) and The average computation time (Avg. CPU), in which the algorithm finds the final solution.

5.3.1 NEH$_M$

The parameter to be calibrated is x, i.e. the number of iterations. The decision of the parameter should be taken on the basis of the relation of the CPU inverted and the quality of the solution. As shown Figure 5.1 the case that obtain the best results is the Case 4.

Figures 5.2 and 5.1 clearly indicate that the cases with high CPU levels result in better ARDI values. However, no great improvement is obtained compared to the increase in time spent. The plots demonstrate no statistically significant difference between the four cases.

Should be highlighted the power of the original NEH to reach a good solution in few iterations. That repeating the algorithm not improve to a great extend the solution only shows the capacity to reach a good solution in few time.

Due to this considerations, Case 1 ($x = n$ if $n \leq 100$ and $x = 1$ if $n > 100$) is selected for being the fastest in terms of CPU and provide a good solution.

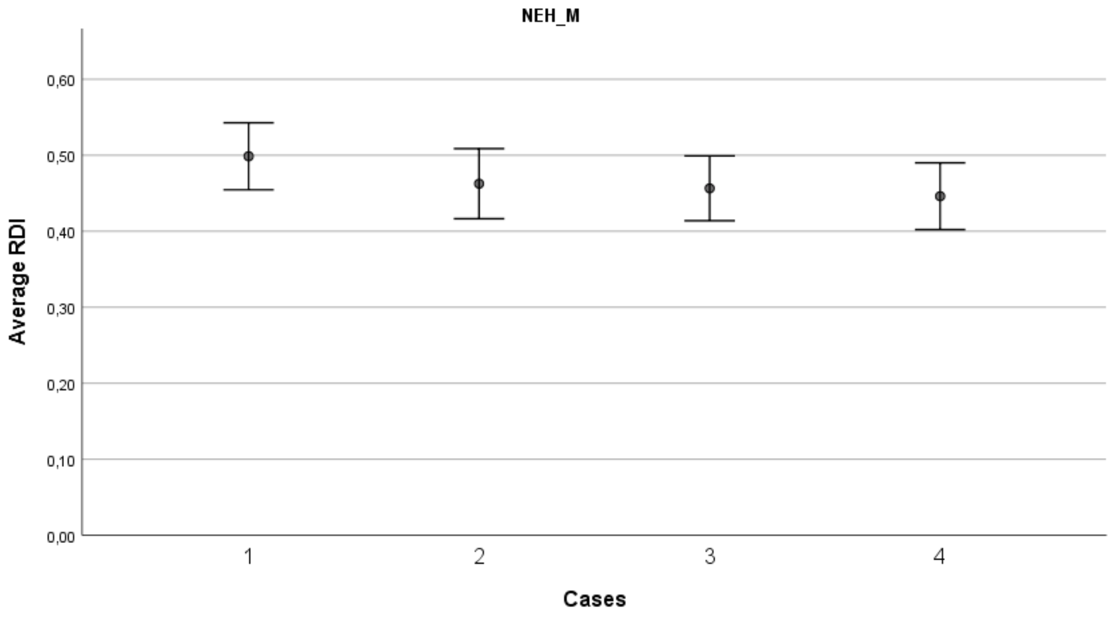

Figure 5.1 95% Confidence Intervals for NEHM.

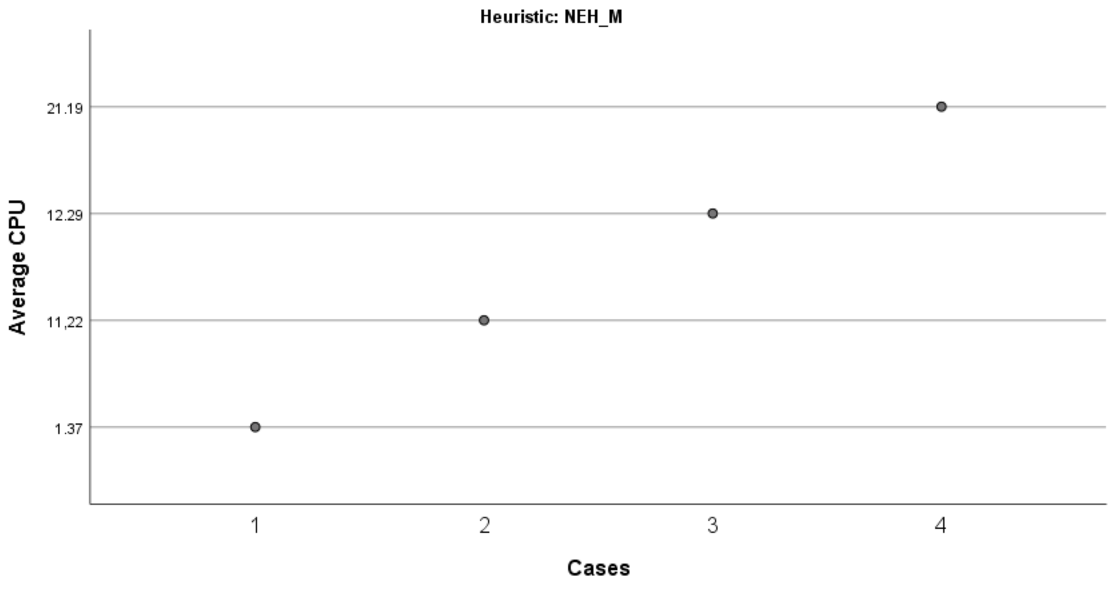

Figure 5.2 CPU average for NEHM.

5.3.2 LR-NEH

The parameter to be calibrated is x, i.e. the number of iterations. The decision of the parameter should be taken on the basis of the relation of the CPU inverted and the quality of the solution. As we can see in Figures 5.3 and 5.4,

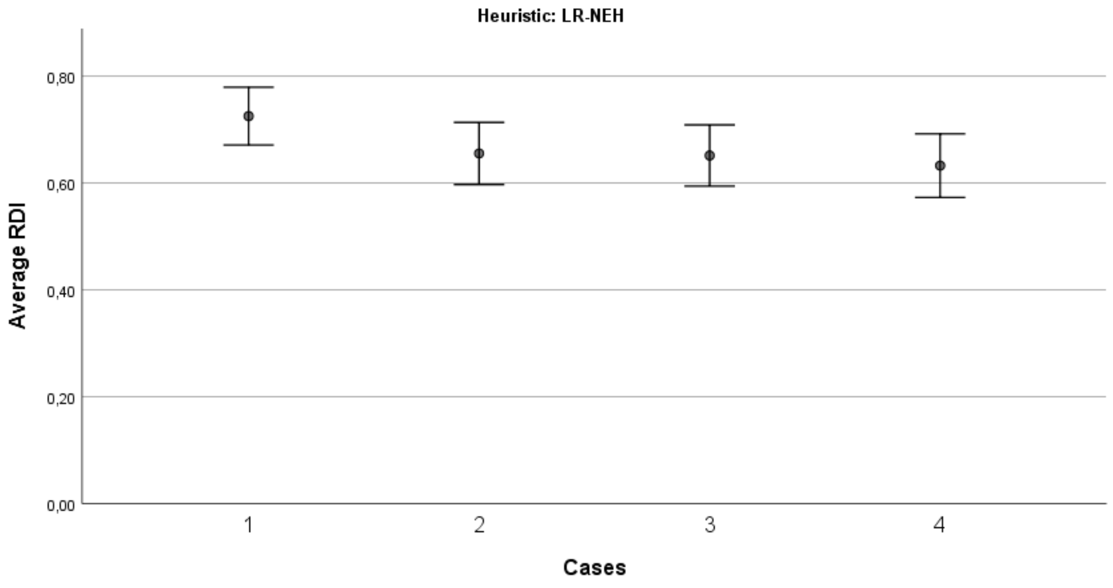

Figure 5.3 95% Confidence Intervals for LR-NEH.

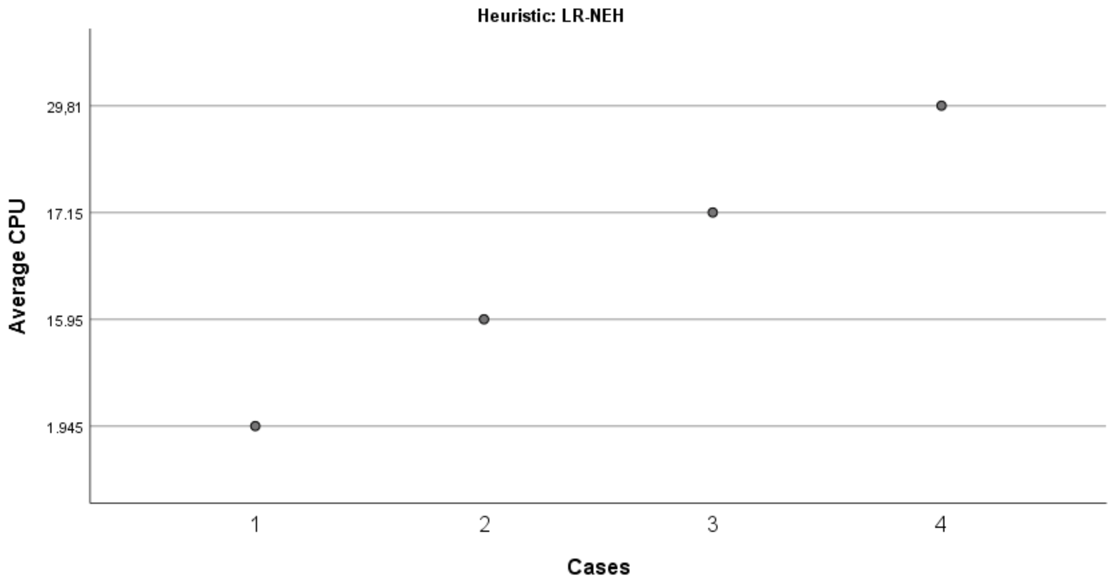

Figure 5.4 CPU average for LR-NEH.

the results clearly indicate that the cases with high CPU levels result in better ARDI values.

However, in this heuristic, the difference between the time spent and

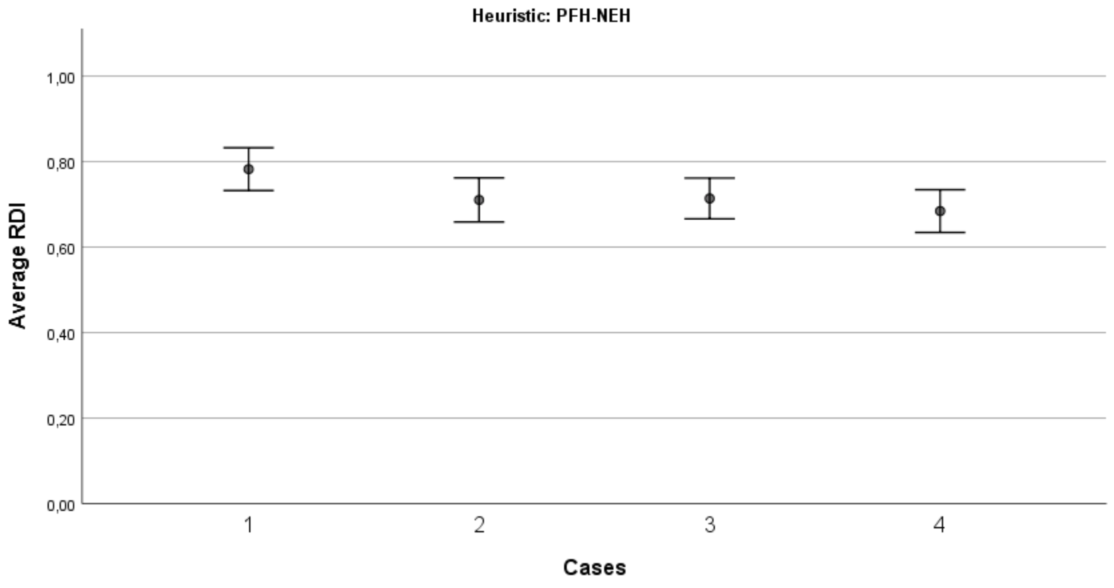

Figure 5.5 95% Confidence Intervals for PFH-NEH.

the improvement of results is even greater. The plots demonstrate no statistically significant difference between the four cases. Therefore, Case 1 ($x = n$ if $n \leq 100$ and $x = 1$ if $n > 100$) is selected for being the fastest in terms of CPU.

5.3.3 PFH-NEH

The parameter to be calibrated is x, i.e. the number of iterations. The decision of the parameter should be taken on the basis of the relation of the CPU inverted and the quality of the solution. As we can see in Figures 5.5 and 5.6, the results clearly indicate that the cases with high CPU levels result in better ARDI values. However, no great improvement is obtained compared to the increase in time spent. The plots demonstrate no statistically significant difference between the four cases. Therefore, as in the other heuristics Case 1 ($x = n$ if $n \leq 100$ and $x = 1$ if $n > 100$) is selected for being the fastest in terms of CPU.

5.4 Heurisics comparison

Once the constructive heuristics are calibrated, is time to compare the results with Vallada test best. In order to compare all the heuristics, we must set the parameter x like in Case 1.

As we can see in 5.7, the plots demonstrate statistically significant differences

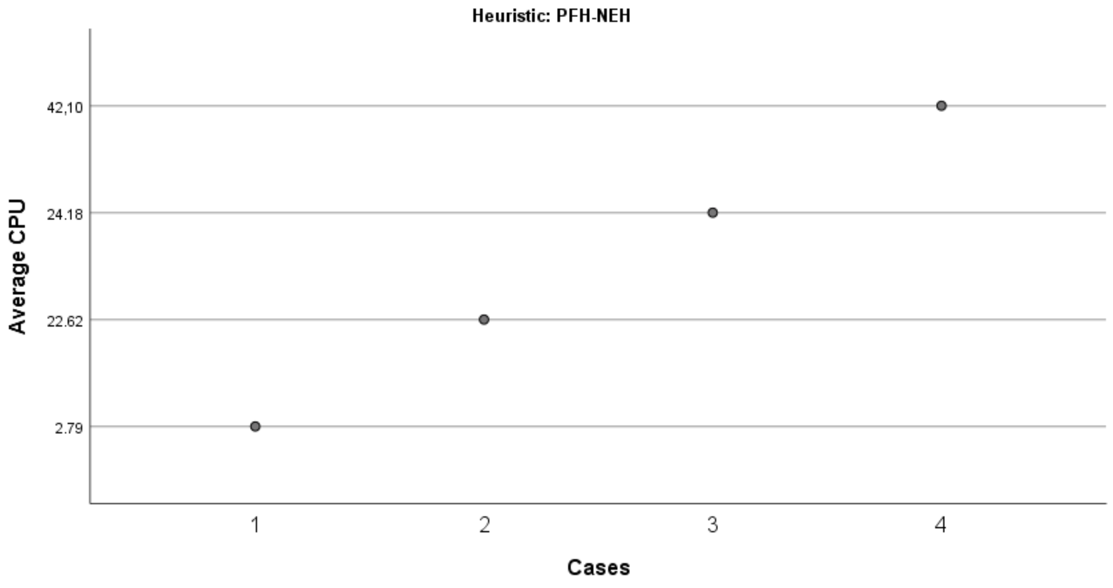

Figure 5.6 CPU average for PFH-NEH.

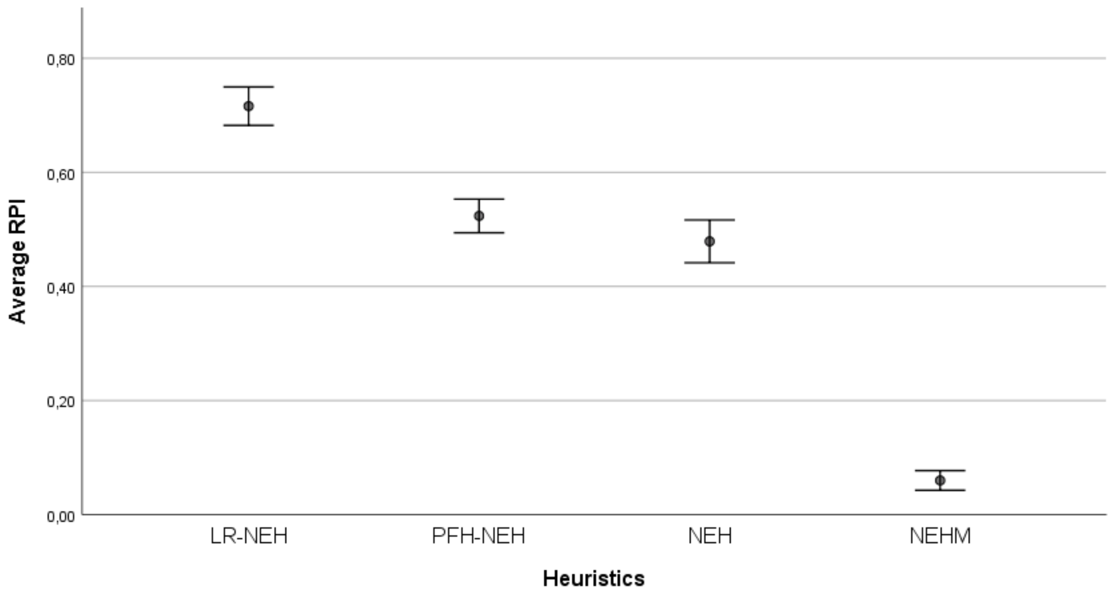

Figure 5.7 95% Confidence Intervals for Heuristics.

between NEH$_M$ and NEH, and between PFH-NEH and LR-NEH. However, there are no significant differences between NEH and PFH-NEH. The best result is provided by the NEH$_M$ heuristic.

5.5 Metaheuristics calibration

5.5.1 IG

Summing up the procedure, it starts with a initial sequence an then the algorithm is applied. All the experiments for the metaheuristics are carried out fixing a stopping criterion of maximum time $t = n\ (m/2)$ 20 milliseconds, so efficiencies will be easy and fast to compare.

The parameters of which depends are:

- The initial sequence obtained by one of the heuristics adapted.
- The parameter T: The parameter of which depend the temperature proposed for the simulated annealing algorithm.
- The parameter d: The number of jobs removed in the destruction phase.

Although NEH$_M$ is the best heuristic and we have just decided to choose the parameter x like Case 1, is should be to compare again for one reason: The NEH$_M$ is the best in terms of RPI but is the most CPU-heavy. We must check if the time invested in the initial sequence is beneficial to the final result. Taking each of the heuristics as a starting sequence, the first step is to compare all heuristics setting Case 1.

The first parameter to calibrate is heuristic. The results in terms of RDI are shown 5.8. As we can see, the plots demonstrate statistically significant differences between LR-NEH and PFH-NEH, but not between NEH$_M$ and LR-NEH.

If we focus on comparing the NEH$_M$ with the LR-NEH, we draw the following conclusions

- If we choose the NEH$_M$ as the initial sequence, we get slightly better results.
- The above conclusion reflects that the more time invested in the initial sequence (NEH$_M$ compare with LR-NEH), the better results we obtain.

The NEH$_M$ is chosen as the initial sequence. The next step is to calibrate the parameter T. The values taken for calibration are $T = \{0.1, 0.2, 0.3, 0.4, 0.5\}$. Therefore we have 5 different combinations to be tested. As we can see in the graph of 5.9 the best result is provided by the $T = 0.2$. Although we can state that $T = 0.2$ provides the best results, there are no significant differences.

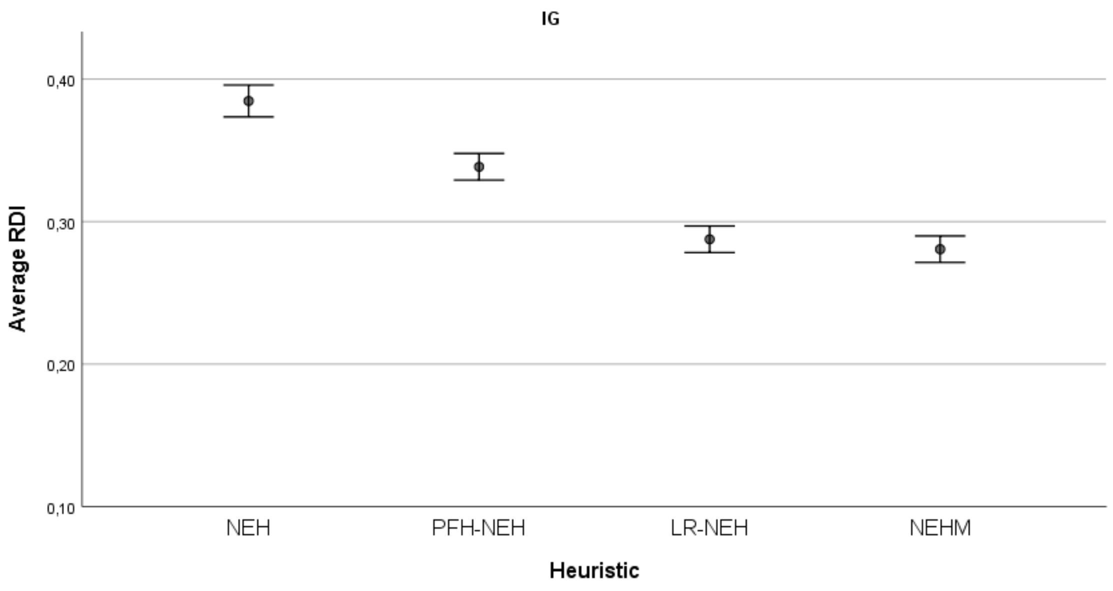

Figure 5.8 95% Confidence Intervals for IG for different Heuristics as the initial sequence.

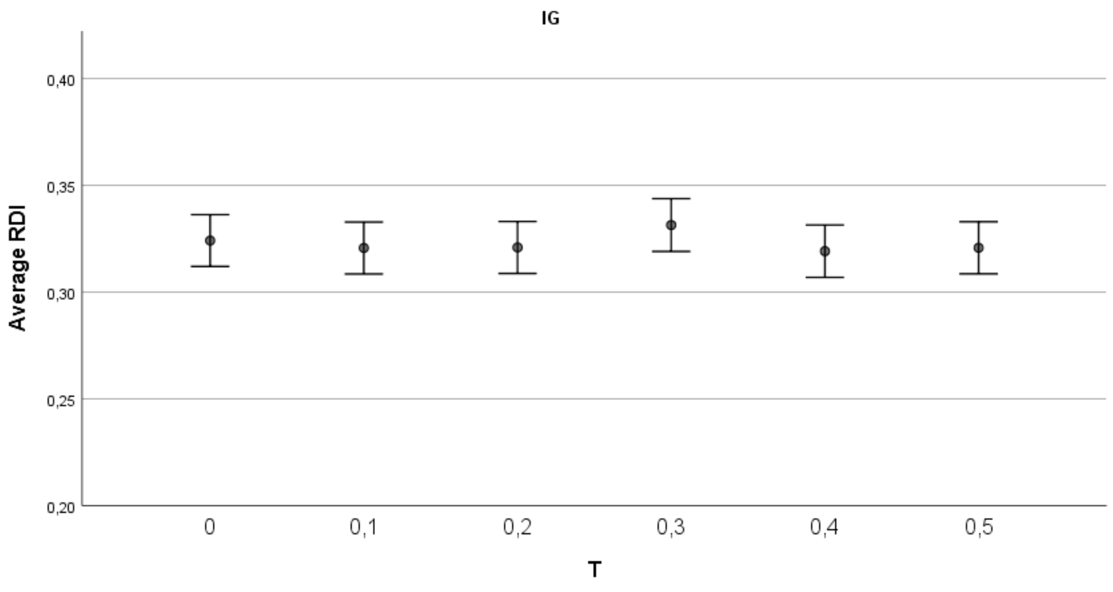

Figure 5.9 95% Confidence Intervals for IG for different values of T.

Finally, the last parameter to tune is the number of jobs removed in the destruction phase. To calibrate the parameter d, we make the same process as for the parameter T. The values taken for calibration are $d = 2,3,4$. Attending to the Figure 5.10, we can state that for $d = 3$ we obtain the best results without significant differences.

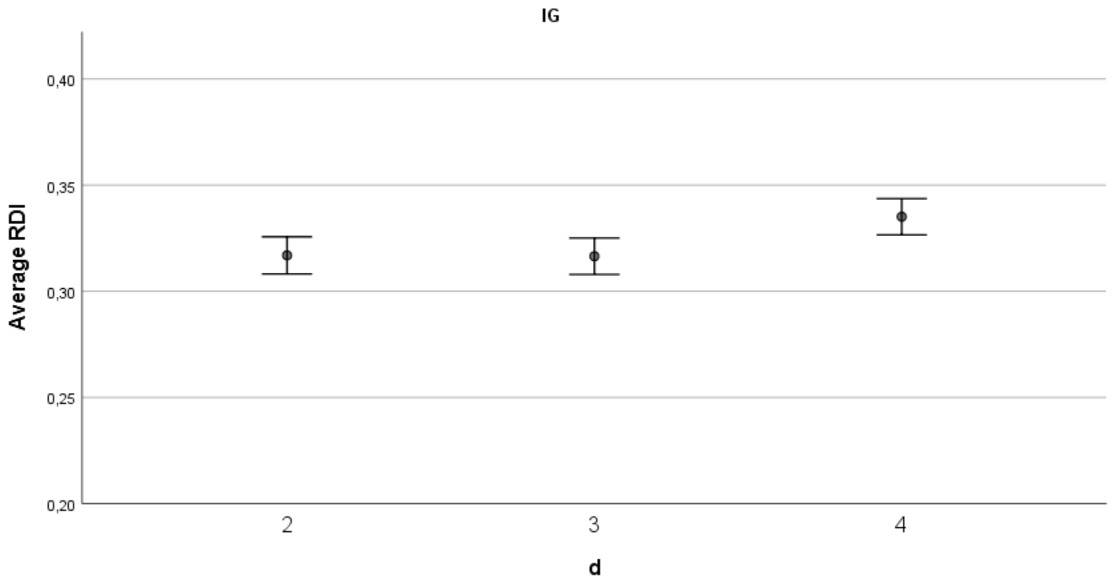

Figure 5.10 95% Confidence Intervals for IG for different values of d.

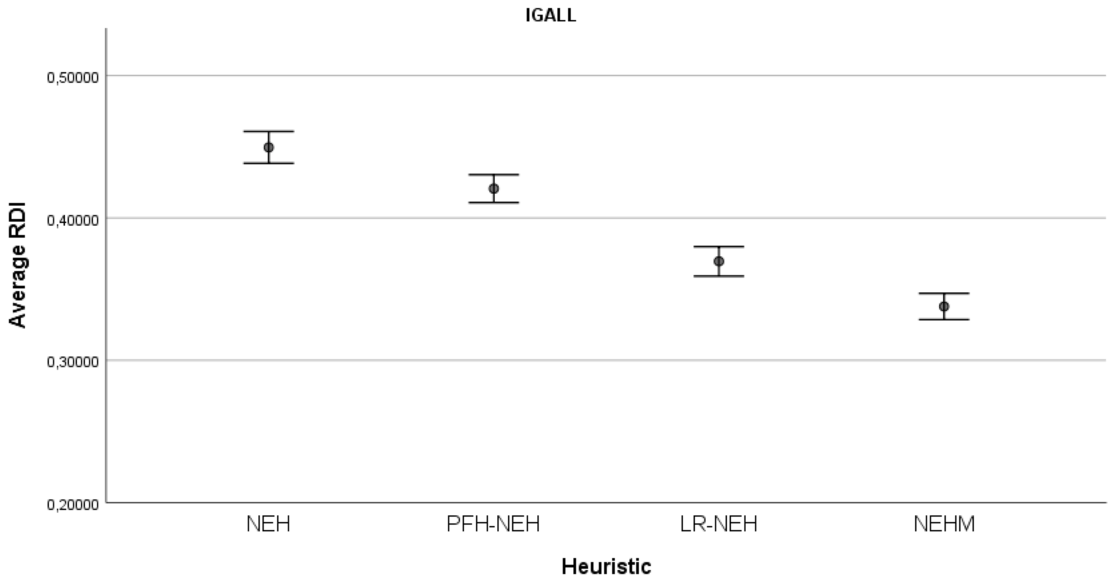

Figure 5.11 95% Confidence Intervals for IGALL for different Heuristics as the initial sequence.

5.5.2 IGALL

Summing up the procedure, it starts with a initial sequence and then the algorithm is applied.

All the experiments for the metaheuristics are carried out fixing a stopping criterion of the same maximum time, so efficiencies will be easy and fast to compare.
The parameters of which depends are:

- The initial sequence obtained by one of the heuristics adapted.
- The parameter T: The parameter of which depend the temperature proposed for the simulated annealing algorithm.
- The parameter d: The number of jobs removed in the destruction phase.

As we have already done for the IG, the first parameter to calibrate is the heuristic. The argument is the same as we stated previously, although NEH_M is the best heuristic and we have just decided to choose the parameter x like Case 1, we decided to compare again for one reason: The NEH_M is the best in terms of RPI but is the most CPU-heavy. We must check if the time invested in the initial sequence is beneficial to the final result. Taking each of the heuristics as a starting sequence, the first step is to compare all heuristics setting Case 1. The results in terms of RDI are shown in Figure A.2. As we can see, the plots demonstrate statistically significant differences between NEH_M and others.

Conclusion reflects that the more time invested in the initial sequence (NEH_M compare with LR-NEH), the better results we obtain. The NEH_M is chosen as the initial sequence.

The next step is to calibrate the parameter T. The values taken for calibration are $T = 0.1, 0.2, 0.3, 0.4, 0.5$ As we can see in 5.12 the best result is provided by the $T = 0.2$. Although we can state that $T = 0.2$ provides the best results, there are no significant differences.
Finally, the last parameter to tune is the number of jobs removed in the destruction phase. To calibrate the parameter d, we make the same process as for the parameter T. The values taken for calibration are $d = 2{,}3{,}4$. Attending to the Figure 5.13 we can state that for $d = 2$ we obtain the best results with significant differences with $d = 4$ but without significant differences with d=3.

5.5.3 VBIH

Summing up the procedure, it starts with a initial sequence an then the algorithm is applied. All the experiments for the metaheuristics are carried out fixing a stopping criterion of the same maximum time, so efficiencies will be easy and fast to compare.
The parameters of which depends are:

- The initial sequence obtained by one of the heuristics adapted.

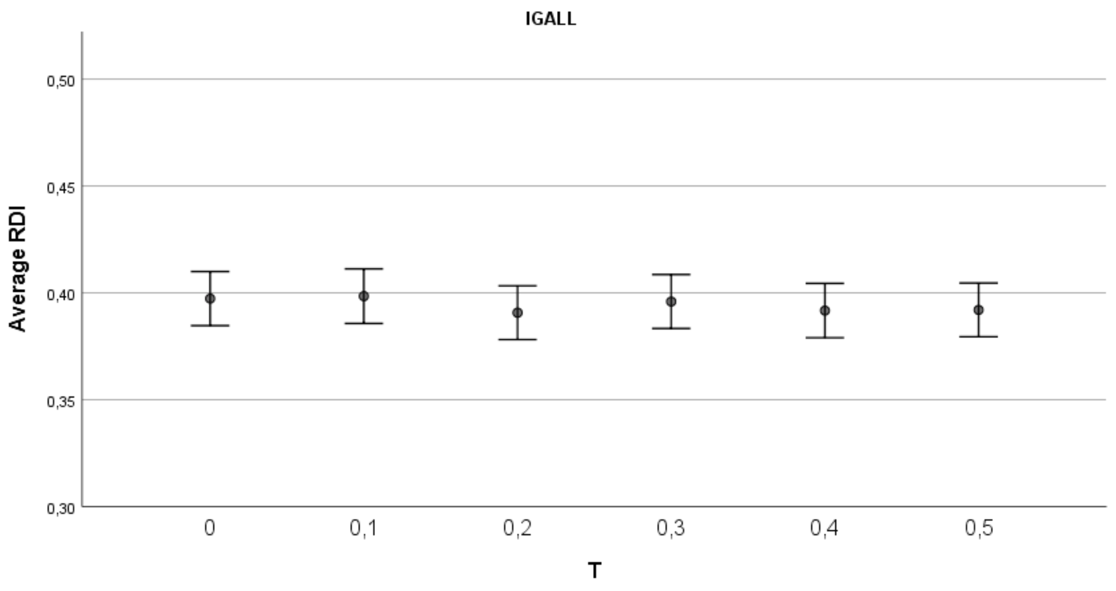

Figure 5.12 95% Confidence Intervals for IGALL for different values of T.

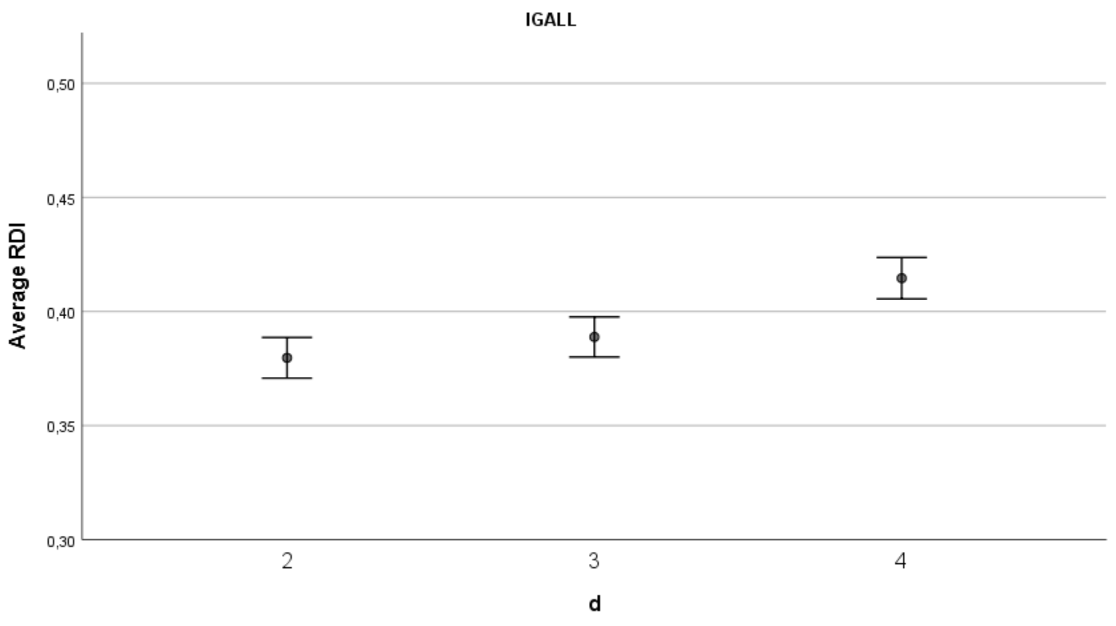

Figure 5.13 95% Confidence Intervals for IGALL for different values of d.

- The parameter T: The parameter of which depend the temperature proposed for the simulated annealing algorithm.
- The parameter b_min: The minimum size of the block of jobs removed in the destruction phase.
- The parameter b_max: The minimum size of the block of jobs removed in the destruction phase.

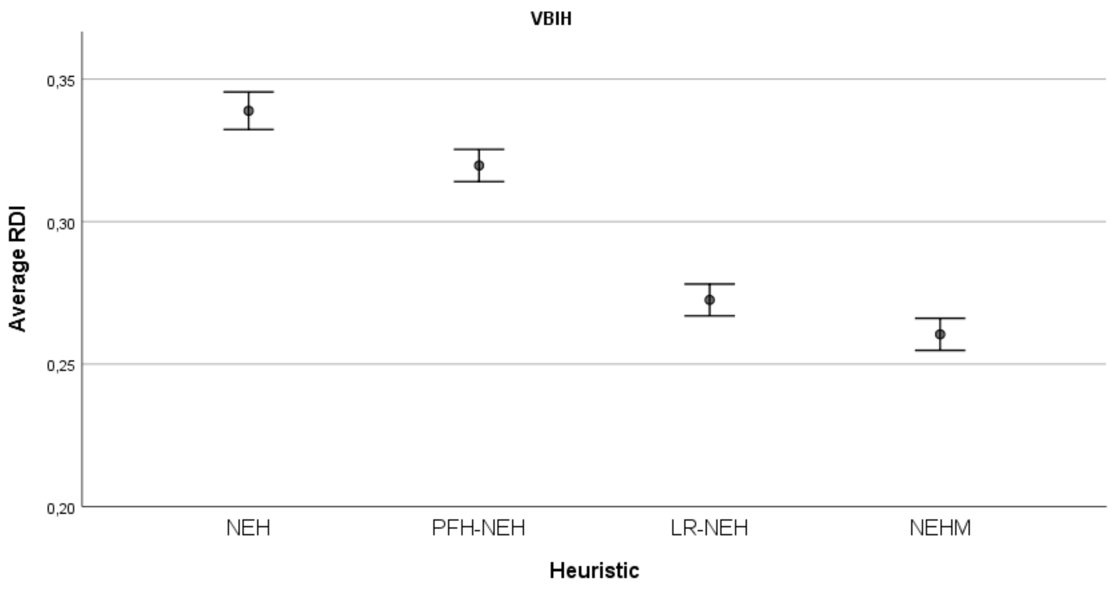

Figure 5.14 95% Confidence Intervals for VBIH for different Heuristics as the initial sequence.

As we have already done for the IG and IGALL, the first parameter to calibrate is the heuristic. The argument is the same as we stated previously.

Taking each of the heuristics as a starting sequence, the first step is to compare all heuristics setting Case 1. The results in terms of RDI are shown in Figure 5.14. As we can see, the plots demonstrate statistically significant differences between NEH$_M$ and others.

The initial sequence chosen is the NEH$_M$. The next step is to calibrate the parameter T. The values taken for calibration are $T = 0.1, 0.2, 0.3, 0.4, 0.5$. As we can see in 5.15 the best result is provided by the $T = 0.4$. Although we can state that $T = 0.5$ provides the best results, there are no significant differences.

To calibrate the parameter b_{min}, we make the same process as for the parameter T. The values taken for calibration are $b_{min} = 2,3$. Attending to the Figure 5.16 we can state that for $b_{min} = 1$ we obtain the best results with significant differences.

The last parameter is b_{max} and and the same process is carried out. The values taken for calibration are $b_{max} = 3,4,5,6$. Attending to the Figure 5.17 we can state that for $b_{max} = 6$ we obtain the best results with significant differences if we compare with $b_{max} = 3$ and 4, and without significant differences compared with $b_{max} = 5$.

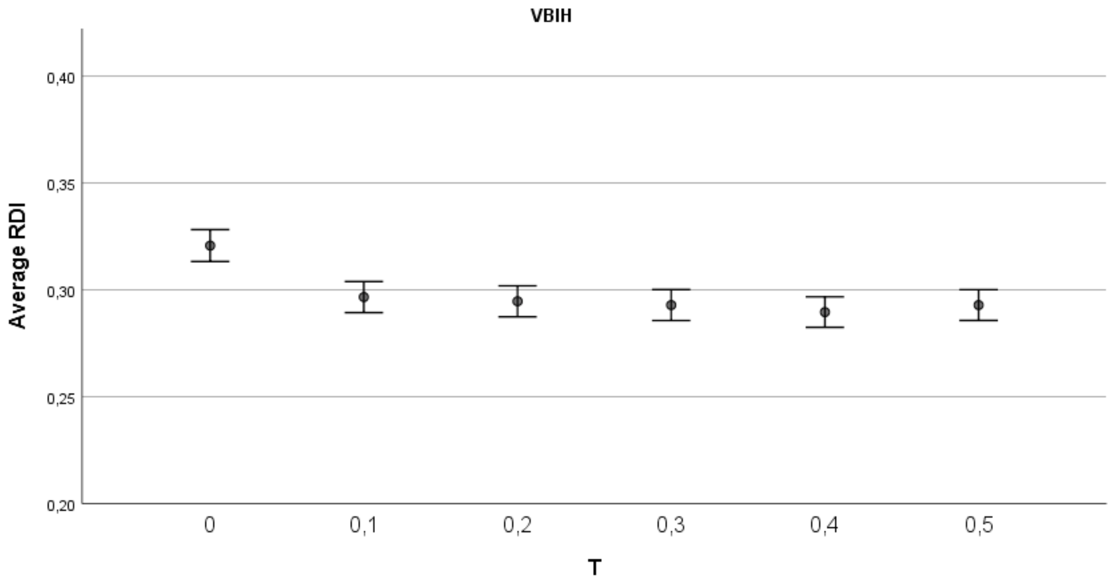

Figure 5.15 95% Confidence Intervals for VBIH for different values of T.

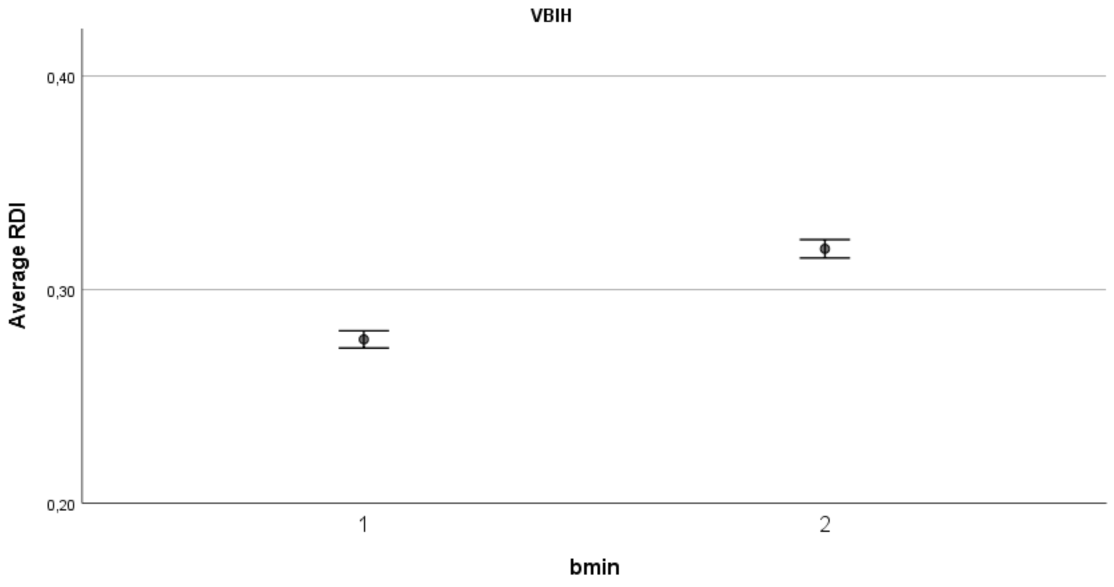

Figure 5.16 95% Confidence Intervals for VBIH for different values of b_{min}.

5.6 Metaheuristics comparison

The next section is dedicated to comparing all metaheuristics once the parameters were set. As we can see in Figure 5.18, the results indicate that the best method in terms of ARDI is the VBIH. The plots demonstrate statistically significant differences between VBIH and others.

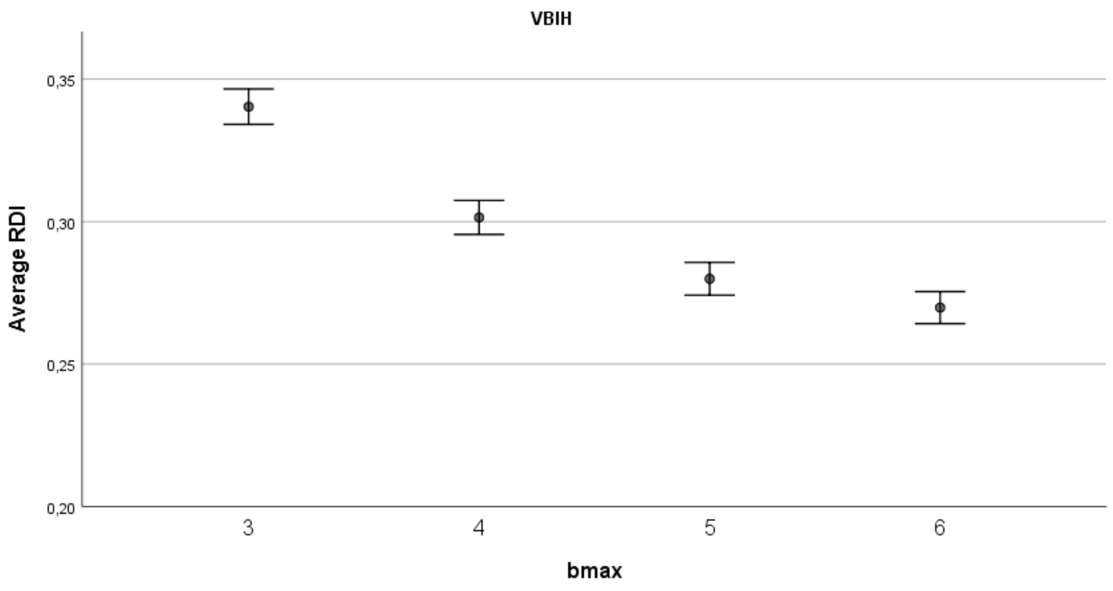

Figure 5.17 95% Confidence Intervals for VBIH for different values of b_{max}.

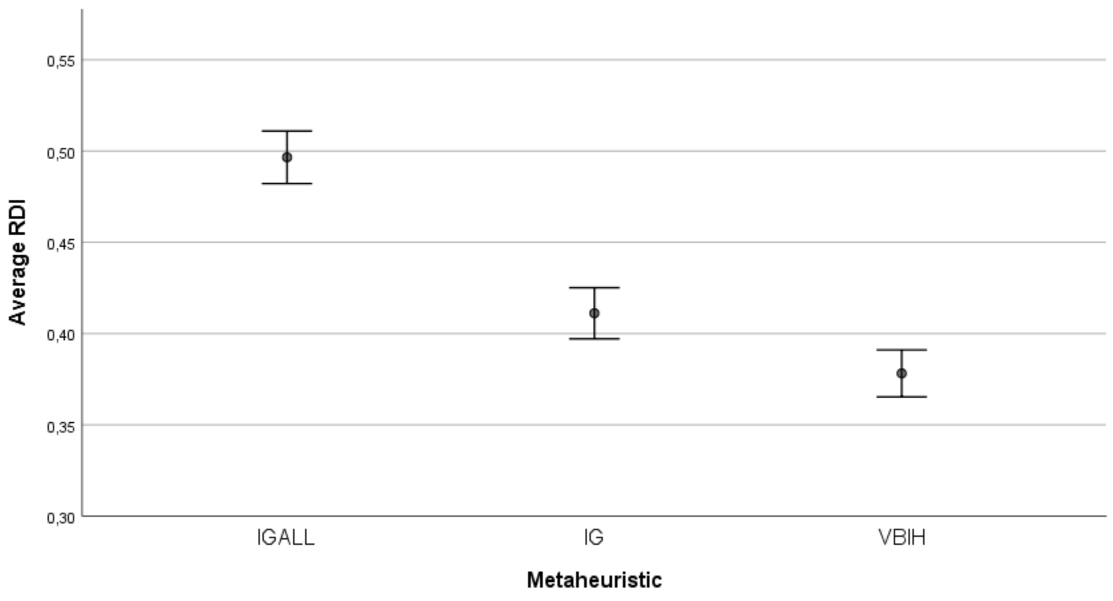

Figure 5.18 95% Confidence Intervals for metaheuristics.

When analysing the influence of the number of jobs (n) and the number of machines (m), we can draw the following conclusions.

As n increases worse results are obtained by VBIH. In fact, for $n > 200$ the best metaheuristic is the IG rather VBIH. The biggest difference between VBIH and the others is for $n < 200$, specifically for $n = 100$.

The same behavior is shown if we analysing the influence of the number of

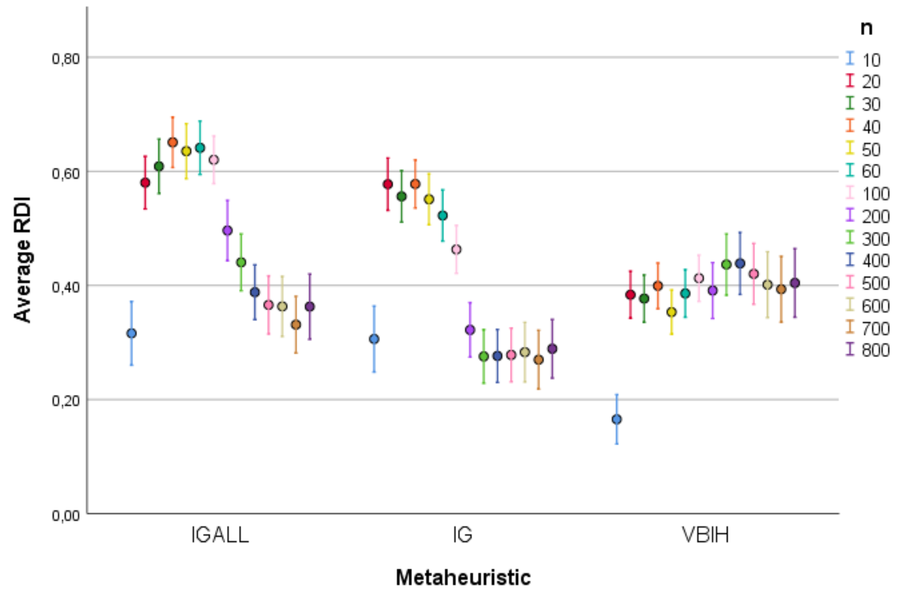

Figure 5.19 95% Confidence Intervals for metaheuristics for different values of n.

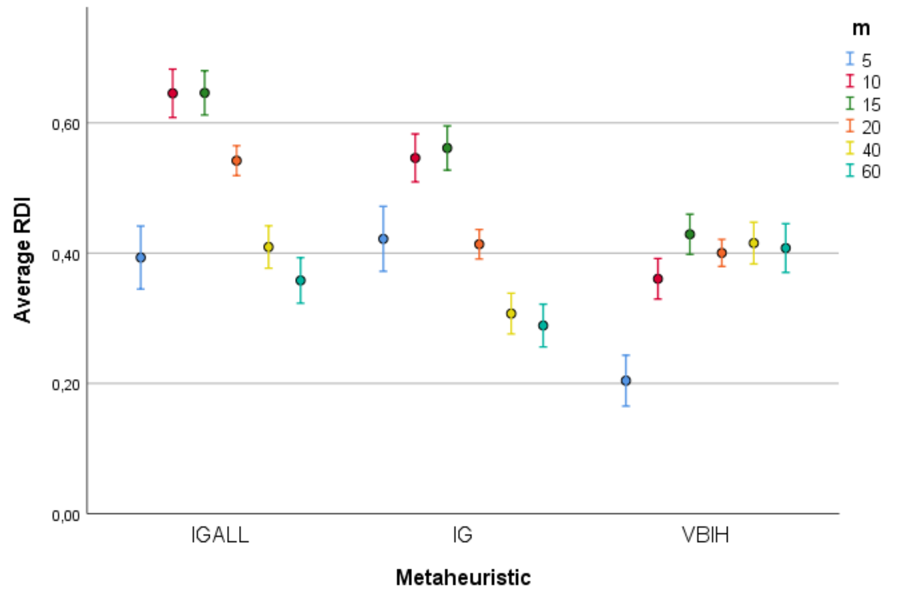

Figure 5.20 95% Confidence Intervals for metaheuristics for different values of m.

machines. The best results obtained by VBIH are for $m = 5$. As m increase wore results are obtained by VBIH. In fact, for $m = 40$ and $m = 60$ the best results are obtained by IG.

6 Conclusions

In this master thesis a set of heuristics, constructive heuristics and metaheuristics has been tested to the $Fm|prmu|TCIT$ problem. Approximate methods have not been considered previously to solve this problem. Therefore, in this master thesis the most efficient methods found in the related literature (PFSP with makespan and total completion time minimization) have been adapted. The calibration and comparison of method has been developed in two groups, the heuristics and metaheuristics. To be calibrate Taillard's benchmark [33] has been used. However, Vallad's benchmark [35] has been used for the comparison due to its statistical advantages. The heuristics and constructive heuristics has been NEH, NEH$_M$ (a modification of NEH), PFH-NEH and LR-NEH.About the calibration, the Case 1 is choosen for all the heuristics. The main reason of the selection is the balance between the improvement of the objective function and the time spends to achieve it. Case 1 is based on set the number of iteration $x = n$ if $n \leq 100$ and 1 if it exceeds 100. The great effort in CPU necessary for instances with a large number of jobs is evident. However, a great improvement in a short time is obtained for this small instances.On the other hand, once the heuristics are calibrated is time to compare. The NEH$_M$ provides the best results, but it is also the one that spends the highest quantity of computational time.

We can conclude that these heuristics provide a good results for our problem, but the implementation as initial sequence to the metaheuristic has been tested. We need to compare if this CPU effort is reflected in an improvement in the final solution. Another conclusion about the comparison of the heuristics selected is the great importance of the first job of the sequence in the objective function. All there are based on repeating a certain algorithm changing the first job, and has been shown the improvement in the objective function.

Related to the adaptation of existing metaheuristics for other problems in the literature the next conclusions has be made. First to compare all them, the calibration should be made. The first of this parameter is the heuristic implemented to obtain the initial sequence. It is demonstrate that the best heuristic to obtain the initial sequence is the NEH$_M$. The CPUE inverted due to repeated the algorithm changing the first job of the sequence improve the solution. It is another evidence of the impact of the first job in the solution. Then the others parameters are calibrate. About the comparison of the metaheuristic, the best results is provide by the VBIH. Specifically, the best results are provide for small instances, been the IG the best one for big instances. We can conclude that the results reveal a good performance of the method adapted.

To end, a future line of research could be to approach the minimization of the TCIT with the minimization of the energy consumption of the machines. Also, based on these results, methods with greater intelligence could be developed based on the objective developed.

Appendix A
Heuristics

A.1 Calibration

Table A.1 LR-NEH: ARDI for each case x.

		Cases			
n	m	1	2	3	4
20	5	0,48	0,48	0,48	0,48
	10	0,75	0,75	0,75	0,75
	20	0,98	0,98	0,98	0,98
	Average	0,74	0,74	0,74	0,74
50	5	0,50	0,50	0,50	0,50
	10	0,78	0,78	0,78	0,78
	20	0,95	0,95	0,95	0,95
	Average	0,74	0,74	0,74	0,74
100	5	0,41	0,41	0,41	0,41
	10	0,63	0,63	0,63	0,63
	20	0,96	0,96	0,96	0,96
	Average	0,67	0,67	0,67	0,67
200	10	0,66	0,22	0,35	0,22
	20	0,94	0,54	0,64	0,54
	Average	0,80	0,38	0,49	0,38
500	20	0,65	0,65	0,38	0,38
	Average	0,65	0,65	0,38	0,38
Average	5	0,46	0,46	0,46	0,46
	10	0,70	0,59	0,63	0,59
	20	0,90	0,82	0,78	0,76
	Average	0,73	0,66	0,65	0,63

Table A.2 NEHM: ARDI for each case x.

		Cases			
n	m	1	2	3	4
20	5	0,61	0,61	0,61	0,61
	10	0,57	0,57	0,57	0,57
	20	0,52	0,52	0,52	0,52
	Average	0,57	0,57	0,57	0,57
50	5	0,43	0,43	0,43	0,43
	10	0,54	0,54	0,54	0,54
	20	0,52	0,52	0,52	0,52
	Average	0,50	0,50	0,50	0,50
100	5	0,29	0,29	0,29	0,29
	10	0,43	0,43	0,43	0,43
	20	0,51	0,51	0,51	0,51
	Average	0,41	0,41	0,41	0,41
200	10	0,37	0,17	0,24	0,17
	20	0,55	0,32	0,37	0,32
	Average	0,46	0,24	0,31	0,24
500	20	0,65	0,65	0,45	0,45
	Average	0,65	0,65	0,45	0,45
Average	5	0,44	0,44	0,44	0,44
	10	0,48	0,43	0,45	0,43
	20	0,55	0,50	0,47	0,46
	Average	0,50	0,46	0,46	0,45

Table A.3 PFH-NEH: ARDI for each case x.

		Cases			
n	m	1	2	3	4
20	5	0,40	0,40	0,40	0,40
	10	0,85	0,85	0,85	0,85
	20	0,78	0,78	0,78	0,78
	Average	0,67	0,67	0,67	0,67
50	5	0,39	0,39	0,39	0,39
	10	0,86	0,86	0,86	0,86
	20	0,80	0,80	0,80	0,80
	Average	0,68	0,68	0,68	0,68
100	5	0,82	0,82	0,82	0,82
	10	0,73	0,73	0,73	0,73
	20	0,89	0,89	0,89	0,89
	Average	0,81	0,81	0,81	0,81
200	10	1,00	0,52	0,69	0,52
	20	0,89	0,51	0,69	0,51
	Average	0,95	0,51	0,69	0,51
500	20	0,98	0,98	0,67	0,67
	Average	0,98	0,98	0,67	0,67
Average	5	0,54	0,54	0,54	0,54
	10	0,86	0,74	0,78	0,74
	20	0,87	0,79	0,76	0,73
	Average	0,78	0,71	0,71	0,68

A.2 Comparison

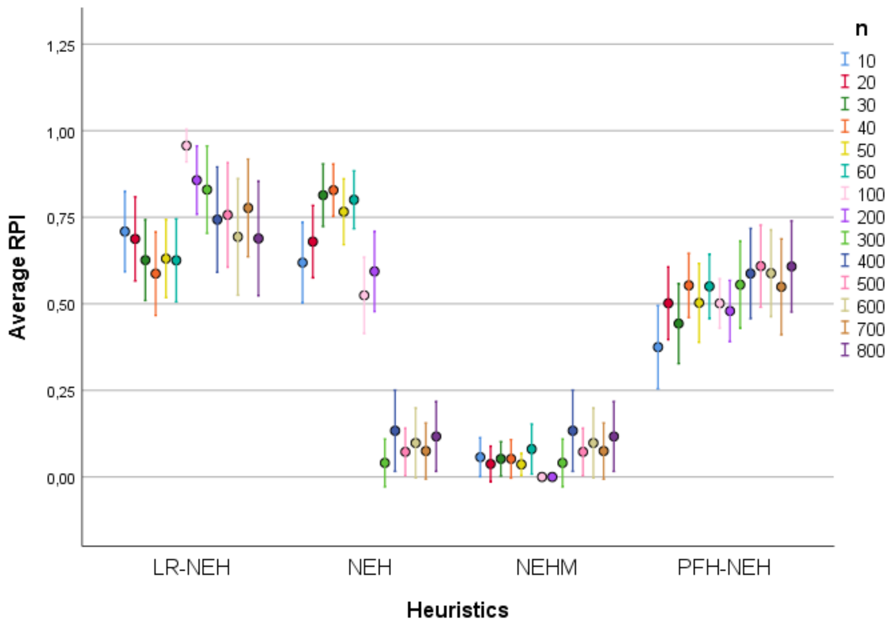

Figure A.1 95% Confidence Intervals for heuristics for different values of n.

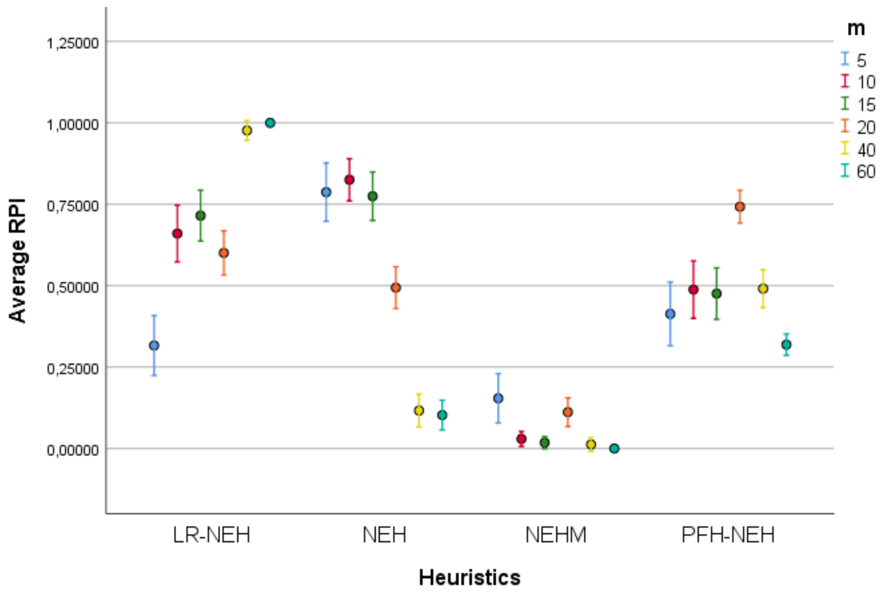

Figure A.2 95% Confidence Intervals for heuristics for different values of m.

Table A.4 Heuristics: ARDI for small instances for n and m.

n	m	LR-NEH	NEH	NEHM	PFH-NEH
10	5	0,49	0,52	0,10	0,30
	10	0,69	0,78	0,03	0,46
	15	0,80	0,60	0,06	0,38
	20	0,85	0,58	0,05	0,35
	Average	0,71	0,62	0,06	0,37
20	5	0,23	0,91	0,15	0,29
	10	0,85	0,76	0,00	0,59
	15	0,81	0,49	0,00	0,54
	20	0,87	0,56	0,00	0,59
	Average	0,69	0,68	0,04	0,50
30	5	0,20	0,89	0,16	0,37
	10	0,66	0,88	0,05	0,40
	15	0,73	0,90	0,00	0,53
	20	0,90	0,59	0,00	0,48
	Average	0,63	0,81	0,05	0,44
40	5	0,39	0,79	0,16	0,47
	10	0,44	0,92	0,05	0,51
	15	0,63	0,91	0,00	0,54
	20	0,89	0,69	0,00	0,69
	Average	0,59	0,83	0,05	0,55
50	5	0,34	0,65	0,05	0,57
	10	0,71	0,77	0,04	0,47
	15	0,62	0,92	0,05	0,36
	20	0,85	0,71	0,00	0,60
	Average	0,63	0,77	0,04	0,50
60	5	0,24	0,96	0,31	0,47
	10	0,61	0,84	0,01	0,49
	15	0,70	0,82	0,00	0,50
	20	0,95	0,58	0,00	0,73
	Average	0,63	0,80	0,08	0,55
Average	5	0,32	0,79	0,15	0,41
	10	0,66	0,83	0,03	0,49
	15	0,72	0,77	0,02	0,48

Table A.5 Heuristics: ARDI for big instances for n and m.

n	m	LR-NEH	NEH	NEHM	PFH-NEH
100	20	0,87	0,79	0,00	0,66
	40	1,00	0,37	0,00	0,43
	60	1,00	0,41	0,00	0,42
	Average	0,96	0,52	0,00	0,50
200	20	0,57	0,91	0,00	0,67
	40	1,00	0,46	0,00	0,45
	60	1,00	0,41	0,00	0,32
	Average	0,86	0,59	0,00	0,48
300	20	0,50	0,12	0,12	0,90
	40	0,99	0,00	0,00	0,51
	60	1,00	0,00	0,00	0,26
	Average	0,83	0,04	0,04	0,56
400	20	0,23	0,40	0,40	0,88
	40	1,00	0,00	0,00	0,60
	60	1,00	0,00	0,00	0,28
	Average	0,74	0,13	0,13	0,59
500	20	0,27	0,22	0,22	0,93
	40	1,00	0,00	0,00	0,52
	60	1,00	0,00	0,00	0,37
	Average	0,76	0,07	0,07	0,61
600	20	0,08	0,29	0,29	0,96
	40	1,00	0,00	0,00	0,54
	60	1,00	0,00	0,00	0,26
	Average	0,69	0,10	0,10	0,59
700	20	0,33	0,13	0,13	0,98
	40	1,00	0,10	0,10	0,34
	60	1,00	0,00	0,00	0,33
	Average	0,78	0,07	0,07	0,55
800	20	0,24	0,35	0,35	0,97
	40	0,83	0,00	0,00	0,54
	60	1,00	0,00	0,00	0,31
	Average	0,69	0,12	0,12	0,61
Average	20	0,60	0,49	0,11	0,74
	40	0,98	0,12	0,01	0,49
	60	1,00	0,10	0,00	0,32

Appendix B
Metaheuristics

B.1 Calibration

Table B.1 IG: ARDI for different values of d.

			d	
n	m	2	3	4
20	5	0,33	0,33	0,33
	10	0,47	0,46	0,48
	20	0,46	0,44	0,40
	Average	0,42	0,41	0,40
50	5	0,30	0,27	0,28
	10	0,36	0,37	0,42
	20	0,39	0,38	0,40
	Average	0,35	0,34	0,37
100	5	0,33	0,33	0,34
	10	0,26	0,27	0,30
	20	0,33	0,32	0,36
	Average	0,30	0,31	0,33
200	10	0,12	0,13	0,16
	20	0,20	0,22	0,27
	Average	0,16	0,18	0,21
500	20	0,27	0,27	0,27
	Average	0,27	0,27	0,27
Average	5	0,32	0,31	0,32
	10	0,30	0,31	0,34
	20	0,33	0,33	0,34
	Average	0,32	0,31	0,33

Table B.2 IG: ARDI for different heuristic as initial sequence.

		Heuristic			
n	m	LR-NEH	NEH	NEHM	PFH-NEH
20	5	0,25	0,50	0,40	0,16
	10	0,41	0,55	0,39	0,52
	20	0,43	0,47	0,35	0,47
	Average	0,37	0,50	0,38	0,39
50	5	0,23	0,34	0,32	0,23
	10	0,41	0,42	0,31	0,41
	20	0,40	0,46	0,33	0,37
	Average	0,35	0,41	0,32	0,34
100	5	0,07	0,56	0,17	0,51
	10	0,19	0,39	0,23	0,30
	20	0,39	0,34	0,25	0,36
	Average	0,22	0,43	0,22	0,39
200	10	0,12	0,12	0,11	0,18
	20	0,27	0,19	0,20	0,25
	Average	0,20	0,16	0,15	0,22
500	20	0,23	0,28	0,28	0,29
	Average	0,23	0,28	0,28	0,29
Average	5	0,19	0,47	0,30	0,30
	10	0,28	0,37	0,26	0,35
	20	0,35	0,35	0,28	0,35
	Average	0,29	0,38	0,28	0,34

Table B.3 IG: ARDI for different values of T.

n	m	0.0	0.1	0.2	0.3	0.4	0.5
20	5	0,33	0,31	0,32	0,34	0,32	0,34
	10	0,46	0,47	0,45	0,48	0,48	0,47
	20	0,44	0,42	0,42	0,44	0,43	0,43
	Average	0,41	0,40	0,40	0,42	0,41	0,41
50	5	0,29	0,30	0,28	0,29	0,28	0,26
	10	0,38	0,39	0,37	0,40	0,39	0,39
	20	0,38	0,38	0,39	0,40	0,39	0,40
	Average	0,35	0,36	0,34	0,36	0,35	0,35
100	5	0,36	0,34	0,32	0,33	0,31	0,31
	10	0,28	0,28	0,27	0,28	0,27	0,27
	20	0,33	0,33	0,36	0,33	0,33	0,34
	Average	0,32	0,31	0,32	0,31	0,30	0,31
200	10	0,13	0,14	0,14	0,14	0,13	0,12
	20	0,23	0,22	0,23	0,23	0,22	0,23
	Average	0,18	0,18	0,18	0,19	0,18	0,18
500	20	0,27	0,27	0,27	0,29	0,25	0,27
	Average	0,27	0,27	0,27	0,29	0,25	0,27
Average	5	0,33	0,32	0,30	0,32	0,30	0,30
	10	0,31	0,32	0,31	0,32	0,32	0,31
	20	0,33	0,32	0,33	0,34	0,32	0,34
	Average	0,32	0,32	0,32	0,33	0,32	0,32

B.2 Comparison

Table B.4 IGALL: ARDI for different values of d.

		d		
n	m	2	3	4
20	5	0,34	0,34	0,36
	10	0,51	0,48	0,55
	20	0,44	0,45	0,45
	Average	0,43	0,42	0,45
50	5	0,29	0,28	0,28
	10	0,48	0,49	0,54
	20	0,46	0,49	0,50
	Average	0,41	0,42	0,44
100	5	0,32	0,32	0,32
	10	0,32	0,34	0,41
	20	0,46	0,50	0,54
	Average	0,37	0,39	0,42
200	10	0,19	0,22	0,24
	20	0,34	0,36	0,38
	Average	0,27	0,29	0,31
500	20	0,35	0,34	0,37
	Average	0,35	0,34	0,37
Average	5	0,32	0,31	0,32
	10	0,38	0,39	0,43
	20	0,41	0,43	0,45
	Average	0,38	0,39	0,41

Table B.5 IGALL: ARDI for different heuristic as initial sequence.

		Heuristic			
n	m	LR-NEH	NEH	NEHM	PFH-NEH
20	5	0,29	0,50	0,40	0,20
	10	0,46	0,59	0,45	0,55
	20	0,46	0,48	0,35	0,49
	Average	0,40	0,52	0,40	0,41
50	5	0,24	0,33	0,32	0,24
	10	0,53	0,57	0,40	0,52
	20	0,53	0,55	0,38	0,48
	Average	0,43	0,48	0,37	0,42
100	5	0,10	0,48	0,18	0,51
	10	0,26	0,51	0,27	0,40
	20	0,58	0,50	0,38	0,55
	Average	0,31	0,50	0,28	0,48
200	10	0,24	0,20	0,14	0,30
	20	0,42	0,34	0,26	0,41
	Average	0,33	0,27	0,20	0,36
500	20	0,31	0,35	0,36	0,40
	Average	0,31	0,35	0,36	0,40
Average	5	0,21	0,44	0,30	0,32
	10	0,37	0,47	0,32	0,44
	20	0,46	0,44	0,35	0,47
	Average	0,37	0,45	0,33	0,42

Table B.6 IGALL: ARDI for different values of T.

		T					
n	m	0.0	0.1	0.2	0.3	0.4	0.5
20	5	0,34	0,35	0,38	0,36	0,34	0,31
	10	0,50	0,52	0,52	0,50	0,52	0,51
	20	0,44	0,46	0,45	0,45	0,44	0,43
	Average	0,43	0,44	0,45	0,44	0,43	0,42
50	5	0,29	0,26	0,29	0,29	0,29	0,28
	10	0,49	0,51	0,51	0,51	0,49	0,52
	20	0,50	0,50	0,47	0,49	0,48	0,48
	Average	0,43	0,42	0,42	0,43	0,42	0,43
100	5	0,33	0,31	0,30	0,32	0,31	0,33
	10	0,35	0,37	0,36	0,36	0,36	0,36
	20	0,51	0,50	0,49	0,50	0,51	0,51
	Average	0,40	0,39	0,38	0,39	0,39	0,40
200	10	0,23	0,21	0,22	0,22	0,21	0,22
	20	0,36	0,37	0,35	0,36	0,36	0,35
	Average	0,29	0,29	0,29	0,29	0,28	0,29
500	20	0,36	0,36	0,34	0,34	0,35	0,37
	Average	0,36	0,36	0,34	0,34	0,35	0,37
Average	5	0,32	0,31	0,32	0,32	0,31	0,31
	10	0,39	0,40	0,40	0,40	0,39	0,40
	20	0,43	0,44	0,42	0,43	0,43	0,43
	Average	0,39	0,39	0,39	0,39	0,39	0,39

Table B.7 VBIH: ARDI for different values of bmin.

		bmin	
n	m	1	2
20	5	0,23	0,26
	10	0,38	0,43
	20	0,35	0,39
	Average	0,32	0,36
50	5	0,22	0,23
	10	0,29	0,34
	20	0,34	0,39
	Average	0,28	0,32
100	5	0,28	0,29
	10	0,21	0,25
	20	0,29	0,36
	Average	0,26	0,30
200	10	0,10	0,14
	20	0,23	0,29
	Average	0,17	0,21
500	20	0,37	0,42
	Average	0,37	0,42
Average	5	0,24	0,26
	10	0,25	0,29
	20	0,32	0,37
	Average	0,27	0,32

Table B.8 VBIH: ARDI for different values of bmax.

		bmax			
n	m	3	4	5	6
20	5	0,31	0,26	0,23	0,20
	10	0,47	0,42	0,38	0,36
	20	0,42	0,38	0,35	0,35
	Average	0,40	0,35	0,32	0,30
50	5	0,26	0,23	0,20	0,20
	10	0,38	0,32	0,30	0,28
	20	0,40	0,37	0,34	0,33
	Average	0,35	0,31	0,28	0,27
100	5	0,31	0,28	0,28	0,27
	10	0,27	0,24	0,21	0,21
	20	0,38	0,32	0,30	0,29
	Average	0,32	0,28	0,26	0,26
200	10	0,15	0,12	0,11	0,10
	20	0,30	0,26	0,25	0,24
	Average	0,22	0,19	0,18	0,17
500	20	0,40	0,39	0,40	0,40
	Average	0,40	0,39	0,40	0,40
Average	5	0,29	0,26	0,24	0,22
	10	0,32	0,27	0,25	0,24
	20	0,38	0,34	0,33	0,32
	Average	0,34	0,30	0,28	0,27

Table B.9 Heuristics: ARDI for small instances for n and m.

n	m	IG	IGALL	VBIH
10	5	0,15	0,17	0,07
	10	0,34	0,39	0,23
	15	0,40	0,38	0,17
	20	0,33	0,32	0,19
	Average	0,31	0,32	0,17
20	5	0,44	0,36	0,21
	10	0,69	0,72	0,46
	15	0,57	0,61	0,43
	20	0,61	0,63	0,44
	Average	0,58	0,58	0,38
30	5	0,37	0,36	0,09
	10	0,60	0,71	0,48
	15	0,64	0,72	0,49
	20	0,62	0,64	0,46
	Average	0,56	0,61	0,38
40	5	0,56	0,51	0,30
	10	0,58	0,73	0,36
	15	0,61	0,71	0,52
	20	0,56	0,66	0,42
	Average	0,58	0,65	0,40
50	5	0,53	0,49	0,23
	10	0,54	0,69	0,33
	15	0,58	0,70	0,47
	20	0,55	0,67	0,39
	Average	0,55	0,64	0,35
60	5	0,48	0,47	0,32
	10	0,53	0,64	0,31
	15	0,57	0,77	0,50
	20	0,51	0,70	0,41
	Average	0,52	0,64	0,39
Average	5	0,42	0,39	0,20
	10	0,55	0,65	0,36
	15	0,56	0,65	0,43

Table B.10 Heuristics: ARDI for big instances for n and m.

n	m	IG	IGALL	VBIH
100	20	0,41	0,69	0,35
	40	0,51	0,61	0,40
	60	0,47	0,56	0,49
	Average	0,46	0,62	0,41
200	20	0,34	0,58	0,28
	40	0,32	0,52	0,41
	60	0,30	0,39	0,48
	Average	0,32	0,50	0,39
300	20	0,29	0,58	0,40
	40	0,29	0,40	0,47
	60	0,25	0,35	0,44
	Average	0,28	0,44	0,44
400	20	0,30	0,44	0,42
	40	0,29	0,40	0,48
	60	0,24	0,33	0,41
	Average	0,28	0,39	0,44
500	20	0,32	0,44	0,46
	40	0,25	0,34	0,42
	60	0,26	0,32	0,38
	Average	0,28	0,37	0,42
600	20	0,32	0,42	0,45
	40	0,28	0,35	0,39
	60	0,25	0,32	0,37
	Average	0,28	0,36	0,40
700	20	0,27	0,35	0,44
	40	0,26	0,35	0,38
	60	0,28	0,30	0,36
	Average	0,27	0,33	0,39
800	20	0,37	0,46	0,52
	40	0,24	0,32	0,36
	60	0,26	0,31	0,33
	Average	0,29	0,36	0,40
Average	20	0,41	0,54	0,40
	40	0,31	0,41	0,42
	60	0,29	0,36	0,41

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Arianna Alfieri, Michele Garraffa, Erica Pastore, and Fabio Salassa, *Permutation flowshop problems minimizing core waiting time and core idle time*, Computers Industrial Engineering **176** (2023), 108983.

[2] A. Baskar and M. Anthony Xavior, *New idle time-based tie-breaking rules in heuristics for the permutation flowshop scheduling problems*, Computers & Operations Research **133** (2021), 105348.

[3] Maria Raquel C. Costa, Jorge M.S. Valente, and Jeffrey E. Schaller, *Efficient procedures for the weighted squared tardiness permutation flowshop scheduling problem*, Flexible Services and Manufacturing Journal **32** (2020), no. 3, 487–522.

[4] Alex Paranahyba de Abreu and Helio Yochihiro Fuchigami, *An efficiency and robustness analysis of warm-start mathematical models for idle and waiting times optimization in the flow shop*, Computers Industrial Engineering **166** (2022), 107976.

[5] Sanchez de-los Reyes Paula, Perez-Gonzalez Paz, and M Framinan Jose, *Permutation flowshop scheduling problem with total core idle time minimization*, IFAC-PapersOnLine **55** (2022), no. 10, 187–191, 10th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2022.

[6] B. Dhanasakkaravarthi and A. Krishnamoorthy, *A New Priority Rule for Initial Ordering of Jobs in Permutation Flowshop Scheduling Problems*, International Journal of Advanced Computer Science and Applications **13** (2022), no. 1, 2022.

[7] Jérémie Dubois-Lacoste, Federico Pagnozzi, and Thomas Stützle, *An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem*, Computers and Operations Research **81** (2017), 160–166.

[8] Victor Fernandez-Viagas and Jose M. Framinan, *A new set of high-performing heuristics to minimise flowtime in permutation flowshops*, Computers Operations Research **53** (2015), 68–80.

[9] Victor Fernandez-Viagas and Jose M. Framinan, *Reduction of permutation flowshop problems to single machine problems using machine dominance relations*, Computers & Operations Research **77** (2017), 96–110.

[10] ______, *A best-of-breed iterated greedy for the permutation flowshop scheduling problem with makespan objective*, Computers and Operations Research **112** (2019).

[11] Victor Fernandez-Viagas, Jose M. Molina-Pariente, and Jose M. Framinan, *Generalised accelerations for insertion-based heuristics in permutation flowshop scheduling*, European Journal of Operational Research **282** (2020), no. 3, 858–872.

[12] Victor Fernandez-Viagas, Rubén Ruiz, and Jose M. Framinan, *A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation*, European Journal of Operational Research **257** (2017), no. 3, 707–721.

[13] Victor Fernandez-Viagas, Jorge M.S. Valente, and Jose M. Framinan, *Iterated-greedy-based algorithms with beam search initialization for the permutation flowshop to minimise total tardiness*, Expert Systems with Applications **94** (2018), 58–69.

[14] J. Fondrevelle, A. Oulamara, and M.-C. Portmann, *Permutation flowshop scheduling problems with time lags to minimize the weighted sum of machine completion times*, International Journal of Production Economics **112** (2008), no. 1, 168–176, Special Section on Recent Developments in the Design, Control, Planning and Scheduling of Productive Systems.

[15] Jose M. Framinan, Rainer Leisten, and Chandrasekharan Rajendran, *Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem*, International Journal of Production Research **41** (2003), no. 1, 121–148.

[16] Jose M. Framinan, Rainer Leisten, and Rubén Ruiz García, *Manufacturing Scheduling Systems*, 2014.

[17] Kannan Govindan, R. Balasundaram, N. Baskar, and P. Asokan, *A hybrid approach for minimizing makespan in permutation flowshop scheduling*, Journal of Systems Science and Systems Engineering 2017 26:1 **26** (2017), no. 1, 50–76.

[18] Johnny C. Ho and Jatinder N.D. Gupta, *Flowshop scheduling with dominant machines*, Computers Operations Research **22** (1995), no. 2, 237–246.

[19] S. M. Johnson, *Optimal two- and three-stage production schedules with setup times included*, Naval Research Logistics Quarterly **1** (1954), no. 1, 61–68.

[20] Jin Pin Liou, *Dominance conditions determination based on machine idle times for the permutation flowshop scheduling problem*, Computers & Operations Research **122** (2020), 104964.

[21] Jiyin Liu and Colin R Reeves, *Constructive and composite heuristic solutions to the P||Ci scheduling problem*, European Journal of Operational Research **132** (2001), no. 2, 439–452.

[22] Kathrin Maassen, Paz Perez-Gonzalez, and Lisa C. Günther, *Relationship between common objective functions, idle time and waiting time in permutation flow shop scheduling*, Computers & Operations Research (2020), 104965.

[23] S.T. McCormick, Michael L. Pinedo, S. Shenker, and B. Wolf, *Sequencing in an assembly line with blocking to minimize cycle time*, Operations Research (1989).

[24] Márcia de Fátima Morais, Matheus Henrique Dal Molin Ribeiro, Ramon Gomes da Silva, Viviana Cocco Mariani, and Leandro dos Santos Coelho, *Discrete differential evolution metaheuristics for permutation flow shop scheduling problems*, Computers & Industrial Engineering **166** (2022), 107956.

[25] M. Nawaz, E.E. Enscore, and I. Ham, *A Heuristic Algorithm for the m-Machine , n-Job Flow-shop Sequencing Problem*, Omega **11** (1982), no. 1, 91–95.

[26] IH Osman and CN Potts, *Simulated annealing for permutation flow-shop scheduling*, Omega **17** (1989), no. 6, 551–557.

[27] Hande Öztop, M. Fatih Tasgetiren, Deniz Türsel Eliiyi, Quan-Ke Pan, and Levent Kandiller, *An energy-efficient permutation flowshop scheduling problem*, Expert Systems with Applications **150** (2020), 113279.

[28] Hande Öztop, M. Fatih Tasgetiren, Levent Kandiller, Deniz Türsel Eliiyi, and Liang Gao, *Ensemble of metaheuristics for energy-efficient hybrid flowshops: Makespan versus total energy consumption*, Swarm and Evolutionary Computation **54** (2020), no. September 2019.

[29] Quan Ke Pan and Rubén Ruiz, *A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime*, Computers & Operations Research **40** (2013), no. 1, 117–128.

[30] Rubén Ruiz and Thomas Stützle, *A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem*, European Journal of Operational Research **177** (2007), no. 3, 2033–2049.

[31] Alex J. Ruiz-Torres, Johnny C. Ho, and José H. Ablanedo-Rosas, *Makespan and workstation utilization minimization in a flowshop with operations flexibility*, Omega **39** (2011), no. 3, 273–282.

[32] Andreia F. Silva, Jorge M.S. Valente, and Jeffrey E. Schaller, *Metaheuristics for the permutation flowshop problem with a weighted quadratic tardiness objective*, Computers & Operations Research **140** (2022), 105691.

[33] E. D. Taillard, *Benchmarks for basic scheduling problems*, European Journal of Operational Research **64** (1993), 278–285.

[34] Mehmet Fatih Tasgetiren, Quan-Ke Pan, Damla Kizilay, and Kaizhou Gao, *A Variable Block Insertion Heuristic for the Blocking Flowshop Scheduling Problem with Total Flowtime Criterion*, Algorithms **9** (2016), no. 4, 71.

[35] Eva Vallada, Rubén Ruiz, and Jose M. Framinan, *New hard benchmark for flowshop scheduling problems minimising makespan*, European Journal of Operational Research **240** (2015), no. 3, 666–677.

[36] Betul Yagmahan and Mehmet Mutlu Yenisey, *Ant colony optimization for multi-objective flow shop scheduling problem*, Computers Industrial Engineering **54** (2008), no. 3, 411–420.