Ph. D. Thesis

The Permutation Flowshop Scheduling Problem: Analysis, Solution Procedures and Problem Extensions

Victor Fernandez-Viagas

supervised by
Prof. Jose Manuel Framiñán Torres

University of Seville

To the memory of Angeles Rubio Sánchez.

Acknowledgements

I want to start the acknowledgements with the main person whose help has made this thesis possible, my advisor Jose M. Framinan. His supervision has been the perfect mix between knowledge and motivation. He has taught me not only how to make good and professional research, but also essential aspects that have contributed to finish my academical formation in several aspects of my life. I really appreciate all the time that he has spent on me and all the help that he selflessly has given to me.

Furthermore, several other persons have undoubtedly helped me to achieve the knowledge needed to finish this thesis. Among them, I want to highlight Ruben Ruiz, Rainer Leisten and Jorge M. S. Valente for hosting me for research stays and also for helping me to improve the quality of this dissertation. Their comments have been an excellent orientation for me to perform this thesis.

Thirdly, I am sincerely grateful to the Industrial Management Research Group of the University of Seville and to the Department of Applied Statistics and Operational Research of the Polytechnic University of Valencia for providing me all the equipment used in this thesis and for making available around 50 computers for all experimentations included in this Thesis.

I also want to be thankful with the great support of my colleagues in the Department of Industrial Organisation and Business Management of the University of Seville. Although each of them has been really important to carry out this Thesis, I want to be specially grateful to the encourage, and the research and methodological help of Manuel Dios, Roberto Dominguez, Jose M. Molina, Jose L. Andrade and Paz Perez.

Finally, I am deeply grateful to Pura, Placido, Rosa, Pachi, Sarah, Luis, Gonzalo and Acosta for their high motivation, support and help they gave to me along these years.

Contents

Part I

PRELIMINARIES

Chapter 1

Introduction

During the past twenty five years, following the massive use of internet and the EU single Market, European manufacturing companies struggle in a more competitive market, where firms from different countries must fight for common customers. As a consequence, prices of the products have decreased and the efficiency in the production processes of the companies have become more and more important. Nowadays, this fact is also increasing due to the competition from companies in developing countries whose labor cost is substantially lower. Therefore, production management is a key element for companies to survive. Production management involves decision making over several issues such as master scheduling, material requirements planning, capacity planning, manufacturing scheduling, ... Among these decisions, manufacturing scheduling plays an essential role on resource productivity and customer service. Its role is also increasing in many service industries as transportation, computer and communications industries, which are moving towards manufacture-to-order and virtual environments.

Manufacturing scheduling deals with the determination of the jobs which are processed for each resource in each instant of time, i.e. establishes the schedules of the resources along the horizon under consideration. In order to determine the best schedule for the shop floor, both the specific constraints and the goal of the shop have to be considered. In these environments, the difficulty of the scheduling problem increases and becomes NP-hard even for the most simple scheduling problems, being extremely complex for real manufacturing scenarios. Additionally, scheduling decisions should be made in short time intervals requiring a rapid response time, due to several aspects such as the lifetime of a schedule, the delay in the suppliers, arrivals of new jobs to be processed, rescheduling due to failures while processing a job, ... All these issues strongly stress the need to find fast and efficient solution procedures (i.e. heuristics and metaheuristics) for solving manufacturing scheduling problems.

In practice, several processing layouts have been adopted by companies to manufacture their products.

Among them, the Permutation Flowshop Scheduling Problem (PFSP in the following), which is the problem addressed in this Thesis, stands out as the most relevant, being one of the most studied problems in Operations Research. There are several reasons for this fact: On the one hand, the flow shop layout is the common configuration in many real manufacturing scenarios, as it presents several advantages over more general job shop configuration, and, in addition, many job shops are indeed a flow shop for most of the jobs. On the other hand, many models and solution procedures for different constraints and layouts have their origins in the flowshop scheduling problem, which increases the importance to find efficient algorithms for this scheduling problem.

Despite the huge number of research conducted on the PFSP, we believe that there is room for improving the current state of the art in the topic by

1. deepening the understanding of the problem with respect to their input parameters,
2. devising new approximate solution procedures for the common employed objectives, and
3. addressing problem extensions to capture more realistic situations.

1.1 Objectives and outline of the Thesis

As stated in the previous section, the goal of this Thesis tries to provide further insights into the PFSP, both deeply analysing the influence of the different input parameters and developing new efficient techniques to solve the problem as well as some problem extensions. To carry out this goal, the following general research objectives are identified:

GO1. To review the PFSP literature for the most common objectives, i.e. makespan, total completion time and due-date-based objectives (total tardiness, and total earliness and tardiness).

GO2. To analyse the influence of the processing times and due dates of the jobs on the PFSP.

GO3. To provide schedulers with faster and more efficient heuristics and metaheuristics to solve the PFSP for makespan, total completion time, total tardiness, and total earliness and tardiness minimisation.

GO4. To demonstrate the efficiency and good performance of the solution procedures developed in GO3.

GO5. To extend the proposals in Goal 3 to some constrained PFSP based on real manufacturing environments.

To achieve these objectives, the Thesis have been structured in five parts as follows:

- Part I is divided into two chapters. In Chapter 1.1, we introduce this Thesis and discuss its main contributions. In Chapter 2, the problem under consideration is stated. The measures to compare approximated algorithms are discussed in Chapter 3. There, the benchmarks used to evaluated the algorithms are introduced and an alternative indicator is proposed to overcome some problems detected using the traditional ones.

- In Part II, we analyse the problem in detail along three chapters. Dealing with Objective GO1, the main contributions in the literature are review for the most-common objective functions in Chapter 4. Additionally, in Chapter 5, we extensively study the behaviour of the problem depending on the configuration of the shops, i.e. processing times and due dates of the jobs (see GO2).

- In Part III, we propose new novelties efficient algorithms to solve the PFSP under several objectives. The procedures, constructive and improvement heuristics and metaheuristics, exploit the specific structure of the problem to both reduce the computational times of them and improve the quality of the solutions. Additionally, they are validated in extensive computational evaluations, comparing them with the state-of-the-art algorithms under the same conditions. More specifically, this part is divided in four chapters and addresses the general research objectives GO3 and GO4. Firstly, a new tie-breaking mechanism to minimise makespan, which can be incorporated in the two most efficient algorithms for the problem, is proposed in Chapter 6. In Chapter 7, two efficient constructive heuristics are proposed to minimise total flowtime. Several tie-breaking mechanisms are proposed and compared to minimise total tardiness in Chapter 8. Finally, four procedures to minimise total earliness and tardiness are proposed in Chapter 9.

- In Part IV, focused in more real manufacturing environment. New constraints are added to the traditional problem as well as different consideration and interaction between factories are taken into account. The proposed environments are solved using efficient approximate methods taken into consideration ideas of the traditional PFSP. More specifically, an iterated non-population algorithm to minimise makespan subject to a maximum tardiness is proposed in Chapter 10. In the Chapter 11, we add the blocking constraints to the traditional PFSP. These constraints take into consideration limited buffers between the machines. This problem, of permutation nature, is solved by means of an efficient beam-search-based constructive heuristic trying to minimise the total completion time. In the Chapter 12, we consider the parallel flowshop scheduling problem also denoted as distributed PFSP where several identical flowshop or even flowshop factories are available in parallel to assign the jobs. The problem is solved using a bounded-search iterated greedy algorithm

- Finally, in Part V, the conclusions of this research and future research lines are discussed.

Chapter 2

Problem statement

2.1 Problem description and notation

The problem under consideration is the permutation flowshop scheduling problem to minimise a certain objective function. The problem consists of the determination of the sequence of n jobs which achieves the minimal objective function value when all jobs are processed (in the order indicated by the sequence) on the m machines of a shop. The following additional hypotheses are usually assumed for the PFSP:

- Processing times are known and deterministic.
- No preemption is allowed.
- Release times are set to 0.
- Sequence-dependent set-up times are considered insignificant.
- Sequence-independent setup times are considered as non-anticipatory and therefore, can be added to the processing time of the jobs on the machines.
- Transportation times can be considered either insignificant or constant.
- Each job can be processed by at most one machine at the same time.
- Each machine can process only one job at the same time.
- Unlimited in-process inventory is considered.
- All machines are available on the whole scheduling horizon.

The notation of the problem can be stated as follows: on each machine i, each job j has a processing time denoted as t_{ij}. Given a sequence of jobs $\Pi := (\pi_1, \dots \pi_n)$, let us denote $p_{ij}(\Pi)$ the processing time of job π_j on machine i, i.e. $p_{ij}(\Pi) = t_{i\pi_j}$. Whenever it does not lead to confusion, this notation is abbreviated to p_{ij}. Analogously, $C_{i\pi_j}(\Pi)$ (abbreviated to C_{ij} whenever it does not lead to confusion) denotes the completion time of job π_j on machine i, whereas $C_{i[j]}$ indicates the completion time of the job scheduled in position j on machine i. $C_{i\pi_j}$ can be calculated in the following recursive manner:

$$C_{i\pi_j} = max\{C_{i-1,\pi_j}, C_{i,\pi_{j-1}}\} + t_{i\pi_j} \tag{2.1}$$

where $C_{0\pi_j} = C_{i0} = 0$.

Then, the completion time of job π_j on the last machine of the shop, i.e. $C_{m\pi_j}$, is denoted as C_{π_j} for simplicity. Analogously, the makespan or maximum completion time, $C_{m\pi_n}$, can be denoted as C_{max}.

Let d_{π_j} be the due date of job π_j, and $t_{\pi_j} = \sum_{i=1}^{m} t_{i\pi_j}$ the sum of the processing times of job π_j across all machines. The tardiness (earliness) of job π_j is defined as $T_{\pi_j} = max\{C_{m\pi_j} - d_{\pi_j}, 0\}$ ($E_{\pi_j} = max\{d_{\pi_j} - C_{m\pi_j}, 0\}$) and the maximum tardiness (earliness) as $T_{max} = \max_{j=1,\dots,n}\{T_{\pi_j}\}$ ($E_{max} = \max_{j=1,\dots,n}\{E_{\pi_j}\}$).

Different criteria have been considered in the literature for the described scheduling problem (see e.g. the reviews by [49, 199, 137]). Without any hesitation, the most common ones are the maximum completion time among the jobs or makespan, the total flowtime (sum of completion times of all jobs), the total tardiness (sum of the tardiness of each job), and total earliness and tardiness.

Makespan and total completion time are related to the fast processing of the products and to a balanced use of resources, both issues being of great importance in make-to-stock manufacturing scenarios. The minimization of makespan, $C_{\max}$, (also denoted as maximum completion time or maximum flowtime) has been commonly chosen by researchers as the objective to optimize in the PFSP (e.g. see [95], [197], [117], [45] or [188] for other objectives in the PFSP). Regarding the minimisation of the sum of the completion times of the jobs (or equivalently mean completion time), it has been consistently pointed out both as relevant and meaningful for today's dynamic production environment [108]. Under the assumption of a zero release time for the jobs, the minimization of total (average) completion time is equivalent to total (average) flowtime minimisation, which leads to stable or even use of resources, a rapid turn-around of jobs and the minimisation of in-process inventory [152].

In contrast, total tardiness and total earliness and tardiness focus on the satisfaction of customers and it is therefore better suited for make-to-order manufacturing scenarios as due dates play a key role [142, 89]. Particularly, the total tardiness highlights a critical concern for manufacturing systems (see e.g.

[156, 141]), since delays may lead to an increase in costs such as penalty clauses in a contract, loss of customers and/or bad reputation for other customers [180]. Regarding the just-in-time based-on objective, i.e. the minimisation of total earliness and tardiness, it aims to reduce the complexity of detailed material planning, the need for shop-floor control, the work-in-process and final inventories, and the transactions associated with shop-floor and purchasing systems, since both the excess of inventory in the shop and the delays on the due dates should be avoided([202]).

In this Thesis, we focus on these objectives to solve the PFSP. They can be defined as:

- Minimisation of makespan: $\min C_{max}$.
- Minimisation of total flowtime: $\min \sum_{\forall j} C_j$ or, equivalently, $\min \sum C_j$.
- Minimisation of total tardiness: $\min \sum T_j = \min \sum_{\forall j} T_j$.
- Minimisation of total earliness and tardiness $\sum_{\forall j} E_j + \sum_{\forall j} T_j$ or, equivalently, $\min \sum E_j + T_j$.

Regarding the notation for the PFSP analysed in this Thesis, we adopt the well-known classification based on the triplet $\alpha/\beta/\gamma$ (see e.g. a detailed description of this classification in [47]). In this classification, α indicates the machines layout (e.g. 1 for single machine, Pm for m identical parallel machines, Fm for a flow shop with m machines, Jm for job shops,...), β shows the constraints of the problem (e.g. *prmu* for permutation problems, *prec* for precedence relationships,...), and finally γ defines the objective function. Thereby, the problems under study are denoted along this Thesis as:

- PFSP to minimise makespan: $Fm|prmu|C_{\max}$ (see [58]).
- PFSP to minimise total flowtime: $Fm|prmu|\sum C_j$ (see [58]).
- PFSP to minimise total tardiness: $Fm|prmu|\sum T_j$ (see [146])
- PFSP to minimise total earliness and tardiness: $Fm|prmu|\sum E_j + \sum T_j$ (see [146])

2.2 Taillard's accelerations

Taillard [189] proposed a very fast mechanism (denoted as Taillard's accelerations) to evaluate sequences in insertion phases of the algorithms, which is explained in detail in this section. This mechanism, originally proposed for the $Fm|prmu|C_{\max}$, is at the core of the excellent performance of the state-of-the-art algorithms.

Firstly, let us assume that a partial schedule of $k-1$ jobs has been constructed. An unscheduled job r (whose processing time in machine i is denoted by t_{ir}) is to be inserted in position l ($l = 1 \ldots k$), thus obtaining k partial sequences of k jobs denoting $\pi(l)$ the sequence when the unscheduled job is inserted in position l. Additionally, let us denote e_{ij} the earliest completion time of job π_j in machine i before inserting the unscheduled job. e_{ij} can be calculated as follows:

$$e_{ij} = max\{e_{i,j-1}, e_{i-1,j}\} + p_{ij}, i = 1 \ldots m, j = 1 \ldots k-1 \tag{2.2}$$

with $e_{0j} = 0$, and $e_{i0} = 0$. Similarly, q_{ij} the duration between the starting time of job π_j on machine i (before inserting the unscheduled job) and the end of all operations can be calculated according to the following expression:

$$q_{ij} = max\{q_{i+1,j}, q_{i,j+1}\} + p_{ij}, i = m \ldots 1, j = k-1 \ldots 1 \tag{2.3}$$

with $q_{m+1,j} = 0$, and $q_{i,k} = 0$.

One possibility to calculate the makespan of each of these k sequences is to use equation (2.1) for each sequence, which results in a complexity $O(n^2m)$. [189] proposed a mechanism based on equations (2.2) and (2.3) to reduce this complexity to $O(nm)$: Since the earliest completion times of the jobs in π prior to position l have not changed, then f_{il} the completion time on machine i of job inserted in position l can be computed in the following manner:

$$f_{il} = \max\{e_{i,l-1}, f_{i-1,l}\} + t_{ir}, i = 1 \ldots m \tag{2.4}$$

with $f_{0l} = 0$. Therefore, $C_{max}(l)$ the completion time of sequence $\pi(l)$ is:

$$C_{max}(l) = \max_{i=1 \ldots m} \{f_{il} + q_{il}\} \tag{2.5}$$

These accelerations can be used in each insertion phase of the heuristics and metaheuristics for the $Fm|prmu|C_{\max}$. Although this mechanism was originally proposed for that problem, it can be extended to some other related manufacturing scheduling problems (such as e.g. $DF|prmu|C_{max}$).

Chapter 3

Evaluation of algorithms

3.1 Introduction

Since the PFSP has been proved to be NP-complete if the number of machines is higher than two for all the objectives considered in this Thesis, most of the contributions have focused on providing approximate methods yielding good (but nor necessarily optimal) solutions in reasonable time. Addressing the general objective GO3, several approximate algorithms are also proposed in this Thesis in Parts III and IV. To obtain conclusions about the efficiency of these approximated methods, they should always be compared on large instances which consider several size of the problems. The benchmarks used to compare them with the state-of-the-art algorithms are described in Section 3.2. Additionally, the value of the objective function of the algorithms should be evaluated in instances with different size (i.e. with different values of n and m). The indicators to evaluate the algorithms in each instance are described in Section 3.3. Additionally, a new indicator is also introduced there in order to have non-instances-dependent indicators. Finally, once the instances and the indicators are set, approximate methods should be compared under the same conditions. In this Thesis, the procedure followed for it is explained in Section 3.4.

3.2 Benchmarks

To perform the comparisons between the proposed and the state-of-the-art algorithms, several sets of instances are generated in this Thesis. Note that, as established below in Section 4, there are different set of instances in the literature for the problems under consideration. Without any hesitation, the set of instances proposed by [190] are the most common ones for the $Fm|prmu|C_{\max}$ and the $Fm|prmu|\sum C_j$. Recently, [198] propose a more extensive and exhaustive benchmark for the $Fm|prmu|C_{\max}$. Regarding the due-dated-based scheduling problems addressed in this Thesis, i.e. the $Fm|prmu|\sum T_j$ and the $Fm|prmu|\sum E_j + \sum T_j$, algorithms are tested typically on the set of instances proposed by [199]. These three benchmarks are used along this Thesis and can be described as follows:

- Benchmark $\mathcal{B}_1$ ([190]), which includes 120 instances with 12 different sizes of instance combining the values $n \in [20, 50, 100, 200, 500]$ and $m \in [5, 10, 20]$. For each instance size, 10 difficult instances are constructed. Processing times are uniformly distributed from 1 to 99 in this testbed.

- Benchmark $\mathcal{B}_2$ (large instances of [198], denoted there as VRF_hard_large instances). This benchmark contains 240 instances for all the combinations of the parameters $n \in [100, 200, 300, 400, 500, 600, 700, 800]$ and $m \in [20, 40, 60]$. Processing times are uniformly distributed from 1 to 99. For each combination of n and m, 2,000 instances are generated and the hardest 10 are chosen to form this benchmark.

- Benchmark $\mathcal{B}_3$ ([199]). This benchmark is composed of a set of 540 instances and is the most extended benchmark for the PFSP with due dates. The benchmark is formed by 5 instances for each combination of $n = \{50, 150, 250, 350\}$, $m = \{10, 30, 50\}$, $T = \{0.2, 0.4, 0.6\}$ and $R = \{0.2, 0.6, 1.0\}$, where T and R are parameters related to the mean and standard deviation of the due dates respectively. These due dates are generated using the procedure described by [147], i.e. following a uniform distribution between $P \cdot (1 - T - R/2)$ and $P \cdot (1 - T + R/2)$, where P is a lower bound for the makespan. Processing times are generated using a uniform distribution [1, 99]. This set of instances are available in http://soa.iti.es.

Additionally, in this Thesis, different sets of instances are presented to evaluate the algorithms and to calibrate the parameters in order to avoid an over calibration of the parameters of the proposed algorithms. The calibration sets are:

- Calibration benchmark $\mathcal{B}_{C1}$. Five instances have been generated for several values of n and m, $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$, where the processing times of each job in each machine are uniformly distributed between 1 and 99.

- Calibration benchmark $\mathcal{B}_{C2}$. It is composed of 340 instances generated following the procedure described by [174]. This benchmark consists of 68 combinations of the parameters of n and m. More specifically, $n = \{20, 50, 80, ..., 410, 440, 470, 500\}$ and $m = \{5, 10, 15, 20\}$. Processing times are uniformly distributed between 1 and 99, and 5 instances are generated for each combination of n and m.

- Calibration benchmark $\mathcal{B}_{C3}$. It is generated according to the procedure by [197]. The number of jobs and machines is set to $n = \{50, 150, 250, 350\}$, $m = \{10, 30, 50\}$ and due dates are generated according to the procedure employed by [147] using an uniform distribution between $P\cdot(1-T-R/2)$ and $P\cdot(1-T+R/2)$. Parameters T and P take the following values in the test: $T = \{0.2, 0.4, 0.6\}$ and $R = \{0.2, 0.6, 1.0\}$. Additionally, processing times are generated according to a uniform distribution between 1 and 99. For each combination of parameters n, m, T and R, two instances are generated summing 216 instances.

3.3 Indicators

For approximate algorithms, there is a trade-off between the quality of the solutions and the time required by the algorithm to obtain them. Therefore, both aspects should be weighted when selecting one algorithm among the set of algorithms available for the problem. When facing an specific scheduling case, different decision intervals may be required, and different quality of the solution can be accepted. Consequently, in most cases there is no *a priori* knowledge of the precise trade-off required by the Decision Maker. Then, the idea of representing the algorithms along the two important criteria (quality of solutions and computational requirements) and excluding the dominated algorithms allows providing the Decision Maker with the set of Pareto-efficient algorithms so he/she can select the most convenient for his/her specific case.

Note that, for a given algorithm, different measures can be devised both for its quality of the solutions and for its computational requirements. However, these are mostly measured by the Average Relative Percentage Deviation ($ARPD1$ or $ARPD2$) and by the average CPU time in seconds, respectively. The $ARPD1$ ($ARPD2$) of algorithm h (out of a total of H algorithm) is obtained by averaging $RPD1_{ih}$ ($RPD2_{ih}$) the Relative Percentage Deviation of algorithm h in instance i over all instances a testbed:

$$RPD1_{ih} = \frac{O^{ih} - \min_{1\leq h\leq H} O^{ih}}{\min_{1\leq h\leq H} O^{ih}} \cdot 100 \tag{3.1}$$

$$RPD2_{ih} = \frac{O^{ih} - UB}{UB} \cdot 100 \tag{3.2}$$

where O^{ih} is the objective function obtained by algorithm h when applied to instance i and UB is the upper bound (best solution known) for that instance.

However, the usual indicator of the quality of algorithm h with respect to total tardiness when applied to a given instance i is the so-called Relative Deviation Index (RDI), which is defined as follows:

$$RDI_{ih} = \frac{sumT_{ih} - Best_i}{Worst_i - Best_i} \cdot 100$$

where $sumT_{ih}$ is the total tardiness obtained by algorithm h when applied to instance i. $Worst_i$ and $Best_i$ are the worst and best known total tardiness for instance j. RDI is usually employed for tardiness instead of the average relative deviation (most used indicator for makespan or flowtime objectives) since tardiness may yield 0 for some instances and therefore the value of the average relative deviation would be distorted (see [199, 88, 90]).

Regarding a indicator to measure the computational effort, it is commonly evaluated by the researcher using the average CPU time (denoted as $ACPU$, Expression (3.3)), which obtained by averaging the CPU times required by algorithm h for all instances.

$$ACPU_h = \sum_{i=1}^{I} T_{ih}/I \tag{3.3}$$

where T_{ih} is the computation time of algorithm h when applied to instance i, and I the number of instances of the testbed.

Using the indicators of the quality of the solutions ($ARPD1$, $ARPD2$ and $ARDI$) and the average CPU times ($ACPU$) in the Pareto set presents a number of issues. $ARPD1$, $ARPD2$ and $ARDI$ are dimensionless indicators that are normalised with respect to the best result obtained for each instance, therefore the influence of the instance (and thus the instance size) is somewhat smoothed. In contrast, CPU times are heavily instance and instance-size dependent. Moreover, given the problem sizes of the most typical benchmarks (see e.g. [190] and [198]), average CPU times of a algorithm are heavily compromised by the CPU times obtained for the biggest 10 instances (those of size 500×20).

To illustrate this shortcoming, let us consider the NEH. This algorithm is known to have a complexity of $O(n^3 \cdot m)$ for the problem under consideration, therefore for the smallest problem size of e.g. Taillard's testbed (20×5) its complexity is $O(10^2)$, whereas it is $O(1.6 \cdot 10^8)$ for size 200×20 and $O(2.5 \cdot 10^9)$ for size 500×20. This enormous differences in computation times imply that, using the CPU time data in [137], the average CPU time for the last 20 instances of Taillard's testbed is 0.36 seconds, while the average for all 120 instances is 0.37 seconds. As a consequence, more than 80% of the testbed (the first 100 instances out of a total of 120) contributes with 0.01 seconds (less than 5%) to the indicator.

Besides, the most used testbed for the problem (Taillard's testbed) is not orthogonal with respect to the number of machines and the number of jobs. More specifically, n ranges from 20 to 500, and m ranges from 5 to 20. Therefore, the CPU times required for one (hypothetical) heuristic with complexity $O(n^3m)$ would grow in this testbed much faster than that of another (hypothetical) heuristic with complexity $O(n^2m^2)$. This could compromise the average results, thus masking the efficiency –or inefficiency– of some algorithms.

In order to overcome these shortcomings, we propose an alternative measure to evaluate the efficiency of algorithms. More specifically, we propose the Average Relative Percentage computation Time ($ARPT1_h$). $ARPT1_h$ is defined as follows:

$$ARPT1_h = \sum_{i=1}^{I} \frac{RPT_{ih}}{I} \tag{3.4}$$

where

$$RPT_{ih} = \frac{T_{ih} - ACT_i}{ACT_i}$$

and

$$ACT_i = \sum_{h=1}^{H} T_{ih}/H$$

where RPT_{ih} is the relative percentage computation time obtained by heuristic h for instance i, H is the number of algorithms considered, and ACT_i is the average (among all algorithms considered) computational times for the instance i.

Additionally, let $ARPT2_h$ represent also the relative percentage computation time where 1 is added to the RPT_{ih} to avoid negative numbers.

$$ARPT2_h = \sum_{i=1}^{I} \frac{RPT_{ih}}{I} + 1 \tag{3.5}$$

3.4 Experimental conditions

In this Thesis a total of 17 approximated methods are proposed and compared with more than hundred algorithms. In order to have a fair comparison between the methods, all selected algorithms of this Thesis (i.e. the state-of-the-art algorithms and the proposed ones) are again fully re-coded in C# and tested under the same conditions which means:

- Using the same computer. This means same processor speed, bus speed, memory speed and size, etc.
- Using the same programming language (C# under Visual Studio 2013) and compiler.
- Using the same operating system.
- Using the same libraries and common functions.
- Using the same stopping criteria for the metaheuristics.

- Using the same set of instances in each comparison.

Additionally, to better fit the computational time of each heuristic, 5 runs are carried out for each instance and the average values of the indicators, for both computational effort and quality of the solutions, are collected.

Part II

ANALYSIS

Chapter 4

State of the art

4.1 Introduction

Literally hundreds of heuristics and metaheuristics in the last decades have proposed exact and approximate algorithms for the PFSP to find the best sequence of jobs according several criteria. Note that the division between heuristics and metaheuristics is ambiguous and different classifications have been proposed in the literature (see e.g. [223], [212]). Along this Thesis, we use the same division as in [172], where heuristics and metaheuristics are analysed separately. There, heuristics naturally stop when the procedure is finished whereas metaheuristics typically stop after a given number of iterations or elapsed CPU time. Regarding heuristics, they can also be divided into constructive heuristics and improvement heuristics ([172]). Constructive heuristics obtain the final sequence by appending jobs –usually in an iterative manner– to an incomplete sequence. Improvement heuristics are usually composed of two phases: a construction phase where a complete sequence is formed, and an improvement phase where the solution is improved by means of some method typically using specific knowledge of the problem.

Among the numerous algorithms in the literature, most research has focused in the minimisation of makespan and total flowtime (see e.g. the reviews by [43], [172] and [137]), although other objectives have been also considered (see e.g. [95] and [52] for the homogeneity of the completion times; [197] for total tardiness; [117] and [179] for total tardiness and earliness; or [188] and [45] for several objectives). In this section, we perform a comprehensive literature review for the problem (see Objective GO1). We focus in the following objectives:

- Minimisation of makespan
- Minimisation of total flowtime

- Minimisation of total tardiness
- Minimisation of total earliness and tardiness

The reviewed works –of computational/experimental nature– show that the PFSP is *empirically* hard, in the sense that the optimal or quasi-optimal sequences statistically represent a very small fraction of the space of feasible solutions, and that there are big differences among the corresponding objective function values. The rest of the chapter is organised as follows: In Section 4.2, we review heuristics and metaheuristics for the $Fm|prmu|C_{\max}$. The $Fm|prmu|\sum C_j$ is reviewed in Section 4.3. Finally, regarding due-dates-based objectives, algorithms are reviewed in Section 4.4.

4.2 Makespan

The permutation flowshop scheduling problem with makespan objective ($Fm|prmu|C_{\max}$) involves the determination of the order of processing of n jobs on m machines while all jobs have the same machine sequence. This problem is, without doubt, one of the most studied problems in Operations Research (see in this regard the reviews by [43, 160, 172]). Aside to the practical relevance of the problem, since the early work by [80], contributions on the $Fm|prmu|C_{\max}$ problem have pioneered the research in scheduling with different objectives and layouts.

The NP-hard nature of the problem for $m \geq 3$ (see [166]) has led the vast majority of research towards the proposal of approximate solutions (usually classified either as heuristics or metaheuristics). Traditionally divided into constructive and improvement types, heuristics have been extensively developed for the $Fm|prmu|C_{\max}$ either to yield a good solution in less CPU time or to find a seed sequence for metaheuristics. Since the publication of the work by [172], more than 100 new algorithms have been proposed in the literature over the last 10 years. Some of these methods –such as the iterated greedy (IG) of [174]– have improved the best existing algorithms in [172]. However, the new state-of-the-art algorithms remains unclear due to the lack of a homogeneous framework to conduct the comparison among algorithms. More specifically, the following problems can be detected:

- Many algorithms are compared under different conditions:
  - Tested under different computer conditions (different programming languages and/or different computers, operating systems, etc.).
  - Comparison of algorithms with different CPU time usages.
  - Use of different benchmarks (see Section 4.2).

- Many algorithms are compared in a non-conclusive way:
  - Lack of comparison against the state-of-the-art (e.g. without comparing with the iterated greedy proposed by [174]).
  - Among the several runs performed in each instance to increase the power of the results, the best runs are used instead of the average for some algorithms.
- New advances in the evaluation of the algorithms:
  - A more extensive benchmark of instances has been recently proposed by [198] (see Section3.2). This testbed can be used to establish statistical differences among algorithms in a sound way, differently from what can be done with older benchmarks (such as those by [190], [10], [158], [206], [30], [71]).
  - A new indicator has been proposed in Chapter 3.3 to measure the CPU requirements of the algorithms in relative terms. This indicator improves the deficiencies of the most common indicator (i.e. average CPU time) for the evaluation of efficient heuristics.

Among the constructive heuristics, which have been proposed in the literature, most of them are variants of the NEH heuristic by [127]. This heuristic consists of two phases:

1. First, jobs are ordered according to an initial order (decreasing sum of processing times).
2. The first job is removed from the initial order and placed in a partial sequence, initially without any job. Next, following this order, each job is removed and tried to be inserted in each possible position of the partial sequence. The position which minimizes the makespan is chosen for the job. The procedure is repeated n-1 times until all jobs are placed in the partial sequence.

Note that the complexity of the main heuristic for the problem, i.e. the NEH, is $O(n^3m)$, as the evaluation of an m-machine makespan can be accomplished in $O(nm)$ and the evaluation of the k subsequences resulting in step k can be completed in $O(n^2m)$. However, due to the Taillard's accelerations explained in Section 2.2, the evaluation of the k subsequences can be done in $O(nm)$ thus reducing the overall complexity of the heuristic to $O(n^2m)$.

If we consider the NEH heuristic as a particular case of a family of heuristics, there are several elements (options) within this family. These are:

- Starting order, i.e. how to obtain an initial order in which the jobs are arranged in the first phase.
- Sequence generation, i.e. how the candidate (sub)sequences are generated from the initial starting order.

- Tie-breaking mechanism, i.e. how ties are treated in the evaluation of the candidate (sub)sequences.

The starting order determines which job is to be picked for insertion in the current (sub)sequence. The original proposal by [127] is to arrange the jobs in descending order of the sum of their processing times. [42] conducted an extensive study with different initial orders and showed that there were significant differences among them and that, the original order remained the best for the makespan objective. These results were later confirmed by [83]. [125] proposed a different starting order based on an estimation of an idle time of the jobs. Although the authors claimed that their proposal outperforms the original NEH, an extensive simulation study carried out by [84] showed that this seems to be true only for $m < 6$ and that the resulting differences were not statistically significant. The latter authors also proposed an initial starting order which they claim to outperform the original one. [35] proposed a modification of the NEH heuristic in which a specific mechanism for tie-breaking is applied in addition to a starting order based on the mean and the variance of the processing times of the jobs and finally, [85] proposed a new modification of the classical. Although this last modification outperforms the original NEH (and the modification of the NEH proposed by [84] and [35], see [86]) in an extended test bed proposed by [85], it was not proved that the proposed starting order was superior to the original in the benchmark set of [190] where the starting order proposed by [35] presents the best results.

With respect to sequence generation, the original proposal is to insert the job in the k possible slots of the current sequence. However, it is clear that different strategies could be adopted, either by reducing the number of candidates (by e.g. evaluating just a fraction of the k possible slots), or by exploring more candidates. With respect to the former strategies, [151] limited the insertion to positions $\lfloor k/2 \rfloor$ to k with good results, while different strategies have been proposed for exploring more candidate solutions by [150] (note that other strategies have been explored for the total flowtime by [207] and by [44], but there is no proof that they are efficient with respect to makespan). In all these contributions, the gains (losses) in the quality of the solutions are compensated by the increase (decrease) in CPU time requirements.

Finally, with respect to the tie-breaking mechanism, modifications with respect to the original tie-breaking mechanism have been suggested by several authors. Note that, in general, tie-breaking mechanisms may refer either to the starting order (i.e. how to rank jobs with the same indicator value in the initial ordering sequence), or to the sequence generation phase (i.e. how to choose among different subsequences with the same best partial makespan). In this chapter we focus on the second type –labelled insertion tie-breaking in the following– so existing contributions will be discussed in detail in Section 6.2.

With or without the aforementioned modifications, NEH has turned out to be the most efficient heuristic found for the problem, and nowadays it remains the cornerstone of subsequent heuristics that have been proposed for the problem and that can be seen as refinements and/or enhancements of NEH. The reason for this efficiency probably lies in the procedure employed for inserting and evaluating –using Taillard's acceleration– the non-scheduled jobs, a mechanism also present in the Iterated Greedy Algorithm (denoted as IG_RS_{LS} in the following) proposed by [174] and considered among the best heuristics for the problem (see [174, 138]).

In this section, we propose a classification to unify the numerous variants of the NEH heuristic published in the literature. The classification use the following notation formed by 3 fields: $NEH(a|b|c)$ where the fields a, b and c are defined by:

- a: Initial order used by the NEH. The following sorting criteria have been considered in the literature:
  - rand: Jobs are randomly ordered. This order is used by [162] in RAER and RAER-di heuristics as comparison heuristics.
  - SD: Non decreasing sum of processing times (original order of the NEH) jobs. This order is used by the following heuristics: NEHR [162], NEHR-di[162], NEH [127], NEH-di [162], NEH1 [83] and NEH1-di [162].
  - AV: sum of the mean and deviation of the processing times (proposed by [35]).
  - NM: order proposed by [125] and used in NEMR and NEMR-di heuristics by [162].
  - KK1: Sorting criterion proposed by [84]. This initial order is applied in NEHKK1 [84] and NEHKK1-di [162] heuristics.
  - KK2: Sorting criterion proposed by [85] in NEHKK2 heuristic.
- b: Once a job is selected for insertion in all positions of a partial sequence, the same makespan can be obtained for several positions causing ties in each iteration. These ties have a great influence on the performance of the constructive heuristics (see [83]). In the original proposal, the first slot (denoted as FS) for which the minimum makespan is achieved is kept as the best sequence. This b field then defines the type of tie-breaking mechanism implemented in the NEH. The following mechanisms have been considered in the literature:
  - TBKK1: mechanism based on the Johnson's rule ([80]) proposed by [83].
  - TBKK2: tie-breaking mechanism based on the Johnson's rule ([80]) proposed by [84].
  - TBKK3: mechanism based on the Johnson's rule ([80]) proposed by [85].
  - DHC: tie breaking mechanism based on a balance usage of machines proposed by [35].

  - RCT: idle-time-based tie breaking mechanism proposed by [162] increasing the complexity of the NEH to $\max\{n^2 \cdot m^2, n^3 \cdot m\}$.

- c: This field is associated with the reversibility property of the problem (see [162]). It establishes that the makespan of the permutation $\Pi := (\pi_1, \ldots, \pi_n)$ in instance I (instance formed by n jobs and m machines with processing times equal to p_{ij}) is the same as the makespan of the reverse permutation $\Pi^{'} := (\pi_n, \ldots, \pi_1)$ in instance $I^{'}$ (instance formed by n jobs and m machines with processing times equals to $p^{'}_{ij} = p_{m-j+1,i}$). Therefore, the following values can be adopted:

  - d: This value is employed to denote a direct instance (i.e. the initial order Π is applied to instances I and $I^{'}$).
  - i: It is employed when the same initial order π is applied to the instance $I^{'}$. Accordingly, this field contains the value di when the inverse instance is carried out after performing the direct instance.

This notation has been employed to classify the different variants of the original NEH heuristic –which can be denoted as $NEH(SD|FS|d)$ in our notation– proposed in the literature. These are summarized in Table 4.1.

Among the heuristics proposed, some of them –i.e. NEH1, NEHKK1, NEHKK2 and NEHD by [83], [84] and [35] respectively – maintain the original complexity of $O(n^2m)$. Other variants with a greater complexity have been proposed by [162], see Table 4.1.

Two different variants with a greater complexity have been proposed by [211] and are denoted as $\mathrm{CL}_{\mathrm{WOTS}}$ and $\mathrm{CL}_{\mathrm{WTS}}$. In $\mathrm{CL}_{\mathrm{WOTS}}$, a new mechanism (denoted as the backward shift mechanism) is added to the traditional insertion phase of the NEH. This mechanism increases the sequences to be evaluated in each iteration by means of a movement of the jobs of the partial sequence. When the tie-breaking mechanism of [162] is added to the $\mathrm{CL}_{\mathrm{WOTS}}$, the heuristic is denoted as $\mathrm{CL}_{\mathrm{WTS}}$

Furthermore, 10 heuristics which also modify the insertion phase of the NEH algorithm have been proposed by [150]. These heuristics are denoted as: FRB1, FRB2, FRB3, $\mathrm{FRB4}_2$, $\mathrm{FRB4}_4$, $\mathrm{FRB4}_6$, $\mathrm{FRB4}_8$, $\mathrm{FRB4}_{10}$, $\mathrm{FRB4}_{12}$ and FRB5. Among them, the FRB1 heuristic is statistically outperformed by several heuristics (e.g. $\mathrm{FRB4}_2$ and $\mathrm{FRB4}_4$) with shorter average CPU times. Finally, [201] proposed a constructive NEH-based heuristic, NEHI, which also considers different interpretations for the ties in the initial order of the NEH.

Regarding metaheuristics employed for the problem, a summary of them is shown in Tables 4.2 and 4.3. The first, second, third and fourth columns indicate the year of publication, the bibliographical

Table 4.1: Summary of heuristics derived from NEH

Heuristic	NEH Notation	Paper
RAER	$NEH(rand\|RCT\|d)$	[162]
RAER-di	$NEH(rand\|RCT\|di)$	[162]
KKER	$NEH(KK1\|RCT\|d)$	[162]
KKER-di	$NEH(KK1\|RCT\|di)$	[162]
NEHR	$NEH(SD\|RCT\|d)$	[162]
NEHR-di	$NEH(SD\|RCT\|d\&i)$	[162]
NEMR	$NEH(NM\|RCT\|d)$	[162]
NEMR-di	$NEH(NM\|RCT\|di)$	[162]
NEH	$NEH(SD\|FS\|d)$	[127]
NEH-di	$NEH(SD\|FS\|di)$	[162]
NEH1	$NEH(SD\|TBKK1\|d)$	[83]
NEH1-di	$NEH(SD\|TBKK1\|di)$	[162]
NEHKK1	$NEH(KK1\|TBKK2\|d)$	[84]
NEHKK1-di	$NEH(KK1\|TBKK2\|di)$	[162]
NEHKK2	$NEH(KK2\|TBKK3\|d)$	[85]
NEHD	$NEH(AD\|DHC\|d)$	[35]
NEHD-di	$NEH(AD\|DHC\|di)$	[162]
CL$_{WTS}$	$NEH(SD\|FS\|d)$ with a backward shift mechanism in the insertion phase	[211]
CL$_{WOTS}$	$NEH(SD\|RCT\|d)$ with a backward shift mechanism in the insertion phase	[211]
NEHI	Best of several runs of $NEH(-\|-\|-)$	[201]
FRB1	Similar to $NEH(SD\|FS\|d)$ including a local search method in the insertion phase	[150]
FRB2	Similar to $NEH(SD\|FS\|d)$ including a local search method in the insertion phase	[150]
FRB3	$NEH(SD\|FS\|d)$ including a local search method in the insertion phase	[150]
FRB4$_k$	$NEH(SD\|FS\|d)$ including a local search method in the insertion phase	[150]
FRB5	$NEH(SD\|FS\|d)$ including a local search method in the insertion phase	[150]

reference, the type of metaheuristic and the acronym (maintaining the same acronym as in the original papers) respectively. The fifth column shows the papers proposing metaheuristics that outperform the referenced one. In the sixth column, the benchmark(s) used for the computational evaluation are shown (the following notation is used: T1, [190]; T2, non-complete set of instances of [190]; R, [158], C, [10]; D, [30]; W, [206]; H, [71]; O, Other set of instances). The seventh column shows the $ARPD2$ values of the metaheuristics when tested on Taillard's benchmark [190]. When the raw makespan value for each instance is given in the paper, the $ARPD2$ is computed again using (3.2) and the best known value for those instances in order to have a common reference. Otherwise, the $ARPD2$ values of the paper are reported. Note that these papers could have used different upper bounds (UB) and the values are therefore only approximations. For papers using the same upper bounds as in [190], a factor of 0.565 is added to correct the $ARPD2$. This value is the difference in $ARPD2$ between the actual upper bounds and the upper bounds of [190].

The eight and ninth columns indicate the programming languages used for coding the algorithms as well as the raw speed of the processors used for the evaluation. Finally, the average CPU time on Taillard's instances as a function of the size of the problem (i.e. n and m) is calculated, when possible, in the last column in order to analyze the CPU requirements of the algorithms. This value is expressed in terms of t_j for metaheuristic j, a variable traditionally used in the literature to measure its stopping criterion as $n \cdot m \cdot t_j/2$ milliseconds (see e.g. [174]). When t_j is not indicated and/or other stopping criteria are used, t_j is calculated as follows:

$$t_j = \sum_{\forall i} t_{ij}$$

and

$$t_{ij} = \frac{2 \cdot CPU_{ij}}{n_i \cdot m_i}$$

where CPU_{ij} is the CPU time in milliseconds required by algorithm j in instance i. n_i and m_i and the number of jobs and machines in instance i. Therefore, t_{ij} balances the CPU time with the size of the problem, and t_j –average of t_{ij}– can be seen as an indicator of the average CPU time requirements of an algorithm, since, given an instance, n_i and m_i are constants for different algorithms.

For clarity, when a paper proposes several metaheuristics, these methods are included in the table as long as they are used as reference metaheuristics in other papers. Otherwise, only the best one among the reported results is selected. The language used to code the algorithms has been included in the table since languages can result in much bigger differences than those caused by the use of varying computer characteristics.

Table 4.2: Summary of metaheuristics I. Notation: AC, Ant Colony Algorithm; TS, Tabu Search; ALA, Adaptive learning approach; GA, Genetic Algorithm; IG, Iterated Greedy; ILS, Iterated local search; DE, Differential Evolution; SS, Scatter Search; DF, Discrete Firefly; BCA, Bee colony algorithm; PSO, Particle Swarm Optimization Algorithm; SA, Simulated Annealing; CS, Cuckoo Search; NN, Neural Network; EA, Evolutionary algorithm; EDA, Estimation of Distribution Algorithm; PA, Population based Algorithm; *, Compared under the same conditions; $'$, In case of a paper proposing two methods, it is used to distinguish the second one from the first one; $''$, In case of a paper proposing three methods, it is used to distinguish the third one from the first and second one; **, $ARPD2$ taken directly from the paper; ***, $ARPD2$ corrected by 0.565.

Year	Ref.	Algorithm	Notation	Outperformed by	Testbed	$ARPD2$(Taillard)	Coding Lang.	Computer	Parameter t
2004	[210]	AC	ACS	[37],[176],[111],[2]	T1	1.4**	C++	700 MHz	608.11
2004	[186]	Neuro-TS	EXTS	[37]	T2	0.245	C++	486 MHz	2884.85
2004	[154]	AC	PACO	[173]'*,[174]*,[138],[138]",[93],[218],[195],[222],[98],[2]*,[194],[22]	T2	0.71**	—	800 MHz	—
2004	[154]	AC	M-MMAS	[173]'*,[174]*,[138],[138]",[93],[195],[222],[98],[2]*,[194]	T2	0.80**	—	800 MHz	—
2004	[112]	SA	LWK-SA1	[208],[100],[99]	T2,D	0.853	—	400 MHz	331.8
2004	[38]	GA	GA	[103]*	O	—	Pascal	—	—
2004	[129]	SA	Hybrid SAA	—	T2	0.893	Pascal	450 MHz	—
2004	[130]	Hybrid SA	Hybrid SAA	—	T2	1.081	Pascal	1.7 GHz	134.06
2004	[128]	GA	GA	[216]*,[214]*,[217],[215]	T2	1.84**	Pascal	450 MHz	—
2006	[1]	ALA	NEH-ALA	[126]*,[93]	T1,C,R,H	1.514	Visual Basic	933 MHz	7907.6
2006	[173]	GA	GA_RMA	[173]'*,[174]*,[138],[138]",[126]*,[23],[98],[216]*	T1	1.12**,1.09**,1.02**	Delphi	2.8 GHz	30,60,90
2006	[173]'	GA	HGA_RMA	[174]*,[138],[138]",[195],[22]	T1	0.55**,0.47**,0.45**	Delphi	2.8 GHz	30,60,90
2006	[134]	DE	DE	[126]*,[149]	O	—	C++	733 MHz	—
2006	[132]	SS	MSSA	—	T2	>0.054	—	AMD2000+	1139.33
2006	[101]	PSO	SPSOA	[21],[217],[17],[75],[16]	T2	3.002	—	—	—
2006	[148]	GRASP	GRASP	—	T2,C,R	19.09	—	—	—
2006	[76]	ILS	LS	—	C,R	—	C	500MHz	
2007	[174]	IG	$\mathbf{IG\_RS_{LS}}$	[138],[138]",Chapter 6'*	T1	0.44**	Delphi	1400 MHz	60
2007	[192]	PSO	PSO_{spv}	[138]',[24],[192]',[77],[103]*,[115]	T2	4.01**	C	2.6 GHz	21.1
2007	[192]'	PSO	PSO_{vns}	[138],[138]",[208],[100],[106],[195],[111],[99],[110],[194],[22],[104]	T2,W	0.47**	C	2.6 GHz	264.64
2007	[106]	MA-PSO	PSOMA	[109],[107],[111],[99],[110],[17],[104]	C,R	—	Matlab	2.2 GHz	—
2007	[103]	PSO	PSO	[221],[100],[28]	T2,D	2.409**	C++	3.2 GHz	111.68
2008	[37]	TS	**3XTS**	—	T1	0.15***	C++	550 MHz,3.2 GHz	196.93
2008	[176]	SS	SS	—	T1	1.57***	C++	3.0 GHz	59.9
2008	[126]	GA	CGALS	[111]	T1	1.02**	Delphi	3.2 GHz	60
2008	[50]	GA	BDS	—	T2	0.64	—	1.8 GHz	7350
2008	[77]	PSO	CPSO	[138],[138]',[138]",[24]	T2	3.4**	C++	3.2 GHz	3.42
2008	[77]'	PSO	CPSO-PNEH	—	T2	0.59**	C++	3.2 GHz	19.12
2008	[77]"	PSO	**H-CPSO**	[138],[138]"	T2	0.45**	C++	3.2 GHz	229.34
2008	[18]	GA	ACGA	[21],[75]	R	—	—	—	—
2008	[149]	DE	HDE	[221],[100],[99],[98]	C,R	—	Delphi	2.8 GHz	—
2008	[102]	PSO	NPSO	[214]*,[216]*,[215],[110],[92]	T2	1.323	—	—	—
2008	[209]	AC	ACS	[216]*,[98]	R	—	VP	2 GHz	—
2008	[138]	IG	$\mathbf{IG_{RIS}}$	Chapter 6*	T1	0.33**	C++	3.0 GHz	30
2008	[138]'	DE	DDE	[138]*,[138]"*,[75]	T1	1.05**	C++	3.0 GHz	30
2008	[138]"	DE	$\mathbf{DDE_{RLS}}$	—	T1	0.32**	C++	3.0 GHz	30
2008	[216]	PSO	IPSO	[98]	T1	0.76**	C++	2.8 GHz	120

Table 4.3: Summary of metaheuristics II

Year	Ref.	Algorithm	Notation	Outperformed by	Testbed	$ARPD2$ (Taillard)	Coding Lang.	Computer	Parameter t
2009	[20]	GA	ACEGA	—	R	—	—	—	—
2009	[93]	Hybrid	PSA	—	T2	1.049	C	2.8 GHz	—
2009	[222]	GA with VNS	NEGA_{VNS}	[195],[22]	T1	0.468	C++	2.4 GHz	112.93
2009	[215]	PSO	ATPPSO	[100],[99],[110]	T2	1.269	—	—	—
2009	[92]	Hybrid PSO	HPSO	[110]	T2	0.760	C	1.73 GHz	555.04
2009	[194]	Hybrid GA	Hybrid GA	[28]	T2	0.63***	C++	1.83 GHz	199.79
2009	[155]	GA	IGA	—	C,R	—	C	—	—
2010	[177]	DF	Discrete Firefly	—	D	—	Matlab	3 GHz	—
2010	[221]	DE	QDEA	[208],[98],[100]	C,R,D	—	—	—	—
2010	[217]	PSO	L-CDPSO	[100],[99]	T2	0.409	—	—	—
2010	[214]	PSO	I-ATTPSO	—	T2	1.331	—	2.9 GHz	—
2010	[64]	NN-GA	ANN-GA-RIPS	—	T2	2.519	—	—	—
2010	[19]	GA	ACGA	[17],[75]	R	—	—	—	—
2011	[21]	GA	HGIA	[17],[75]	T2,R	1.16	C++	1.86 GHz	—
2011	[107]	Hybrid PSO	PSO-EDA_PI	[109],[17]	C,R,W	—	Matlab	—	—
2011	[157]	NN	ANN-GA	—	T2	2.34**	—	—	—
2012	[24]	GA	Self-Guided GA	[138]',[75]	T2	1.85**	Java	3.2 GHz	—
2012	[195]	EDA with AC	$\mathbf{EDA_{ACS}}$	—	T1	0.572**,0.508**,0.463**	C	2.4 - 2.8 GHz	30,60,90,200
2012	[98]	BCA	CDABC	—	C,R,T1,D	0.62**	Matlab	3.0 GHz	120
2012	[2]	AC	NACA	—	T1	0.582**	C++	2 GHz	108.92
2013	[111]	Hybrid BCA	HDABC	—	T2,R	0.48**	C++	3.06 GHz	42.52
2013	[115]	PSO	PSOENT	[222]	T1	1.65**	Fortran	—	104.42
2013	[109]	MA-PSO	MPSOMA	—	C,R	—	Matlab	2.4 GHz	—
2013	[100]	DE-MA	ODDE	[208]	C,R,T2,D	0.400	Matlab	3.0 GHz	—
2013	[99]	CS	**HCS**	—	C,R,T2,D	0.401	Matlab	3.0 GHz	—
2013	[17]	EA	BBEA	[75]	T2,R	1.75**	—	3.4 GHz	—
2014	[208]	Hybrid	HTLBO	—	C,R,D	—	C++	2.0 GHz	—
2014	[218]	PSO	**PSO**	—	T2	0.39**	C++	3.0 GHz	—
2014	[23]	SA	SEASA	—	T1	0.94**	—	1.73 GHz	35
2014	[110]	Hybrid DE	L-HDE	—	T2,C,R	0.750	Matlab	3.0 GHz	—
2014	[16]	EDA	BBEDA	[75]	T2,R	1.420**	—	—	—
2015	[65]	GA-SS	HGSS	—	D	—	C++	2.4 GHz	—
2015	[22]	PA	HLBS	—	T1	0.45**,0.38**,0.35**,0.30**	C	2.4 and 2.8 GHz	30,60,90,200
2015	[28]	CS	DISCS	—	T2,W	2.84**	Matlab	2.7 GHz	10.88
2015	[75]	EA	LMBBEA	—	T2,R	0.89	—	—	—
2015	[104]	EA	HBSA	[149]	C,R	—	—	—	—

4.3 Total flowtime

The $Fm|prmu|\sum C_j$ is known to be NP-hard, therefore most of the research on this topic is devoted to developing approaches yielding good (but not necessarily optimal) solutions in reasonable computation time. An excellent review on these heuristics is provided by [137] where 14 of these heuristics are identified as efficient. In the following, we just outline the main aspects of these heuristics and refer the interested reader to the paper by [137] for a more detailed description of all existing heuristics in the literature.

- Heuristic $LR(x)$ [108]. This heuristic constructs a solution for the problem by appending, one by one, the unscheduled jobs (jobs in set U in the following) at the end of a sequence S of already scheduled jobs. To do so, ξ_{jk} an indicator of the suitability for job j ($j \in U$) to be scheduled in last position (position $k+1$ where k indicates the amount of scheduled jobs in each iteration) is calculated according to:

  $$\xi_{jk} = (n-k-2) \cdot IT_{jk} + AT_{jk}$$

  where IT_{jk} estimates the weighted idle time induced when scheduling job j in position $k+1$, i.e.:

  $$IT_{jk} = \sum_{i=2}^{m} \frac{m \cdot max\{C_{i-1,j} - C_{i,[k]}, 0\}}{i + k \cdot (m-i)/(n-2)}$$

  and AT_{jk} is the so-called artificial flowtime and it is defined as the sum of the completion time of job j plus the completion time of job p, an artificial job with processing times equal to the average processing time of the other jobs in U (excluding job j), and can be computed as follows:

  $$AT_{jk} = C_{mj} + C_{mp}$$

  More specifically, the $LR(x)$ heuristic operates as follows:

  1. Sort all jobs in ascending order of indicator ξ_{j0} (Let us U denote such ordered set). Ties are broken in favor of jobs with higher IT_{j0}.
  2. Use each of the first x ranked jobs in U as the first job in S, and then constructs a solution by appending the rest of the jobs one by one using indicator ξ_{jk}
  3. Out of the x solutions so obtained, select the one with the minimum flowtime.

- Heuristic $LR(x)-FPE(y)$ [108]. This is a composite heuristic where a local search method (denoted $FPE(y)$) is applied to the solution of $LR(x)$. $FPE(y)$ consists of the following steps: For each job

j in a sequence, this job is exchanged with the next y jobs in the sequence, and the flowtimes of the so-obtained solutions are evaluated. If any of the solutions has improved the flowtime, then the local search procedure is repeated. Otherwise, the local search stops.

- Heuristic NEH [127] (see Section 4.2 for a detailed description). Originally conceived for minimizing the makespan in a permutation flowshop, this well-known algorithm has been used as a reference method for many problems in the literature. Its application to the flowtime minimisation problem was discussed by [42], and it was found that the best option is to first sort the jobs in ascending sum of their processing times.

- Heuristic Raj [151]: This heuristic can be seen as a version of the NEH, but here job k is inserted only in slots $\lfloor k/2 \rfloor$ to k, thus reducing the computation time. Additionally, jobs are initially sorted in ascending order of index T_j as defined in equation (4.1), breaking ties in favor of the job with the lowest sum of total processing times.

$$T_j = \sum_{i=1}^{m} (m - j + 1) \cdot p_{ij} \tag{4.1}$$

- Heuristic $LR - NEH(x)$ [137]. This is a composite heuristic where the last $n/4$ steps of each x sequences obtained applying the $LR(x)$ procedure are carried out according to the NEH heuristic instead of the normal procedure of the $LR(x)$ algorithm, i.e., the first $3/4n$ jobs of each sequence are scheduled according to the $LR(x)$ procedure and the rest according to the NEH procedure.

- Heuristic RZ [152]. This heuristic consists of two steps: An initial ordering, and an improvement phase. With respect to the initial ordering, the jobs are sorted in ascending order of the total processing times. The improvement phase (denoted iRZ in the following) consists of inserting each job in the sequence in the rest of positions updating the sequence when a better solution is found.

- Heuristic $RZ - LW$ [97]. This heuristic consists in iteratively performing iRZ until no further improvement is found.

- Heuristic ICi [96]. This is a family of composite heuristics where an initial solution is obtained by using $LR(1)$ and then improved by using different local search methods. If the local search is performed using the iRZ procedure, then the heuristic is denoted $IC1$. Heuristic $IC2$ performs FPE on the solution obtained by $IC1$. Finally, $IC3$ consists of running $IC1$ and then performing a local search denoted as $FPE - R$, which is essentially FPE adding a restart from the first job every time the current solution is improved.

- Heuristic $PRi(x)$ [137]. These are several composite heuristics: $PR1(x)$ performs iRZ on each one of the x sequences obtained by heuristic $LR-NEH(x)$. $PR2(x)$ first run the heuristic $LR-NEH(x)$ and then tries to improve this solution using a VNS-like (Variable Neighborhood Search) local search method. This method was introduced by [192] and consists in an insertion and interchange variant of the classical VNS where insertion and interchange movements are repeated until no further improvement is found. $PR3(x)$ performs x times a iRZ and two NEH methods after an initial solution obtained by heuristic $LR-NEH(10)$. Finally, $PR4(x)$ replaces the iRZ method of $PR3(x)$ by a VNS local search. In order to bound the computation time of the heuristics, if the CPU time reaches the value of $0.01 \cdot n \cdot m$ seconds, a last loop is performed and the procedure terminates.

All aforementioned heuristics have at least a complexity of $O(n^3 \cdot m)$, and most of them use the LR heuristic to generate a seed solution.

4.4 Due-date related objectives

In this section, we review the PFSP minimising due-date-based objectives. We focus in two well-known decision problems: the $Fm|prmu|\sum T_j$ and the $Fm|prmu|\sum E_j + \sum F_j$ problems.

Regarding the $Fm|prmu|\sum T_j$ problem, most researchers have focused on developing solution procedures (i.e. heuristics) that do not guarantee the optimality of the solution, but that can provide a (hopefully) good solution in a reasonable time interval due to the NP-hard nature of the problem. More specifically, several heuristics and metaheuristics have been proposed in the literature for the problem, such as those by e.g. [56, 90, 153, 46, 197].

The extensive computational evaluation of heuristics for the problem carried out by [199] shows that NEHedd is a key constructive heuristic for the problem since, aside to being very efficient, the rest of efficient heuristics in the literature with more average CPU time employ NEHedd as an initial solution. More specifically, more than half of the state-of-the-art improvement heuristics or metaheuristics for the problem use NEHedd as a starting solution. This fact can be also seen in more recent works, such as [197], or [178]. The NEHedd heuristic differs from the NEH heuristic in the starting order (jobs are arranged now according to the Earliest Due Date or EDD rule), and in the evaluation of the partial sequences (as the one with lowest total tardiness is selected). Taillard's acceleration cannot be applied to the NEHedd, and, although [197] propose a mechanism similar to that by [96], the complexity of the NEHedd remains $O(n^3 \cdot m)$.

Regarding the $Fm|prmu|\sum E_j + \sum F_j$ problem, given the acceptance of just-in-time systems in practice, there is a growing interest in the last decades in analysing scheduling problems where both earliness and tardiness are penalised (see e.g. reviews [4], [94], [81] and [183]). Although, this type of problems is collectively known as E/T problems, the problem under consideration is the PFSP to minimise total earliness and tardiness, $Fm|prmu|\sum E_j + \sum T_j$. Some exact approaches and approximate algorithms have been proposed in the literature for this problem. However, both the NP-hard nature of the problem (see [117]) and the huge computation times required by the optimal approaches even for small instances (not more than 20 jobs) justify the need to develop fast approximate algorithms. The methods are classified into two groups:

- Methods for E/T problems on a flowshop where the idle time can be inserted.
- Methods for E/T problems on a flowshop where the idle time cannot be inserted.

On the one hand, regarding the problem allowing the insertion of idle times, [168] propose several mixed-integer models for the problem including insertion of idle time as well as considering unlimited and zero buffer. [15] propose some approximate approaches to solve the PFSP to minimize the sum of earliness and tardiness with common due dates for all jobs. Finally, [118] combine the VNS search with the mixed integer programming to solve the same problem but without common due dates.

On the other hand, without insertion of idle times, [123] propose an optimal algorithm for the PFSP with two machines to minimise the sum of maximum earliness and tardiness, as well as branch-and-bound algorithms are developed in [114] for several multi-objective functions including the total earliness and tardiness. [213] was first in proposing an approximate algorithm (more specifically a simulated annealing algorithm) to solve the PFSP to minimise the sum of weighted earliness and tardiness. In [179], a genetic algorithm (GA) has been developed which outperform several metaheuristics of similar research problems as well as the algorithm proposed by [213]. Using the same benchmark, [117] propose an iterated local search (ILS) where a variable neighborhood descent is iteratively repeated after a perturbation mechanism, which gave better results than the GA. However, both the GA and the ILS algorithms use the NEHedd (originally proposed for the $Fm|prmu|\sum T_j$ by [88]) and the Earliest Due Date or EDD rule respectively, which are either very simple seed sequences or simple adaptations from another research problems.

4.5 Conclusions

Makespan

Although the excellent performance of non-population based algorithms was shown by [174],[138], the literature using this type of metaheuristic is scarce and researchers have mainly been focused on the implementation of algorithms using several populations in parallel. Note that the best metaheuristics (see e.g. iterated greedy algorithm) and the best heuristics (see e.g. NEH) include Taillard's accelerations, which is a special characteristic of the $Fm|prmu|C_{\max}$. It probably represent the main reason for the excellent behaviour of insertion phases in the algorithms and could explain its extensive use in the heuristics and metaheuristics of the last decade, as well as the excellent performance of the NEH and IG-based algorithms. Both in NEH and the Iterated Greedy algorithm, ties among (sub)sequences yielding the lowest makespan may occur. In the original proposals, no specific mention on ties is given, so it is usually assumed that the first slot for which the minimum makespan is achieved when inserting job in position k is kept as the best (sub)sequence. However, the mechanism employed to break these ties has a great influence on the performance of these algorithms, as well as it represents an advance in their intensification, as [83] first attested for the NEH. To the best of our knowledge, there is no proposal of integrating tie-breaking mechanisms in the Iterated Greedy algorithm. In our opinion, these facts highlight that a special effort should be made, firstly, in a better understanding of the problem and its properties (specific objective SO1 and secondly in applying them to non-populations algorithms for the problem (specific objective SO2).

In view of Tables 4.1, 4.2 and 4.3, there are very few papers whose methods are directly compared with the state-of-the-art algorithms (i.e. the IG_RS$_{\text{LS}}$ by [174]). Most of them are directly compared with metaheuristics of the same type (i.e. papers proposing PSO metaheuristics are compared with other PSO metaheuristics). Additionally, among all analyzed metaheuristics, only 9 papers (less than 10%) explicitly state that the metaheuristics are compared using the same conditions. Finally, there is no homogeneity in the set of instances used to compare the methods. Most metaheuristics (56) are tested in Taillard's benchmark, although only 20 of these use all 120 instances of the testbed. The rest of the testbeds used were mainly Reeves' (23 times) and Carlier's (15 times). From this literature review, the current state-of-the-art is far from easy to identify.

As a conclusion, a new evaluation of the approximate methods for the $Fm|prmu|C_{\max}$ problem is pertinent (specific objective SO3) and may serve firstly to establish a clear picture of the state-of-the-art within this important problem, and secondly, to give indications of possible avenues for future research. Additionally, a special effort should be also made when comparing efficient heuristics against the best metaheuristics under the same stopping criterion since the CPU time required by some heuristics is

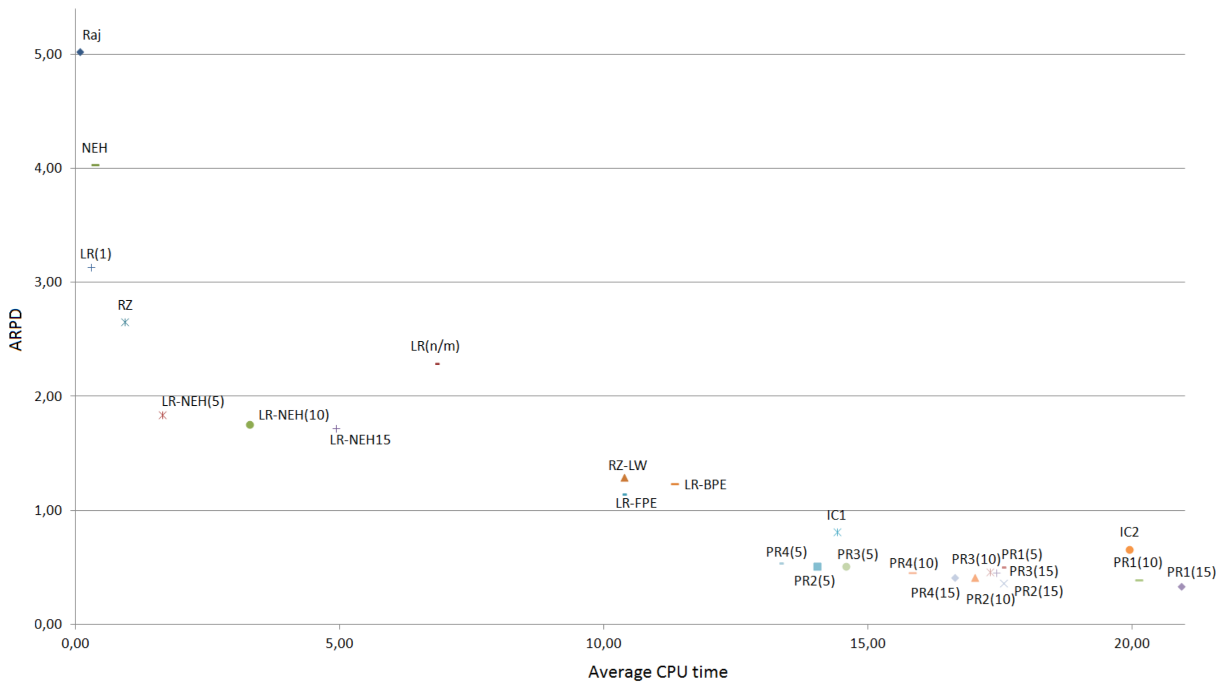

Figure 4.1: Pareto set using the average computational time [137]

relatively high in comparison with some metaheuristics (specific objective SO4).

Total flowtime

The heuristics discussed in Section 4.3 constitute the (so-far) set of efficient heuristics for the problem, as found by [137] in their exhaustive analysis of all existing heuristics for the $Fm|prmu|\sum C_j$ problem with respect to the quality of the solutions and computational requirements. Since there is a clear tradeoff between the solution obtained by one heuristic, and its computation time, the authors were able to depict a Pareto set to place the efficient heuristics for the problem in view of their performance on the well-known Taillard's testbed (see Figure 4.1). As it turns out, this Pareto set is formed by the following heuristics: Raj, $LR(1)$, RZ, $LR-NEH(5)$, $LR-NEH(10)$, $LR-NEH(15)$, $LR-FPE$, $PR4(5)$, $PR2(5)$, $PR3(5)$, $PR4(10)$, $PR4(15)$, $PR2(15)$ and $PR1(15)$.

From the analysis of the Pareto set, some conclusions can be derived:

- As it can be seen in Table 7.1, all the efficient heuristics consist on variation/adaptations of the following five main (or primary) procedures: NEH, $LR(x)$, FPE, iRZ and VNS. More specifically, the $LR(x)$ heuristic is present in 12 of the 14 heuristics in the Pareto set.

- Regarding the complexity of the five primary procedures, NEH is known to be $O(n^3 \cdot m)$, the same complexity as LR. It is easy to check that each iteration of iRZ is $O(n^3 \cdot m)$. Hence, the iRZ

has a complexity of $k \cdot n^3 \cdot m$ with k the number of iterations in which there is an improvement in the objective function. The complexity of FPE corresponds to $x \cdot k \cdot n^2 \cdot m$ (for $FPE - R$, the worst case is $O(x \cdot k \cdot n^3 \cdot m)$), with k indicating again the number of iterations with improvement in the objective function. From the complexity of these five procedures, the complexity of the rest of algorithms in the Pareto set can be easily obtained (this information is summarised in Table 7.1). As it can be seen, each heuristic in the Pareto set has at least a complexity of $n^3 \cdot m$. Note that the parameter k cannot be nor bounded neither linked to the problem size. However, in the computational experience carried out in Taillard's testbed (see Section 3.2), this value is usually larger than both n and m.

A detailed analysis of this Pareto set reveals that 12 out of the 14 heuristics employ a mechanism for constructing the solutions based in the heuristic by [108]. However, its complexity is still high in comparison with the best heuristic of related scheduling problems (see e.g. $Fm|prmu|C_{max}$). For this problem, new advances should come from a reduction in the complexity of the heuristics to obtain similar or even better solutions with much lesser CPU time (specific objective SO5)

Due-date related objective

Despite the excellent performance of the NEHedd heuristic, we believe that additional improvements could be gained by further analysis of the problem under consideration. First, the $Fm|prmu|\sum T_j$ and the $Fm|prmu|\sum E_j + \sum F_j$ could resemble different scheduling problems depending on the due dates of the jobs for each specific instance: Intuitively, it is clear that, for an instance with due dates much greater than the sum of the processing times of its jobs, almost every schedule may yield zero total tardiness for the $Fm|prmu|\sum T_j$ (turning the problem into a trivial one), as well as the problem resembles that of flowtime maximisation for the $Fm|prmu|\sum E_j + \sum F_j$. Analogously, unachievable due dates for each job results in an instance for which almost every sequence yields tardiness for every job and therefore both problems resembles that of minimising flowtime. By conducting an analysis of these possible scenarios (specific objective SO6 and SO7), further insights into the problem can be obtained, so the performance of the NEHedd procedure can be enhanced (specific objectives SO8 and SO9).

Summary of specific objectives

Several specific objectives for the PFSP have been identified above, which will be addressed in this Thesis. In this section, we summarise them:

SO1. To provide further insights into the problem and its properties.

SO2. To propose non-populations algorithms for the $Fm|prmu|C_{max}$ problem.

SO3. To perform a new computational evaluation of heuristics and metaheuristics for the $Fm|prmu|C_{max}$ problem.

SO4. To compare efficient heuristics against the best metaheuristics under the same stopping criterion for the $Fm|prmu|C_{max}$ problem.

SO5. To develop new efficient heuristics decreasing the typical complexity of the algorithms for the $Fm|prmu|\sum C_j$ problem.

SO6. To analyse different scenarios depending on the due dates for the $Fm|prmu|\sum T_j$ problem.

SO7. To analyse different scenarios depending on the due dates for the $Fm|prmu|\sum E_j + T_j$ problem.

SO8. To design efficient heuristics for the $Fm|prmu|\sum T_j$ problem.

SO9. To develop of efficient heuristics for the $Fm|prmu|\sum E_j + T_j$ problem.

Chapter 5

Influence of input parameters

5.1 Introduction

In the vast majority of the works reviewed in Chapter 4, it has been assumed that a) processing times are not job- and/or machine-correlated, b) processing times are not resource dependent, c) due dates are generated by the same distribution for all jobs and d) all machines are initially available. In this chapter, addressing Objectives GO2, SO1, SO6 and SO7, we try to go deeper in the understanding of the influence of these assumptions over the PFSP.

Firstly, we will show that under certain conditions of the due dates, the $Fm|prmu \sum T_j$ and the $Fm|prmu \sum E_j T_j$ can be reduced to other different related scheduling problems. In order to show that, several properties are shown for both scheduling problems, to identify the theoretical conditions required for them. In addition, we analyse how far the due dates traditionally generated in the literature are from the above conditions.

Secondly, we will show that under certain conditions, or correlated processing times, the PFSP could be considered easily solvable or be equivalent to other decision problems. To address it, several properties and dominance rules are presented to analyse the relation between the PFSP and the single machine scheduling problem, denoted as SMSP, for makespan and total flowtime minimisation depending on the processing times of the jobs. Additionally, in order to empirically compare the problems, 11 algorithms (5 for makespan and 6 for total flowtime) are tested on an extensive testbed with more than 600,000 instances designed for the PFSP with machine correlated processing times. Four algorithms have been designed to solve directly the instances of the PFSP. Five of them reduce each instance to an equivalent SMSP considering only the most saturated machine, whereas the other 2 algorithms solve a reduced PFSP without considering the machines before the most saturated one. Results show that the algorithms designed for the PFSP and for the SMSP tend to be very similar for several values of the parameters of the testbed. The goal is to prove the intuition that, when in a PFSP there is a machine much more saturated than the rest, then the problem should be similar to the equivalent SMSP considering only the most saturated machine. Thereby, we intend to explore the theoretical and empirical boundaries between these two problems.

Finally, we will also analyse the different relationships between the processing time of an operation and the number of resources assigned to that operation. Traditionally, different functions have been used in the literature in order to map the processing time of the operation with the amount of resources assigned to the operation. Obviously, this relation depends on several factors such as the type of resource and/or decision problem under study. Although in the literature there are hundreds of papers using these relations in their models or methods, most of them do not justify the motivation for choosing a specific relation over another one. In some cases, even wrong justifications are given and, hence, infeasible or nonappropriated relations have been applied for the different problems, as we will show. Thus, we intend to fill this gap establishing the conditions where each relation can be applied by analysing the relations between the processing time of an operation and the amount of resources assigned to that operation.

More specifically, the outline of the rest of this chapter is organised as follows:

- In Section 5.2, the influence of the due dates is analysed.
- In Section 5.3, we discuss the influence of the processing times.
- In Section 5.4, we discuss the functions used in the literature for controllable processing times.

5.2 Due dates

The $Fm|prmu|\sum T_j$ and $Fm|prmu|\sum E_j + \sum T_j$ problems are highly influenced by the due dates of the jobs in a specific instance. In this section, we make an effort to gain a better understanding of the problems so the performance of existing solution procedures can be enhanced.

Theoretical analysis

Let first state four simple properties:

Property 5.2.1. *Let $\mathcal{I}$ be an instance of the $Fm|prmu|\sum E_j + \sum T_j$ problem, and WM the worst (maximum) makespan for the instance. If $d_j \geq WM\ \forall j$, an optimal solution for $\mathcal{I}$ is obtained by solving the corresponding $Fm|prmu|-\sum C_j$ problem for $\mathcal{I}$.*

Proof. Since each due date is greater or equal than the worst makespan WM, then each due date d_j is greater or equal than its completion time, $C_{mj}(\Pi)$ (i.e. $d_j \geq WM \geq C_{m,j}(\Pi)$, $\forall j, \Pi$). Hence, minimising $\sum_{\forall j} max\{C_{m,j}(\Pi)-d_j, 0\}+\sum_{\forall j} max\{d_j - C_{m,j}(\Pi), 0\} = 0+\sum_{\forall j} d_j - C_{m,j}(\Pi) = \sum_{\forall j} d_j - \sum_{\forall j} C_{m,j}(\Pi) = const - \sum_{\forall j} C_{m,j}(\Pi)$. □

Property 5.2.2. *Let $\mathcal{I}$ be an instance of the $Fm|prmu|\sum E_j + \sum T_j$ problem verifying that $d_j \leq \sum_{i=1}^{m} t_{ij}$ $\forall j$. Then, an optimal solution for $\mathcal{I}$ can be obtained by solving the corresponding $Fm|prmu|\sum C_j$ problem for $\mathcal{I}$.*

Proof. Considering $d_j \leq t_j$ $\forall j$, each completion time $C_{m,j}(\Pi)$ ($\forall$ Π) is greater or equal than its due date, d_j, since t_j is a lower bound of the makespan of the job j. Hence $\sum_{\forall j} max\{C_{m,j}(\Pi) - d_j, 0\} + \sum_{\forall j} max\{d_j - C_{m,j}(\Pi), 0\} = \sum_{\forall j}(C_{m,j}(\Pi) - d_j) + 0 = \sum_{\forall j} C_{m,j}(\Pi) - \sum_{\forall j} d_j = \sum_{\forall j} C_{m,j}(\Pi) + const.$ □

Property 5.2.3. *Let $\mathcal{I}$ be an instance of the $Fm|prmu|\sum T_j$ problem, and WM be the maximum (worst) makespan that can be obtained for $\mathcal{I}$. If $d_j \geq WM, \forall j$, then each feasible sequence π is an optimal solution for $\mathcal{I}$. That is, $\mathcal{I}$ has $n!$ optimal solutions.*

Proof. The proof of this property is obvious: since WM is the worst makespan of the problem (i.e. $WM \geq C_{m,j}, \forall j$) and each due date is greater than or equal to WM (i.e. $d_j \geq WM \geq C_{m,j}$, $\forall j$), then minimising $\sum_{\forall j} max\{C_{m,j} - d_j, 0\}$ is equal than minimising $\sum_{\forall j} max\{-P, 0\} = 0$, where P is a non-negative number, and hence each feasible solution is an optimal solution of the problem. □

Property 5.2.4. *Let $\mathcal{I}$ be an instance of the $Fm|prmu|\sum T_j$ problem with $d_j \leq \sum_{i=1}^{m} t_{ij}$ $, \forall j$. Then, an optimal solution for $\mathcal{I}$ can be obtained by solving the corresponding $Fm|prmu|\sum C_j$ problem for $\mathcal{I}$.*

Proof. Trivial in view of Property of 5.2.2. □

On the one hand, from Properties 5.2.1 and 5.2.2, it is clear that extremely loose due dates transform the $Fm|prmu|\sum E_j + T_j$ into a PFSP with the objective of flowtime maximization. Extremely tight due dates lead to a problem similar to the PFSP with flowtime minimisation. Both bounds represent opposite objective functions and therefore, algorithms specifically focused on yielding good solutions for instances with loose due dates would necessarily perform bad for tight due dates. Thereby, depending on the due dates, three different scheduling problems can be solved: the $Fm|prmu|\sum C_j$ problem in case of tight due dates, the $Fm|prmu| - \sum C_j$ problem in case of loose due dates and the original $Fm|prmu|\sum E_j + \sum T_j$ problem in the rest of the cases. This fact speaks for the difficulties to find constructive heuristics that perform well for the problem, which in our opinion is reflected by the fact that the NEH –an algorithm not designed for this specific problem– is the only constructive heuristic proposed so far.

On the other hand, for the $Fm|prmu|\sum T_j$ problem, Properties 5.2.3 and 5.2.4 formalise the interdependence between the due dates and processing times of an instance, and the type of optimisation problem. If the due dates are extremely tight, the $Fm|prmu|\sum T_j$ is similar to that of flowtime minimisation ($Fm|prmu|\sum C_j$) according to Property 5.2.4 whereas extremely loose due dates lead to a trivial problem according to Property 5.2.3.

Therefore, a problem instance can be classified along these two extreme cases (extremely loose and tight due dates). To do so, we first define for each job j the following indicator v_j:

$$v_j = \frac{d_j - t_j}{WM - t_j} \tag{5.1}$$

Clearly, $v_j \leq 0$ indicates that the due date cannot be met for job j, regardless of the position where it is scheduled. Similarly, $v_j \geq 1$ corresponds to the case where the completion time of job j is lower than its due date. By adequately truncating v_j, we can obtain a normalised indicator for job j, i.e.: $\min\{1; \max\{0; v_j\}\} \in [0,1]$.

Then, the indicator v can be defined as:

$$v = \sum_{j=1}^{n} \frac{\min\{1; \max\{0; v_j\}\}}{n} = \sum_{j=1}^{n} \frac{\min\{WM - t_j; \max\{0; d_j - t_j\}\}}{n \cdot (WM - t_j)} \tag{5.2}$$

It can be shown that $v \in [0,1]$, and that if, for a given instance, $v = 0$ (tight due dates), then minimising the total tardiness is equivalent to minimising the total flowtime. On the other extreme, if $v = 1$ (loose due dates), then any sequence is optimal.

In addition to how tight/loose the due dates are, the variability of the due dates among jobs also plays an important role in the optimization problem, which is formalised using the following property:

Property 5.2.5. *The sequence $\pi^{edd} := \left(\pi_1^{edd}, \cdots, \pi_n^{edd}\right)$ obtained by the EDD rule, is an optimal solution of the $Fm|prmu|\sum_j T_j$ problem if $d_{\pi_j^{edd}} \geq d_{\pi_{j-1}^{edd}} + \sum_{i=1}^m t_{i\pi_j^{edd}}$ (or, equivalently, $d_{\pi_j^{edd}} \geq \sum_{k=1}^{j}\sum_{i=1}^m t_{i\pi_k^{edd}}$), $\forall j > 1$, and $d_{\pi_1^{edd}} \geq \sum_{i=1}^m t_{i\pi_1^{edd}}$.*

Proof. Taking into account that $C_{m,\pi_{j-1}^{edd}} + \sum_{i=1}^m t_{i\pi_j^{edd}}$ is an upper bound of $C_{m,\pi_j^{edd}}$, i.e. $C_{m,\pi_{j-1}^{edd}} + \sum_{i=1}^m t_{i\pi_j^{edd}} \geq C_{m,\pi_j^{edd}}$, the property can be easily proved recursively, as follows: Beginning with the first job of the sequence, π_1^{edd}, and assuming that $d_{\pi_1^{edd}} \geq \sum_{i=1}^m t_{i\pi_1^{edd}}$, then $C_{m,\pi_1^{edd}} - d_{\pi_1^{edd}} \leq C_{m,\pi_1^{edd}} - \sum_{i=1}^m t_{i\pi_1^{edd}} = \sum_{i=1}^m t_{i\pi_1^{edd}} - \sum_{i=1}^m t_{i\pi_1^{edd}} = 0$, where it has been used that the completion time of the first job is equal to the sum of processing times, i.e. $C_{m,\pi_1^{edd}} = \sum_{i=1}^m t_{i\pi_1^{edd}}$. Hence, the first term of the objective function is zero, i.e. $C_{m,\pi_1^{edd}} - d_{\pi_1^{edd}} \leq 0 \longrightarrow max(C_{m,\pi_1^{edd}} - d_{\pi_1^{edd}}, 0) = 0$.

Following with the job in second position and assuming that $d_{\pi_2^{edd}} \geq d_{\pi_1^{edd}} + \sum_{i=1}^m t_{i\pi_2^{edd}}$, where

$C_{m,\pi_1^{edd}} \leq d_{\pi_1^{edd}}$ by means of the job in the first position. Then $d_{\pi_2^{edd}} \geq C_{m,\pi_1^{edd}} + \sum_{i=1}^{m} t_{i\pi_2^{edd}}$. Note that $C_{m,\pi_1^{edd}} + \sum_{i=1}^{m} t_{i\pi_2^{edd}}$ is an upper bound of $C_{m,\pi_2^{edd}}$ and, hence $d_{\pi_2^{edd}} \geq C_{m,\pi_1^{edd}} + \sum_{i=1}^{m} t_{i\pi_2^{edd}} \geq C_{m,\pi_2^{edd}}$ which implies that the completion time of the job in second position is again lower than its due date and that the second term of the objective function is again zero, i.e. $C_{m,\pi_2^{edd}} - d_{\pi_2^{edd}} \leq 0 \longrightarrow max(C_{m,\pi_2^{edd}} - d_{\pi_2^{edd}}, 0) = 0$.

For the job in a position j, we assume $d_{\pi_j^{edd}} \geq d_{\pi_{j-1}^{edd}} + \sum_{i=1}^{m} t_{i\pi_j^{edd}}$. As $C_{m,\pi_{j-1}^{edd}} \leq d_{\pi_{j-1}^{edd}}$ from the previous job and $C_{m,\pi_{j-1}^{edd}} + \sum_{i=1}^{m} t_{i\pi_j^{edd}} \leq C_{m,\pi_j^{edd}}$, then the completion time of the job in position j is lower than its due date as well as the jth term of the objective function is zero, i.e. $C_{m,\pi_j^{edd}} - d_{\pi_j^{edd}} \leq 0 \longrightarrow max(C_{m,\pi_j^{edd}} - d_{\pi_j^{edd}}, 0) = 0$.

Taking into account the last expression, the minimisation of total tardiness can be written as $max \sum max\{C_{m,j} - d_j, 0\} = max(0)$ and, hence, the EDD rule is optimal. □

Property 5.2.5 suggests that, for instances with high values of indicator v (i.e. loose due dates) and a high variability in the due dates of the jobs, the EDD rule may have a good performance for $Fm|prmu|\sum T_j$, as the due dates would have a greater influence on the objective function than the completion times of the jobs. Clearly, for such instances, employing more sophisticated algorithms might not pay off.

Analysis of the methods to generate the due dates

The five simple properties stated above determine different extreme cases of $Fm|prmu|\sum T_j$ and $Fm|prmu|\sum E_j + T_j$ where good/optimal solutions by algorithms designed for different problems (e.g. $Fm|prmu|\sum C_j$ and $Fm|prmu| - \sum C_j$). Obviously, the interest lies in finding efficient algorithms for instances in between these extreme cases. Therefore it is useful to review the different sets of instances that have been generated in the literature to check whether they adequately cover the specific tardiness minimisation case, or not.

To the best of our knowledge, testbeds for both scheduling problems have been built employing three different methods to generate due dates:

- [56] generate the due dates according to a uniform distribution drawn between the sum of processing time of the job and this sum plus an upper bound. This method for generating due dates is labelled in the following as GS.

- [147] generate the due dates using two parameters, T and R, related to the mean and variance of the due dates, respectively, according to an uniform distribution between $P \cdot (1 - T - R/2)$ and

$P \cdot (1 - T + R/2)$, where P is a lower bound for the makespan. This method is labelled in the following as PV.

- In [67], due dates are generated according to $(1 + 3 \cdot U[0,1]) \sum t_{ij}$. This method is denoted as HR in the following.

Clearly, these methods produce instances with different values of the indicator v and, in the case of the PV method, parameter R controls the variability of the due dates among jobs. To analyse the range of instances generated by each method, three different benchmarks have been built in the following manner: we consider the data regarding number of jobs, machines, and processing times as in the testbed $\mathcal{B}_3$, and generate three testbeds:

- The first testbed is generated using the PV procedure with parameters $T = \{0.2, 0.4, 0.6\}$ and $R = \{0.2, 0.6, 1.0\}$, and produced 5 replicates for each combination of m, n, T, and R. In total, 540 instances were obtained (see Section 3.2 for a more detailed description of this benchmark).
- The second testbed is generated using the GS procedure. To have the same number of instances than in the previous testbed, 45 replicates are generated for each combination of m and n.
- The third testbed is generated in an analogous manner to the previous one (with 45 replicates for each combination of m and n), but using the HR procedure for due date generation.

For each instance in the three benchmarks, the indicator v has been calculated according to expression (5.2), where the worst makespan, WM has been approximated using a modified version of the NEH to maximise makespan. The amount of instances for different intervals of v is shown in Figure 5.1 for the three benchmarks. In the figure in the left side, the percentage of instances is classified according to the parameter v whereas the figure in the right shows the cumulative percentage of instances. As can be seen, HR and specially GS produce many instances with very low values of v for which the problem is similar to minimising the total flowtime. For HR, 65% instances have a v lower than 0.15 while with GS all the instances have v lower than 0.20. Hence, in this Thesis, we focus in the generation of due dates according to the PV method, which is more likely to generate instances in the range of interest of both scheduling problems.

Computational Analysis

To further analyse the similarities between the related scheduling problem in the chosen set of instances, we empirically analyse the relationship between $Fm|prmu|\sum T_j$ and $Fm|prmu|\sum C_j$, for low values of

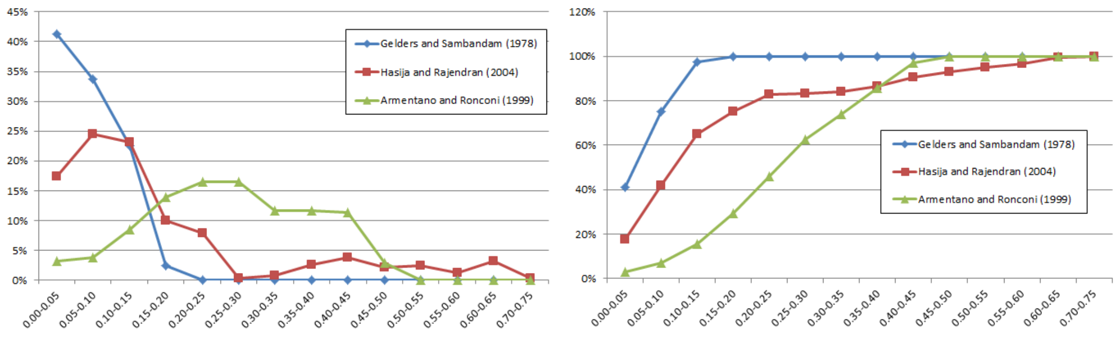

Figure 5.1: Distribution of the percentage of instances depending on v for different generation of due dates (In the left, the percentage of instances of the testbeds in each interval of v is shown, while the right figure shows the cumulative percentage of instances).

the parameter v. Thereby, we solve all instances in the testbed with the PV due date generation method using the NEHedd heuristic and the NEH heuristic for flowtime minimisation (denoted as NEH_FT). In addition, we obtain the solution given for each instance by the EDD rule in order to test the influence of higher values of v and R. Note that there are only two differences between NEHedd and NEH_FT:

1. The starting order of NEHedd is the EDD rule whereas in NEH_FT the starting order is the ascending order of the sum of the processing times, and
2. When iteratively constructing the solution, NEHedd selects the best partial sequence with lowest total tardiness, while NEH_FT selects the one with lowest flowtime.

As seen in Section 3.3, the usual indicator of the quality of the solutions with respect to tardiness is the relative deviation index (RDI). However, to better compare the performance obtained by the different heuristics that are to be tested in Chapter 8 and those by the NEHedd (which is the reference heuristic for $Fm|prmu|\sum T_j$ problem), we build the Compared Relative Deviation Index ($CRDI$), which is simply the difference between the RDI of the heuristic i and that of the NEHedd when both heuristics are applied to instance j, i.e.:

$$RDI_{ih} - RDI_{i,NEHedd} = CRDI_{ih} = \frac{sumT_{ih} - sumT_{i,NEHedd}}{Worst_i - Best_i} \cdot 100 \tag{5.3}$$

Clearly, $CRDI \in [-100, 100]$. In the subsequent experiments, $Worst_i$ and $Best_i$ are taken from the best and worst known total tardiness for the instances recorded in http://soa.iti.es/problem-instances. The values of $CRDI_{edd}$ and $CRDI_{NEH\_FT}$ are shown in Figure 5.2 with respect to indicator v for each instance of the benchmark, while Figure 5.3 groups the results for different values of R. The following conclusions can be obtained according to those results:

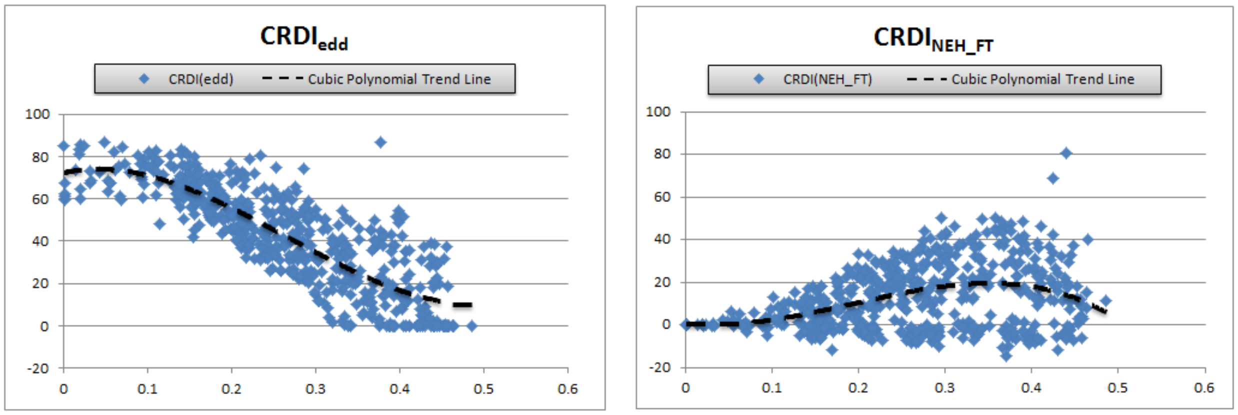

Figure 5.2: $CRDI_{edd}$ and $CRDI_{NEH\_FT}$ for different values of v in each instance of benchmark $\mathcal{B}_3$.

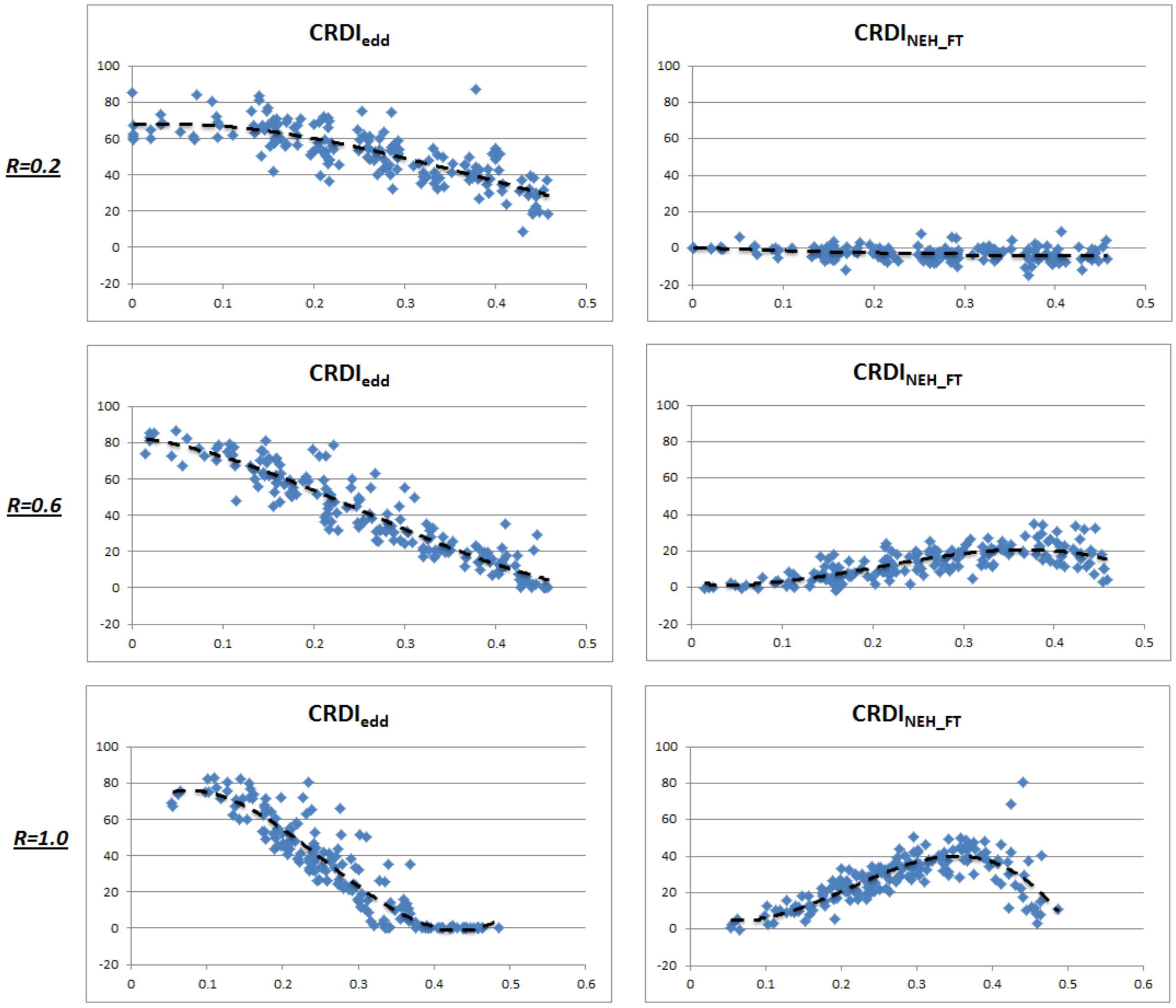

Figure 5.3: $CRDI_{edd}$ and $CRDI_{NEH\_FT}$ in each instance of benchmark $\mathcal{B}_3$ for different values of parameters v and R.

- As predicted by Property 5.2.4, the performance of NEH_FT and NEHedd procedure is very similar for low values of v. $CRDI_{NEH\_FT}$ is on average 0.79 for instances with $v < 0.1$ and 2.91 for instances with $v < 0.15$.

- NEH_FT outperforms NEHedd when the variance of the due dates is low, i.e. $R = 0.2$, even for high values of v. The average $CRDI_{NEH\_FT}$ for $R = 0.2$ is -2.71. This fact can be explained if we analyse the objective function when the variance of the due dates is zero (common due dates). Then, minimising $\sum_j max\{C_{m,j} - d_j, 0\} = \sum_{j \in late}(C_{m,j} - d_j) = \sum_{j \in late} C_{m,j} - \sum_{j \in late} d_j = \sum_{j \in late} C_{m,j} - L \cdot const$, where L is the number of jobs late. The first term is directly included in the minimisation of total flowtime, while the second term decreases when minimising total flowtime.

- In general, the performance of NEH_FT deteriorates as v increases until it reaches medium-high values (this is particularly clear for the combination of parameters $R = 0.6$ and $R = 1.0$), i.e. NEH_FT procedure only performs better when the problem can be reduced to either a flowtime minimisation problem (low v) or to a trivial one (high v).

- The performance of the EDD rule improves as v increases.

- For high values of v and a high variance of the due dates ($R = 1.0$), the EDD rule performs roughly as good as the NEHedd procedure, i.e. $CRDI_{edd} \simeq 0$. This could be predicted as a consequence of Property 5.2.5, since if the variance of the due dates of an instance is high enough to verify the conditions of Property 5.2.5, then EDD is optimal.

5.3 Processing times

This section –of computational/experimental nature– show that the $Fm|prmu|C_{\max}$ problem is also *empirically* hard, in the sense that optimal or quasi-optimal sequences statistically represent a very small fraction of the space of feasible solutions, and that there are big differences among the corresponding makespan values. In the vast majority of works solving the $Fm|prmu|C_{\max}$ problem, it has been assumed that a) processing times are not job- and/or machine-correlated, and b) all machines are initially available. However, some works (see [206] and [144]) have found that the problem turns to be almost trivial (i.e. almost every sequence yields an optimal or quasi-optimal solution) if one of these assumptions is dropped. To the best of our knowledge, no theoretical or experimental explanation has been proposed by this rather peculiar fact.

Our hypothesis is that, under certain conditions of machine availability, or correlated processing times, the performance of a given sequence in a flowshop is largely determined by only one stage, thus effectively transforming the flowshop layout into a single machine. Since the single machine scheduling problem with makespan objective is a trivial problem where all feasible sequences are optimal, it would follow that, under these conditions, the equivalent $Fm|prmu|C_{\max}$ problem is almost trivial. To address this working hypothesis from a general perspective, we investigate some conditions that allow reducing a permutation flowshop scheduling problem to a single machine scheduling problem, focusing on the two most common objectives in the literature, namely makespan and flowtime. Our work is a combination of theoretical and computational analysis, therefore several properties are derived to prove the conditions for an exact (theoretical) equivalence, together with an extensive computational evaluation to establish an empirical equivalence.

The additional notation necessary for this section can be set as follows. Let us $it_{i\pi_k}$ be the idle time immediately before job π_k on machine i of a PFSP. Clearly,

$$it_{i\pi_k} = \begin{cases} C_{i\pi_k} - C_{i\pi_{k-1}} - t_{i\pi_k}, & k \in 2 \ldots n \\ C_{i\pi_k} - t_{i\pi_k}, & k = 1 \end{cases} \tag{5.4}$$

or analogously,

$$it_{i\pi_k} = \max\{0, C_{i-1,\pi_k} - C_{i,\pi_{k-1}}\}, \forall k, C_{i0} = C_{0j} = 0 \tag{5.5}$$

Regarding the single machine scheduling problem, denoted as SMSP, n jobs have to be scheduled in a shop with a unique machine. The processing times and the completion times of job j in that machine are denoted by t_j and C_j respectively.

Once the PFSP and SMSP decision problems have been formulated, let us introduce some useful definitions. For a given instance of the PFSP, the machine s with the highest sum of processing times is denoted as *saturated machine*. More specifically:

$$s = \arg\max_i \sum_j t_{ij}$$

The remaining machines $i \neq s$ are denoted as *non saturated machines*. Additionally, let us consider two types of dominance between machines.

- Dominance type I, (see e.g. [11]): a machine a dominates (type I) a machine b if $t_{aj} \geq t_{bj'}, \forall j \neq j'$, where the machine b is consequently denoted as type-I-dominated machine.

- Dominance type II, (see e.g. [73], [25] and [203]): a machine a dominates (type II) a machine b (denoted as type-II-dominated) if $\min_{\forall j} t_{aj} \geq \max_{\forall j} t_{bj'}$.

Additionally, let us define the following dominance case for a flowshop of more than two machines:

- Case *ddm*: Each machine i dominates (type I) machine $i+1$, $\forall i \in [1, m-1]$.
- Case *idm*: Each machine i dominates (type I) machine $i-1$, $\forall i \in [2, m]$.
- Case *idm-ddm*: In this case, each machine i_1 ($\forall i_1 \leq s$) dominates (type I) machine $i_1 - 1$ and each machine i_2 ($\forall i_2 \geq s$) dominates (type I) machine $i_2 + 1$. Obviously, machine s is the saturated machine.

Finally, let us define the equivalence between PFSP and SMSP as follows: Given an instance $\mathcal{I}$ of a PSFP with processing times t_{ij}, and $\hat{\mathcal{I}}$ an artificial instance of a SMSP with $\hat{t}_j = t_{sj}$, we say that, for instance $\mathcal{I}$, both problems are equivalent regarding objective F if, for any feasible sequence Π, $F_{\mathcal{I}}(\Pi) = F_{\hat{\mathcal{I}}}(\Pi) + constant$. In other words, PFPS and SMSP are equivalent for an instance if the objective function values of all feasible sequences applied to both problems differ only with respect to a constant. Obviously, for an instance where both problems are equivalent, the optimal sequences are the same.

Theoretical analysis

Equipped with the above definitions, several properties can be derived to state when some PFSP instances are equivalent to SMSP for makespan and/or total flowtime minimisation under some (rather restrictive) assumptions. Since the PFSP with 2 machines has been widely analysed in the literature as an important particular case of the general m-machines cases, the properties presented in this section also adopt this division, and are formalised in two separate sections.

PFSP with 2 machines

Let first assume that machine s is the most saturated in the PFSP with 2 machines. In order to be able to show that one instance of the PSFP with two machines is equivalent to the SMSP, we need to state several properties and corollaries as well as define a condition to be satisfied for the saturated machine (the hardest condition needed to prove the properties and corollaries is that $t_{sj} \geq t_{ij'}$, $\forall j \neq j^{'}, i \neq s$).

First, we study the case where the first machine is saturated, i.e. $s = 1$. Let us define the following property in order to provide further insight into the understanding of this case:

Property 5.3.1. *Let* $\Pi := (\pi_1, \cdots, \pi_k, \cdots, \pi_n)$ *be a sequence of jobs with* $t_{1\pi_k} \geq t_{2\pi_{k-1}}, \forall k \geq 2$. *Then, the completion time of job* π_k *in the second machine equals its completion time on the first machine plus its processing time in the second, i.e.* $C_{2\pi_k} = C_{1\pi_k} + t_{2\pi_k}$, $\forall k$.

Proof. The property can be recursively proved in view of the definition of the completion time of job π_k on the first machine:

$$C_{1\pi_k} = \begin{cases} C_{1\pi_{k-1}} + t_{1\pi_k}, & \forall k \geq 2 \\ t_{1\pi_k}, & k = 1 \end{cases}$$

and in the second one:

$$C_{2\pi_k} = \begin{cases} \max\{C_{1,\pi_k}, C_{2,\pi_{k-1}}\} + t_{2\pi_k}, & \forall k \geq 2 \\ t_{1\pi_k} + t_{2\pi_k}, & \forall k = 1 \end{cases}$$

Beginning with the second job of the sequence, π_2: on the one hand, taken into account $t_{1\pi_2} \geq t_{2\pi_1}$, the completion time on the first machine is $C_{1\pi_2} = C_{1\pi_1} + t_{1\pi_2} = t_{1\pi_1} + t_{1\pi_2} \geq t_{1\pi_1} + t_{2\pi_1} = C_{2\pi_1} \longrightarrow C_{1\pi_2} \geq C_{2\pi_1}$; on the other hand, the completion time in the second machine is $C_{2\pi_2} = \max\{C_{1,\pi_2}, C_{2,\pi_1}\} + t_{2\pi_2} = C_{1,\pi_2} + t_{2\pi_2} \longrightarrow C_{2\pi_2} = C_{1,\pi_2} + t_{2\pi_2}$ using the expression of on the first machine.

Following with the third job of the sequence, π_3: the completion time on the first machine is $C_{1\pi_3} = C_{1\pi_2} + t_{1\pi_3} \geq C_{1\pi_2} + t_{2\pi_2} = C_{2\pi_2} \longrightarrow C_{1\pi_3} \geq C_{2\pi_2}$; in the second machine, the completion time is $C_{2\pi_3} = \max\{C_{1,\pi_3}, C_{2,\pi_2}\} + t_{2\pi_3} = C_{1,\pi_3} + t_{2\pi_3} \longrightarrow C_{2\pi_3} = C_{1,\pi_3} + t_{2\pi_3}$.

Analogously, in a recursive manner, for job in position k, π_k: on the first machine, the completion time is $C_{1\pi_k} = C_{1\pi_{k-1}} + t_{1\pi_k} \geq C_{1\pi_{k-1}} + t_{2\pi_{k-1}} = C_{2\pi_{k-1}} \longrightarrow C_{1\pi_k} \geq C_{2\pi_{k-1}}$; then the completion time in the second machine is $C_{2\pi_k} = \max\{C_{1,\pi_k}, C_{2,\pi_{k-1}}\} + t_{2\pi_k} = C_{1,\pi_k} + t_{2\pi_k} \longrightarrow C_{2\pi_k} = C_{1,\pi_k} + t_{2\pi_k}$.

□

This property extends the following result found by [120], which can be seen now as a corollary of the above property:

Corollary 5.3.1. *([120]: First part of Theorem 3 for $m = 2$). Let $\mathcal{I}$ be an instance of the PFSP where machine 2 is type-II-dominated by machine 1. Then, the completion time of a job on the second machine is equal to the completion time of a job on the first machine plus its processing time on the second machine.*

Proof. The proof of the corollary is obvious in view of Property 5.3.1. □

Corollary 5.3.2. *Let $\Pi := (\pi_1, \cdots, \pi_k, \cdots, \pi_n)$ be a sequence of jobs with $t_{1\pi_k} > t_{2\pi_{k-1}}, \forall k \geq 2$. Then, the idle time $it_{2\pi_k}$ is always greater than 0, i.e. $it_{2\pi_k} > 0$, $\forall k$.*

Proof. The proof of the corollary is obvious in view of Property 5.3.1 just by taking into account the definition of idle time given in Equation (5.5). □

The above property and corollaries establish respectively that the completion time of each job on the second machine depends only on its completion time on the first machine and that there are always idle time in the second machine. This occurs if the processing time of each job on the first machine is higher than its previous job on the second machine. Extending this condition to the processing time of each other job on the second machine, the equivalence between $F2|prmu|C_{max}$ with $1||C_{max}$ of the first machine is theoretically established in the Theorem 5.3.1 with the exception of the last job of the sequence.

Theorem 5.3.1. *Let $\mathcal{I}$ be an instance of the $F2|prmu|C_{max}$ where $t_{1j} \geq t_{2j'}$, $\forall j \neq j^{'}$ (i.e. machine 2 is dominated type I), and $\hat{\mathcal{I}}$ be an instance of the $1||C_{max}$ problem where $\hat{t}_j = t_{1j}$. Let Π be a sequence of the form $\Pi := \{\sigma, g\} := (\sigma_1, \cdots, \sigma_k, \cdots, \sigma_{n-1}, g)$ where g is the last job of the sequence and σ is an unknown sequence of $n-1$ jobs. Let C_{max} be the makespan on instance $\mathcal{I}$ of Π and $\hat{C}_{max}$ be the makespan on instance $\hat{\mathcal{I}}$ of Π. Then, for each feasible sequence, $C_{max} = \hat{C}_{max} + t_{2,g}$.*

Proof. Let us consider the PFSP with two machines to minimise makespan. In view of Property 5.3.1, $C_{max} = C_{2\pi_n} = C_{1\pi_n} + t_{2\pi_n}$. Then, minimising C_{max} in the $F2|prmu|C_{max}$ is equivalent to minimise $C_{1\pi_n} + t_{2\pi_n}$. Considering that π_n is job g, $C_{max} = C_{1\pi_n} + t_{2\pi_n} = C_{1\pi_n} + t_{2,g} = \hat{C}_{max} + t_{2,g}$. □

Corollary 5.3.3. *Under the conditions of Theorem 5.3.1, the optimal solution for $\mathcal{I}$ and $\hat{\mathcal{I}}$ is the same. Additionally, any sequence of the form $\Pi := \{\sigma, e\}$ is optimal for both instances where e is the job with the least processing time on the second machine, i.e. $t_{2e} = \min_{\forall j} t_{2j}$.*

Proof. The proof of the theorem is obvious in view of Theorem 5.3.1 and since each feasible solution is optimal for the $1||C_{max}$ problem. □

Note that the result of this theorem is given in [72] for $m = 2$ under more restrictive conditions, i.e. $\min_{\forall j} t_{1,j} \geq \max_{\forall j'} t_{2,j'}$, which can be seen now as a special case of the above result:

Corollary 5.3.4. *([72]: Theorem 2 for $m = 2$). Let $\mathcal{I}$ be an instance of the $F2|prmu|C_{max}$ where the machine 2 is type-II-dominated by the machine 1. Then, any sequence of the form $\Pi := \{\sigma, e\}$ is optimal where σ is any sequence of $n-1$ jobs and e satisfies $t_{2e} = \min_{\forall j} t_{2j}$.*

Proof. The proof of the theorem is obvious in view of Corollary 5.3.3. □

On the other hand, the equivalence between $F2|prmu|\sum C_j$ and $1||\sum C_j$ is theoretically proved by Theorem 5.3.2 and Corollary 5.3.5.

Theorem 5.3.2. *Let $\mathcal{I}$ be an instance of the $F2|prmu|\sum C_j$ where $t_{1j} \geq t_{2j'}$, $\forall j \neq j^{'}$ (i.e. machine 2 is dominated type I), and $\hat{\mathcal{I}}$ be an instance of the $1||\sum C_j$ problem where $\hat{t}_j = t_{1j}$. Let Π be a sequence of the form $\Pi := (\pi_1, \cdots, \pi_k, \cdots, \pi_n)$. Let $S(\Pi)$ be the total flowtime on instance $\mathcal{I}$ of Π and $\hat{S}(\Pi)$ be the total flowtime on instance $\hat{\mathcal{I}}$ of Π. Then, for each feasible sequence, $S(\Pi) = \hat{S}(\Pi) + \sum_{\forall j} t_{2,j}$.*

Proof. Let us consider the PFSP with two machines to minimise the total flowtime, i.e. $\sum_{\forall j} C_{2j}$. In view of Property 5.3.1, $\sum_{\forall j} C_{2j} = \sum_{\forall j}(C_{1j} + t_{2j}) = \sum_{\forall j} C_{1j} + \sum_{\forall j} t_{2j} = \sum_{\forall j} C_{1j} + C$ where d is a constant. □

Then, the minimisation of total flowtime in the second machine ($\sum_{\forall j} C_{2j}$), goal of the $F2|prmu|\sum C_j$, is equivalent to the minimisation of total flowtime on the first machine ($\sum_{\forall j} C_{1j}$) which is the goal of the $1||\sum C_j$ problem of the first machine.

Corollary 5.3.5. *Under the conditions of Theorem 5.3.2, the optimal solution for $\mathcal{I}$ and $\hat{\mathcal{I}}$ is the same where an optimal solution is obtained ordering the jobs according to the non-decreasing processing times on the first machine.*

Proof. The proof of the theorem is obvious in view of Theorem 5.3.2 and since the non-decreasing sum of the processing times is optimal for the $1||\sum C_j$ problem. □

For $m = 2$ and a more restrictive condition of processing times, this result is found by [72]:

Corollary 5.3.6. *([72]: Theorem 4 for $m = 2$). Let $\mathcal{I}$ be an instance of the $F2|prmu|\sum C_j$ where the machine 2 is type-II-dominated by the machine 1. Then, an optimal solution can be obtained ordering the jobs in ascending order of their processing times on the first machine.*

Proof. The proof of the theorem is obvious in view of Corollary 5.3.5 □

For the case where the second machine is saturated, the following property is needed to prove the equivalence between the problems:

Property 5.3.2. *Let $\Pi := (\pi_1, \cdots, \pi_k, \cdots, \pi_n)$ be a sequence of jobs with $t_{2\pi_{k-1}} \geq t_{1\pi_k}, \forall k \geq 2$. Then, the completion time of each job, π_k, on the second machine is equal to the its processing time plus the completion time of the previous job, π_{k-1}, on the second machine machine with the exception of the first job, i.e. $C_{2\pi_k} = C_{2\pi_{k-1}} + t_{2\pi_k}$, $\forall k \geq 2$, and $C_{2\pi_1} = t_{1\pi_1} + t_{2\pi_1}$.*

Proof. The proof of the property is obvious using the same reasoning as in Property 5.3.1. □

This property extends the results by [72] and [120], but the opposite cannot be asserted.

Corollary 5.3.7. *([120], second part of Theorem 3 for $m = 2$; and [72], Lemma 1 for $m = 2$). Let $\mathcal{I}$ be an instance of the PFSP where the machine 1 is type-II-dominated by the machine 2. Then, the completion time of each job on the second machine is equal to the its processing time plus the completion time of the previous job (on the second machine machine), with the exception of the first job in the sequence which is equal to the sum of the processing times of this job on both machines.*

Proof. The proof of the corollary is obvious in view of Property 5.3.2. □

Corollary 5.3.8. *Let* $\Pi := (\pi_1, \cdots, \pi_k, \cdots, \pi_n)$ *be a sequence of jobs with* $t_{1\pi_k} \leq t_{2\pi_{k-1}}, \forall k \geq 2$. *Then, the idle time* $it_{2\pi_k}$ *is equal to 0, i.e.* $it_{2\pi_k} = 0$, $\forall k \in 2 \ldots n$.

Proof. The proof of the corollary is obvious in view of Property 5.3.2 just by taking into account the definition of idle time given in Equation (5.5). □

Then, when the first job of the sequence is known, the equivalence between the $F2|prmu|C_{max}$ and $1||C_{max}$ is established in Theorem 5.3.3.

Theorem 5.3.3. *Let* $\mathcal{I}$ *be an instance of the* $F2|prmu|C_{max}$ *where machine 1 is dominated type I, and* $\hat{\mathcal{I}}$ *be an instance of the* $1||C_{max}$ *problem where* $\hat{t}_j = t_{2j}$. *Let* Π *be a sequence of the form* $\Pi := \{f, \sigma\} := (f, \sigma_1, \cdots, \sigma_k, \cdots, \sigma_{n-1})$ *where* f *is the first job of the sequence and* σ *is an unknown sequence of* $n-1$ *jobs. Let* C_{max} *be the makespan on instance* $\mathcal{I}$ *of* Π *and* $\hat{C}_{max}$ *be the makespan on instance* $\hat{\mathcal{I}}$ *of* Π. *Then, for each feasible sequence,* $C_{max} = \hat{C}_{max} + t_{1,f}$.

Proof. The proof of the theorem is obvious in view of Property 5.3.2 and Corollary 5.3.8, or using the reversibility property of the $Fm|prmu|C_{\max}$ problem. □

Note that a consequence of this theorem is that the optimal solution of both problems is identical.

Corollary 5.3.9. *Under the conditions of Theorem 5.3.3, the optimal solution for* $\mathcal{I}$ *and* $\hat{\mathcal{I}}$ *is the same. Additionally, any sequence of the form* $\Pi := \{f, \sigma\}$ *is optimal for both instances where* f *is the job with the least processing time on the second machine, i.e.* $t_{1,f} = \min_{\forall j} t_{1,j}$.

Proof. The proof of the theorem is obvious in view of Theorem 5.3.3 and since each feasible solution is optimal for the $1||C_{max}$ problem. □

A similar result is found by [72] for both $m = 2$ and a more restrictive condition, but the opposite cannot be asserted.

Corollary 5.3.10. *([72]: Theorem 1 for* $m = 2$*). Let* $\mathcal{I}$ *be an instance of the* $F2|prmu|C_{max}$ *where the machine 1 is type-II-dominated by the machine 2. Then, any sequence of the form* $\Pi := \{f, \sigma\}$ *is optimal where* σ *is any sequence of* $n-1$ *jobs and* f *satisfies* $t_{1f} = \min_{\forall j} t_{1j}$.

Proof. The proof of the theorem is obvious in view of Corollary 5.3.9 □

Additionally, the equivalence between the $F2|prmu|\sum C_j$ and $1||\sum C_j$ is established in Theorem 5.3.4 for the case of a fixed first job in the sequence.

Theorem 5.3.4. *Let $\mathcal{I}$ be an instance of the $F2|prmu|\sum C_j$ where machine 1 is dominated type I, and $\hat{\mathcal{I}}$ be an instance of the $1||\sum C_j$ problem where $\hat{t}_j = t_{2j}$. Let Π be a sequence of the form $\Pi := \{f, \sigma\} := (f, \sigma_1, \cdots, \sigma_k, \cdots, \sigma_{n-1})$ where f is the first job of the sequence and σ is an unknown sequence of $n-1$ jobs. Let $S(\Pi)$ be the total flowtime on instance $\mathcal{I}$ of Π and $\hat{S}(\Pi)$ be the total flowtime on instance $\hat{\mathcal{I}}$ of Π. Then, for each feasible sequence, $S(\Pi) = \hat{S}(\Pi) + n \cdot t_{1,f}$.*

Proof. The proof of the theorem is obvious in view of Property 5.3.2 and Corollary 5.3.8. □

Corollary 5.3.11. *Under the conditions of Theorem 5.3.4 and considering f as a fixed job on the first sequence position, an optimal schedule is obtained ordering the remaining jobs (sequence σ) according to the non-decreasing processing times on the second machine.*

Proof. The proof of the theorem is obvious in view of Theorem 5.3.4 and since each feasible solution is optimal for the $1||\sum C_j$ problem. □

For a more restrictive condition, the same result is found by [72].

Corollary 5.3.12. *([72]: Theorem 3 for $m = 2$). Let $\mathcal{I}$ be an instance of the $F2|prmu|C_{max}$ where the machine 1 is type-II-dominated by the machine 2. Then, an optimal schedule $\Pi := \{f, \sigma\}$, where f is a fixed job on the first sequence position, is obtained ordering the remaining jobs (sequence σ) according to the non-decreasing processing times on the second machine.*

Proof. The proof of the theorem is obvious in view of Corollary 5.3.11 □

Note that the conditions to reach the equivalence between the PFSP and the SMSP to minimise makespan and total flowtime can be reduced when initial availabilities are considered, see Theorem 5.3.5. In this case, both problems are equivalent regardless the sequence of jobs when the conditions are fulfilled.

Theorem 5.3.5. *Let $\mathcal{I}$ be an instance of the PFSP where machine 1 is dominated type I, and $\hat{\mathcal{I}}$ be an instance of the SMSP problem where $\hat{t}_j = t_{2j}$. Let a_2 be the initial availability of the second machine on both instances. Let δ_j the difference between the processing time of job j on the first and second machines i.e. $\delta_j = t_{1j} - t_{2j}$. If*

$$a_2 \geq \sum_{\forall \delta_j > 0} \delta_j + \max_{\forall j}\{t_{1j}\} \tag{5.6}$$

Then, the completion time of each job, π_k, $\forall k > 1$, in the second machine is equal to the its processing time plus the completion time of the previous job or, analogously, the idle time before job π_k is always equals to 0, *Proof.* According to the definition of idle time, Expression (5.5), an idle time equals to 0 implies that $C_{2,\pi_{k-1}} \geq C_{1,\pi_k}$, $\forall k$.

For $k = 1$ (the first job in the sequence), the expression is $C_{2,\pi_0} \geq C_{1,\pi_1} \longrightarrow a_2 \geq t_{1,\pi_1}$ which is satisfied attending to Expression 5.6.

For $k = 2$, $C_{1,\pi_2} = C_{1,\pi_1} + t_{1,\pi_2} = t_{1,\pi_1} + t_{1,\pi_2}$ and $C_{2,\pi_1} = a_2 + t_{2,\pi_1}$ as $it_{i,\pi_1} = 0$. Then, $C_{2,\pi_1} \geq C_{1,\pi_2} \longrightarrow a_2 + t_{2,\pi_1} \geq t_{1,\pi_1} + t_{1,\pi_2} \longrightarrow a_2 \geq t_{1,\pi_1} + t_{1,\pi_2} - t_{2,\pi_1} \longrightarrow a_2 \geq t_{1,\pi_2} + \delta_{\pi_1}$. This condition is fulfilled according to Expression 5.6.

Analogously, for a generic $k = l$, the condition to reach an idle time equals to zero before π_l is $a_2 \geq t_{1,\pi_2} + \sum_{j\in[1,l]} \delta_{\pi_j}$. Since $\max_{\forall j}\{t_{1j}\} + \sum_{\forall \delta_j > 0} \delta_j \geq t_{1,\pi_2} + \sum_{j\in[1,l]} \delta_{\pi_j}$ and according to Expression (5.6), the previous condition is always satisfied. □

PFSP with m machines

Similar results of the equivalence between the PFSP and the SMSP can be found for a flowshop factory with more than two machines. In this section, we detect four possible requirement to be fulfilled by an instance in order to achieve the equivalence between both problems.

Case *ddm*

Let us define some properties and corollaries, before beginning analysing the equivalence between the problems for the *ddm* dominance case.

Property 5.3.3. *Let $\Pi := (\pi_1, \cdots, \pi_k, \cdots, \pi_n)$ be a sequence of jobs with $t_{i,\pi_k} \geq t_{i+1,\pi_{k-1}}, \forall k \geq 2$ and $\forall i > 1$. Then, the completion time of job π_k on the last machine equals its completion time on the first machine plus the sum of the processing times on the rest of machines, i.e.:*

$$C_{m,\pi_k} = C_{1,\pi_k} + \sum_{i=2}^{m} t_{i,\pi_k}, \forall k$$

or equivalently:

$$C_{m,\pi_k} = \sum_{j=1}^{j-1} t_{1,\pi_j} + \sum_{i=1}^{m} t_{i,\pi_k}, \forall k$$

Proof. The proof of the property is obvious applying recursively Property 5.3.1. □

The same result is also found by e.g. [11] and [203] for a more restrictive condition of dominance:

Corollary 5.3.13. *([11], Corollary 3.3; and [203], Observation 2). Let $\mathcal{I}$ be an instance of the PFSP where each machine $i+1$ is type-II-dominated by machine i, $\forall i$. Then, the completion time of a job on* *the last machine is defined by:*

$$C_{m,\pi_k} = \sum_{j=1}^{j-1} t_{1,\pi_j} + \sum_{i=1}^{m} t_{i,\pi_k}, \forall k$$

Proof. The proof of the corollary is obvious in view of Property 5.3.3. □

Then, for this case of dominance between machines, the equivalence between $Fm|prmu|\sum C_j$ and $1||\sum C_j$ is defined in Theorem 5.3.7 as well as the equivalence between the $Fm|prmu|C_{\max}$ and $1||C_{max}$, when the last job of the sequence is fixed, is established in Theorem 5.3.6.

Theorem 5.3.6. *Let $\mathcal{I}$ be an instance of the $Fm|prmu|C_{\max}$ where the machines are dominated according to ddm, and $\hat{\mathcal{I}}$ be an instance of the $1||C_{max}$ problem where $\hat{t}_j = t_{1,j}$. Let Π be a sequence of the form $\Pi := \{\sigma, g\} := (\sigma_1, \cdots, \sigma_k, \cdots, \sigma_{n-1}, g)$ where g is the last job of the sequence and σ is an unknown sequence of $n-1$ jobs. Let C_{max} be the makespan on instance $\mathcal{I}$ of Π and $\hat{C}_{max}$ be the makespan on instance $\hat{\mathcal{I}}$ of Π. Then, for each feasible sequence, $C_{max} = \hat{C}_{max} + \sum_{i=2}^{m} t_{i,g}$.*

Proof. The proof of the theorem is obvious in view of Property 5.3.3 and Theorem 5.3.1. □

Theorem 5.3.7. *Let $\mathcal{I}$ be an instance of the $Fm|prmu|\sum C_j$ where the machines are dominated according to ddm, and $\hat{\mathcal{I}}$ be an instance of the $1||\sum C_j$ problem where $\hat{t}_j = t_{1j}$. Let Π be a sequence of the form $\Pi := (\pi_1, \cdots, \pi_k, \cdots, \pi_n)$. Let $S(\Pi)$ be the total flowtime on instance $\mathcal{I}$ of Π and $\hat{S}(\Pi)$ be the total flowtime on instance $\hat{\mathcal{I}}$ of Π. Then, for each feasible sequence, $S(\Pi) = \hat{S}(\Pi) + \sum_{\forall j, i>1} t_{i,j}$.*

Proof. The proof of the theorem is obvious in view of Property 5.3.3 and Theorem 5.3.2. □

Case *idm*

For the *idm* case, the following property establishes the value of the completion time of each job on the last machine:

Property 5.3.4. *Let $\Pi := (\pi_1, \cdots, \pi_k, \cdots, \pi_n)$ be a sequence of jobs with $t_{i,\pi_{k-1}} \geq t_{i-1,\pi_k}, \forall k \geq 2$ and $\forall i > 1$. Then, the completion time of each job, π_k, on the last machine is equal to its processing time plus the completion time of the previous job, π_{k-1}, on the last machine machine, with the exception of the first job, i.e. $C_{m,\pi_k} = C_{m\pi_{k-1}} + t_{m,\pi_k}$, $\forall k \geq 2$, and $C_{m,\pi_1} = \sum_{\forall i} t_{i,\pi_1}$. Equivalently,*

$$C_{m\pi_k} = \sum_{i=1}^{m-1} t_{i,\pi_1} + \sum_{j=1}^{k} t_{m,\pi_j}, \forall k$$

Proof. The proof of the property is obvious applying recursively Property 5.3.2. □

For more restrictive conditions, the same result is found by e.g. [11], [72] and [203].

Corollary 5.3.14. *([72], Lemma 1; [11], Corollary 3.1; and [203], Observation 1). Let $\mathcal{I}$ be an instance of the PFSP where each machine i is type-II-dominated by machine $i+1$. Then, the completion time of a job on the last machine is defined by:*

$$C_{m,\pi_k} = \sum_{i=1}^{m-1} t_{i,\pi_1} + \sum_{j=1}^{k} t_{m,\pi_j}, \forall k$$

Proof. The proof of the corollary is obvious in view of Property 5.3.4. □

Additionally, the Property 5.3.4 implies that there is no idle time on the last machine after the fist job of the sequence:

Corollary 5.3.15. *Let $\Pi := (\pi_1, \cdots, \pi_k, \cdots, \pi_n)$ be a sequence of jobs and $\mathcal{I}$ be an instance of the PFSP where the machines are dominated according to idm. Then, the idle time IT_{m,π_k} is equal to 0, $\forall k > 1$.*

Proof. The proof of the corollary is obvious in view of Property 5.3.4 just by taking into account the definition of idle time given in Equation (5.5). □

The equivalence between the $Fm|prmu|C_{max}(\sum C_j)$ and $1||C_{max}(\sum C_j)$, when the first job of the sequence is fixed, is established in Theorem 5.3.8 (5.3.9).

Theorem 5.3.8. *Let $\mathcal{I}$ be an instance of the $Fm|prmu|C_{\max}$ where the machines are dominated according to idm, and $\hat{\mathcal{I}}$ be an instance of the $1||C_{max}$ problem where $\hat{t}_j = t_{m,j}$. Let Π be a sequence of the form $\Pi := \{f, \sigma\} := (f, \sigma_1, \cdots, \sigma_k, \cdots, \sigma_{n-1})$ where f is the first job of the sequence and σ is an unknown sequence of $n-1$ jobs. Let C_{max} be the makespan on instance $\mathcal{I}$ of Π and $\hat{C}_{max}$ be the makespan on instance $\hat{\mathcal{I}}$ of Π. Then, for each feasible sequence, $C_{max} = \hat{C}_{max} + \sum_{i=1}^{m-1} t_{i,f}$.*

Proof. The proof of the theorem is obvious in view of Property 5.3.4 and Corollary 5.3.15. □

Theorem 5.3.9. *Let $\mathcal{I}$ be an instance of the $Fm|prmu|\sum C_j$ where the machines are dominated according to idm, and $\hat{\mathcal{I}}$ be an instance of the $1||\sum C_j$ problem where $\hat{t}_j = t_{m,j}$. Let Π be a sequence of the form $\Pi := \{f, \sigma\} := (f, \sigma_1, \cdots, \sigma_k, \cdots, \sigma_{n-1})$ where f is the first job of the sequence and σ is an unknown sequence of $n-1$ jobs. Let $S(\Pi)$ be the total flowtime on instance $\mathcal{I}$ of Π and $\hat{S}(\Pi)$ be the total flowtime on instance $\hat{\mathcal{I}}$ of Π. Then, for each feasible sequence, $S(\Pi) = \hat{S}(\Pi) + n \cdot \sum_{i=1}^{m-1} t_{i,f}$.*

Proof. The proof of the theorem is obvious in view of Property 5.3.4 and Corollary 5.3.15. □

Case *idm-ddm*

Similar to the previous case, the completion time of each job on the last machine is defined by the following property for the *idm-ddm* case:

Property 5.3.5. *Let $\Pi := (\pi_1, \cdots, \pi_k, \cdots, \pi_n)$ be a sequence of jobs with $t_{i_1,\pi_{k-1}} \geq t_{i_1-1,\pi_k}, \forall k \geq 2, s \geq i_1 > 1$ and $t_{i_2,\pi_k} \geq t_{i_2+1,\pi_{k-1}}, \forall k \geq 2, i_2 \geq s$, i.e. following a dominance configuration type idm-ddm where the saturated machine is s. Then, the completion time of each job, π_k, on the last machine is:*

$$C_{m\pi_k} = \sum_{i_1=1}^{s-1} t_{i_1,\pi_1} + \sum_{j=1}^{k} t_{s,\pi_j} + \sum_{i_2=s+1}^{m} t_{i_2,\pi_k}, \forall k$$

Proof. The proof of the property is obvious in view of Properties 5.3.3 and 5.3.4. □

For a more restrictive condition, the same result is found by [203].

Corollary 5.3.16. *([203], Observation 4). Let $\mathcal{I}$ be an instance of the PFSP where each machine $i_1 < s$ is type-II-dominated by machine $i_1 + 1$ as well as each machine $s < i_2 \leq m$ is type-II-dominated by machine $i_2 - 1$. Then, the completion time of a job on the last machine is defined by:*

$$C_{m\pi_k} = \sum_{i_1=1}^{s-1} t_{i_1,\pi_1} + \sum_{j=1}^{k} t_{s,\pi_j} + \sum_{i_2=s+1}^{m} t_{i_2,\pi_k}, \forall k$$

Proof. The proof of the corollary is obvious in view of Property 5.3.5. □

Then, for this case of dominance between machines, the equivalence between $Fm|prmu|\sum C_j$ and $1||\sum C_j$ is defined in Theorem 5.3.11 fixing the first job of the sequence as well as the equivalence between the $Fm|prmu|C_{\max}$ and $1||C_{max}$, when the first and last job of the sequence is fixed, is established in Theorem 5.3.10.

Theorem 5.3.10. *Let $\mathcal{I}$ be an instance of the $Fm|prmu|C_{\max}$ where the machines are dominated according to idm-ddm, and $\hat{\mathcal{I}}$ be an instance of the $1||C_{max}$ problem where $\hat{t}_j = t_{sj}$.Let Π be a sequence of the form $\Pi := \{f, \sigma, g\} := (f, \sigma_1, \cdots, \sigma_k, \cdots, \sigma_{n-2}, g)$ where f and g are respectively the first and the last job of the sequence and σ is an unknown sequence of $n-2$ jobs. Let $S(\Pi)$ be the total flowtime on instance $\mathcal{I}$ of Π and $\hat{S}(\Pi)$ be the total flowtime on instance $\hat{\mathcal{I}}$ of Π. Then, for each feasible sequence,*

$$C_{max} = \hat{C}_{max} + \sum_{i_1=1}^{s-1} t_{i_1,f} + \sum_{i_2=s+1}^{m} t_{i_2,g}$$

Proof. The proof of the theorem is obvious in view of Property 5.3.5. □

Theorem 5.3.11. *Let $\mathcal{I}$ be an instance of the $Fm|prmu|\sum C_j$ where the machines are dominated according to idm-ddm, and $\hat{\mathcal{I}}$ be an instance of the $1||\sum C_j$ problem where $\hat{t}_j = t_{sj}$.Let Π be a sequence of the form $\Pi := \{f, \sigma\} := (f, \sigma_1, \cdots, \sigma_k, \cdots, \sigma_{n-1})$ where f is respectively the first job of the sequence and*

σ is an unknown sequence of $n-1$ jobs. Let C_{max} be the makespan on instance $\mathcal{I}$ of Π and $\hat{C}_{max}$ be the makespan on instance $\hat{\mathcal{I}}$ of Π. Then, for each feasible sequence,

$$S(\Pi) = \hat{S}(\Pi) + \sum_{\forall j, i_2 > s} t_{i_2,j} + n \cdot \sum_{i_1=1}^{s-1} t_{i_1,f}$$

Proof. The proof of the theorem is obvious in view of Property 5.3.5. □

Generic Case

The assumptions to achieve this equivalence are much harder in the generic case of a factory with $m > 2$ machines. In fact, it is necessary that the processing time of each job j on the saturated machine s is higher than both the sum of the processing times on the machines before s of each job $j^{'} \neq j$, and the sum of the processing times on the machines after s of each job $j^{'} \neq j$. Obviously, this behaviour is hardly found in real-life environments. It thus represents only a sufficient but not necessary condition to state the equivalence.

Theorem 5.3.12. *Let $\mathcal{I}$ be an instance of the $Fm|prmu|C_{\max}$ with $t_{sj} \geq \sum_{i<s} t_{ij'}$ and $t_{sj} \geq \sum_{i>s} t_{ij'}$, $\forall j \neq j^{'}$, and $\hat{\mathcal{I}}$ be an instance of the $1||C_{max}$ problem where $\hat{t}_j = t_{sj}$. Let f and g be the fixed first and last job of a sequence of jobs $\Pi := (f, \sigma_1, \cdots, \sigma_k, \cdots, \sigma_{n-2}, g)$ where σ is an unknown sequence of $n-2$ jobs. Then, the $Fm|prmu|C_{\max}$ is equivalent to the $1||C_{max}$ of machine s.*

Proof. The proof is obvious using the same reasoning as Properties 5.3.1 and 5.3.2. □

Theorem 5.3.13. *Let $\mathcal{I}$ be an instance of the $Fm|prmu|\sum C_j$ with $t_{sj} \geq \sum_{i>s} t_{ij'}$ and $t_{sj} \geq \sum_{i>s} t_{ij'}$, $\forall j \neq j^{'}$, and $\hat{\mathcal{I}}$ be an instance of the $1||\sum C_j$ problem where $\hat{t}_j = t_{sj}$. Let f and g be the fixed first and last job of a sequence of jobs $\Pi := (f, \sigma_1, \cdots, \sigma_k, \cdots, \sigma_{n-2}, g)$ where σ is an unknown sequence of $n-2$ jobs. Then, the $Fm|prmu|\sum C_j$ is equivalent to the $1||\sum C_j$ of machine s.*

Proof. The proof is obvious using the same reasoning as Properties 5.3.1 and 5.3.2. □

All the properties presented in Sections 5.3 and in this section analyse the assumptions required to theoretically prove the equivalence between the PFSP and the SMSP of the most saturated machine, under the minimisation of makespan and total flowtime. The equivalence between both scheduling problems is theoretically proved in the Theorems 5.3.1, 5.3.2, 5.3.3, 5.3.4, 5.3.5, 5.3.7, 5.3.7, 5.3.8, 5.3.9, 5.3.10, 5.3.11, 5.3.12 and 5.3.13 for different conditions. Given an instance, the fulfillment of these conditions then indicates that solving the equivalent SMSP is analogous as solving the original PFSP. However, this equivalence could be approximately satisfied under milder conditions. In the next section, we empirically analyse them by means of an extensive computational experience.

Computational analysis

In this section we consider the following issues: the procedure to generate the instances; the description of the implemented heuristics; the results for several values of the parameters of the testbed; and finally, the boundary lines between both problems.

Testbed generation

Using the above formulations and definitions, existing testbeds for the PFSP can be analysed. As mentioned in the previous section, most algorithms for the PFSP problem have been tested on benchmarks where the processing times follow a uniform distribution. However, in the experiments by [206], a structured benchmark with job-correlated, machine-correlated and mixed-correlated processing times is employed. Regarding machine-correlation, processing times are generated using a uniform distribution considering the following two aspects:

- For each machine, the mean of the processing times is generated from 1 to 100 depending on a parameter.
- For each machine, the width of the uniform distribution is uniformly generated from 2 to 10.

The goal of this section is to show that the problem can be reduced to a SMSP when there is a machine saturated in the PFSP. The study of this equivalence between the problems must obviously be done without the influences of another effects and therefore, a specific benchmark is generated to test the experiments performed in Section 11.4, where different levels for the saturated machine are considered. Firstly, the same distribution is considered for each non-saturated machine since it is the worst case. Additionally, the consideration of different distributions in the non-saturated machines would strongly hinder the understanding on the cause of the equivalence between the problems. Secondly, the same width of the uniform distribution (equivalently variance) is considered for all machines (including the most saturated one) both for clarity and to reduce the parameters of the proposed benchmark.

Taking into account the previous discussion, the processing times for our computational experience are then generated according to the expression (5.7) for the non-saturated machines and (5.8) for the saturated machine (denoted as s):

$$t_{ij} \to U[\epsilon \cdot (1-\beta), \epsilon \cdot (1+\beta)], \forall i \neq s, j \in 1, \dots, n \tag{5.7}$$

$$t_{sj} \to U[\epsilon \cdot (1+\gamma-\beta), \epsilon \cdot (1+\gamma+\beta)], j \in 1, \dots, n \tag{5.8}$$

where the following parameters must be defined:

- ϵ: Mean of the processing times on non-saturated machines.
- β: Half length of the interval of the uniform distribution of each machine with respect to the mean processing time ϵ, i.e. $\epsilon\beta$ yields the half width of the interval, and $2 \cdot \epsilon \cdot \beta$ is the full length of the interval of the uniform distribution of each machine (including the saturated machine).
- γ: Increase of the mean processing times on the saturated machine s relative to ϵ. In this way, $(1+\gamma)\epsilon$ represents the expected processing time on machine s whereas the expected value of the processing time for the rest of the machines is ϵ.

Regarding the initial availability of the machines, the following parameter must be considered in the benchmark:

- δ: Number of jobs being processed in the shop floor to create a fictitious initial unavailability. More specifically, we generate δ jobs, which are sequenced according to certain heuristic (in this Thesis, we use the NEH of [127] and [48] for makespan and flowtime minimisation respectively). The processing of these jobs according to such sequence creates a_i, the initial unavailability in each machine i, which can be computed as follows: $a_i = C_{i,\pi_n} - C_{1,\pi_n}$.

Implemented heuristics

The following simple 11 algorithms are implemented to analyse the relationship between SMSP and PFSP problems. More specifically, we will design the following procedures:

- PF_B(MK) and PF_B(FT) will be designed to provide good –hopefully best– solutions for the PFSP problems with makespan and flowtime objective, respectively.
- PF_W(MK) and PF_W(FT) will be designed to obtain bad –hopefully worst– solutions for the PFSP problems with makespan and flowtime objective, i.e. they seek makespan and flowtime *maximization*.
- SM_R(MK) and SM_B(FT) will be designed to provide the best solutions for the equivalent SMSP problems if only the saturated machine in the PFSP is considered.
- SM_W(FT) will be designed to provide bad –hopefully worst– solutions for the equivalent SMSP problem if only the saturated machine in the PFSP is considered. Note that the corresponding worst procedure for makespan would be also SM_R(MK).

- SM_EB(MK) and SM_EB(FT) will be designed to provide good –hopefully best– solutions for the equivalent SMSP problems considering the saturated machine in the PFSP and the influence of the last and first jobs in this machine.
- M_B(MK) and M_B(FT) will be designed to provide good –hopefully best– solutions for a reduced PFSP formed by machines $i^{'} \in \{s, m\}$ for makespan and flowtime objectives, respectively.

With these procedures we can check the (statistical) equivalence of SMSP and PFSP problems on a set of instances, since, for the cases where the objective function values found by SM_B(FT), SM_R(MK), SM_EB(i) and M_B(i) are close to those provided by PF_B(i), and those found by SM_W(FT) and PF_W(i) are similar, then both problems (PFSP and SMSP) are (approximately) equivalent. In order to implement these procedures, the following decisions have been taken:

- SM_B(FT): Jobs are sorted in non-decreasing order of their processing times on machine s, which corresponds to the optimal solution of the equivalent $1||\sum C_j$ problem.
- SM_W(FT): Jobs are sorted in non-increasing processing times on machine s, which is expected to provide a bad solution for the equivalent $1||\sum C_j$ problem.
- SM_R(MK): Jobs are sorted according to sequence $(1, \ldots, n)$, which is a random solution for the $1||C_{max}$ problem. Since, for this problem, each solution is optimal, this procedure would yield both the best and worst solutions for the problem.
- SM_EB(MK): Since the idea is to take into account the influence of the first and last jobs of the sequence on the non-saturated machines, the procedure reduces the $Fm|prmu|C_{\max}$ to a problem similar to the SMSP problem where the first and last jobs add the processing times on the machines before and after the saturated machine s to their processing times, i.e. given a sequence Π of jobs, the processing times are:

$$t^{'}_{i\pi_k} = \begin{cases} t_{i\pi_k} + \sum_{i=1}^{s-1} t_{i\pi_k}, & k = 1 \\ t_{i\pi_k}, & \forall k \neq 1, n \\ t_{i\pi_k} + \sum_{i=s}^{m} t_{i\pi_k}, & k = n \end{cases}$$

  To find a good Π^f final sequence, as in the SM_R(MK), jobs are first sorted randomly (let us denoted Π^R to this sequence). Then, two simple phases are carried out as follows to find the first and last job:

  - In case of $\delta = 0$, the first job of the sequence is the job with minimal sum of processing times before machine s, i.e. π_1^f is the job F satisfying that $\sum_{i=1}^{s-1} t_{iF} \leq \sum_{i=1}^{s-1} t_{ij} \forall j$. In case of $\delta > 0$, the first job is the same as in the SM_R(MK) heuristic, $\pi_1^f = \pi_1^R$.

  - The last job of the sequence is the job with minimal sum of processing times after machine s respectively, i.e. π_n^f is the job L which satisfies that $\sum_{i=s+1}^m t_{iL} \leq \sum_{i=s+1}^m t_{ij}$, $\forall j$.

- SM_EB(FT): Designed for flowtime, SM_EB(FT) solves the equivalent SMSP problem as the SM_B(FT). However, in contrast to that heuristic, the SM_EB(FT) considers the influence of the first job in the machines before the non-saturated machine, i.e. $i < s$. Thereby, the processing time of the first job is the sum of the processing times of the machines $i \leq s$, i.e. given a sequence Π of jobs, the processing times are:

$$t'_{i\pi_k} = \begin{cases} t_{i\pi_k} + \sum_{i=1}^{s-1} t_{i\pi_k}, & k = 1 \\ t_{i\pi_k}, & \forall k > 1 \end{cases}$$

  The procedure of the heuristic consists of two phase: first phase is the same heuristic SM_B(FT) where jobs are ordered according to non-decreasing processing times on machine s; then, the first job of the sequence is the job with minimal sum of processing times until machine s, i.e. π_1^f is the job F which satisfies that $\sum_{i=1}^s t_{iF} \leq \sum_{i=1}^s t_{ij} \forall j$.

- PF_B(i) ($i \in [MK, FT]$): Since the idea is to provide good solutions for $Fm|prmu|\sum C_j$ and $Fm|prmu|C_{\max}$, we use the NEH heuristic of [127] and [48], respectively.

- PF_W(i) ($i \in [MK, FT]$): Since the idea is to provide bad solutions for $Fm|prmu|\sum C_j$ and $Fm|prmu|C_{\max}$, we use the NEH heuristic for makespan and total flowtime *maximisation.*

- M_B(i) ($i \in [MK, FT]$): These heuristics use the same NEH heuristics to solve a reduced PFSP considering only machines $i' \in \{s, m\}$. The operations of the jobs in the other machines are omitted. Note that the initial availabilities, a_i, are calculated from the saturated machine, i.e. $a_{i'} = C_{i',\pi_n} - C_{s,\pi_n}$, $\forall i' \in \{s, m\}$.

Evaluation of the solutions

Traditionally, the related literature has used Relative Percentage Deviation (RPD) and the CPU time to measure both the quality of the solution and the required computational effort of heuristic r in an instance s. More specifically, the average RPD (ARPD) and the average CPU time (ACPU) obtained over a set of S instances can be defined as follows:

$$ARPD_r = \frac{\sum_{\forall s} RPD_{rs}}{S} \tag{5.9}$$

$$ACPU_r = \frac{\sum_{\forall s} T_{rs}}{S} \tag{5.10}$$

where

$$RPD_{rs} = \frac{OFV_{rs} - Best_s}{Best_s} \cdot 100 \tag{5.11}$$

OFV_{rs} is the objective function value (makespan or total flowtime) obtained by heuristic r in instance s. $Best_s$ is the best solution among the implemented heuristics in that instance, i.e. $Best_s := \min_r OFV_{rs}$. Finally, T_{rs} is the CPU time of heuristic r in instance s.

The consideration of initial availability introduces a disruption in the evaluation of the objective function which must be taken into account. This disruption is illustrated with the following example: Let us assume a PFSP problem with two machines and two jobs. Processing times of the first and second jobs on the machines are $t_{11} = 10$, $t_{21} = 40$, and $t_{12} = 10$, $t_{22} = 50$ respectively. For the two possible sequences i.e. $\pi^1 = (1,2)$ and $\pi^2 = (2,1)$, the total flowtimes are $\sum C_{m,\pi^1_j} = 150$ and $\sum C_{m,\pi^2_j} = 160$. In terms of RPD, $RPD(\sum C_{m,\pi^1_j}) = 0$ and $RPD(\sum C_{m,\pi^2_j}) = 6.67$.

Let us now assume that the second machine is not available until time 300. Then, the total flowtime of both sequences change to $\sum C_{m,\pi^1_j} = 730$ and $\sum C_m, \pi^2_j = 740$ respectively, while RPD are $RPD(\sum C_{m,\pi^1_j}) = 0$ and $RPD(\sum C_{m,\pi^2_j}) = 1.37$. Although in this case the initial availability of the second machine clearly does not influence the hardness of the problem, its influence on the RPDs is very high.

To avoid this issue, we do not consider the time 0 as reference for the completion times. Instead, we consider a reference (denoted as B) based on the first job of the sequence. Nevertheless, in order not to have a sequence-dependent reference, we consider as the first job of the sequence all jobs and we average theirs completion times, i.e. $B = \frac{\sum_{j=1}^{n} C_{m,\pi_1=j}}{n}$. Once B is obtained, the completion time of each job on the last machine is reduced by B for each heuristic.

Computational results

All algorithms are coded in the same language (C# under Visual Studio 2013) and under an Intel Core i7-3770 with 3.4 GHz and 16 GB RAM. They are tested in an set of instances following the indications of Section 5.3, which includes $n \in \{20, 50, 100, 200\}$, $m \in \{2, 5, 10, 20\}$ and two values of $\delta \in \{0, 100\}$ representing an initially empty and loaded shop respectively. Processing times are generated according to the expression (5.7) and (5.8) with the following parameters:

- $\epsilon = 50$.

- $\beta \in \{0.10, 0.20, 0.40, 0.60, 1.00\}$.
- $\gamma \in \{0.00, 0.04, 0.08, \ldots, 2.96, 3.00\}$.

For each combination of parameters (n, m, δ, ϵ, β and γ), 10 instances are generated forming a total of 121,600 instances.

Values of $ARPD$ of the heuristics with ($\delta = 100$) and without ($\delta = 0$) considering initial availabilities are shown in Table 5.1 for each value of the parameter n, m and β, and for some values of γ. Clearly, the heuristics SM_B(FT), SM_R(MK), SM_EB(i) and M_B(i) go closer to PF_B(i) when the parameters β and m decrease, and γ and n increase. In Figure 5.4 and 5.5 for makespan and total flowtime minimisation respectively, it can be seen the $ARPD$ of the heuristics for the complete set of values of the parameter γ and δ as well as the decreasing tend of each curve. Several aspects can be highlighted from the results:

Table 5.1: ARPD of the heuristics

		ARPD of heuristics for makespan minimisation					ARPD of heuristics for total flowtime minimisation					
		SM_R(MK)	SM_EB(MK)	PF_B(MK)	PF_W(MK)	M_B(MK)	SM_O(FT)	SM_EB(FT)	PF_B(FT)	PF_W(FT)	SM_W(FT)	M_B(FT)
$\delta = 0$	$\gamma = 0.00$	7.84	7.32	0.00	28.64	8.35	11.10	10.51	0.04	35.14	20.11	8.26
	$\gamma = 0.20$	5.48	4.41	0.01	20.88	4.21	7.30	6.65	0.06	31.79	19.50	5.04
	$\gamma = 0.40$	3.80	2.69	0.00	13.95	2.54	4.99	4.30	0.11	27.57	18.19	3.74
	$\gamma = 0.60$	3.09	1.81	0.00	9.68	1.89	3.59	2.79	0.13	23.97	16.69	2.81
	$\gamma = 0.80$	2.63	1.27	0.01	6.93	1.53	2.53	1.76	0.16	21.17	15.21	1.97
	$\gamma = 1.00$	2.20	0.97	0.00	5.28	1.09	1.87	1.10	0.18	18.90	13.80	1.57
	$\gamma = 2.00$	1.30	0.32	0.00	2.62	0.61	1.00	0.21	0.22	12.32	9.36	1.00
	$\beta = 0.10$	0.36	0.06	0.00	0.80	0.18	0.30	0.07	0.08	3.16	2.38	0.28
	$\beta = 0.20$	0.81	0.23	0.00	1.93	0.43	0.72	0.27	0.15	6.85	5.08	0.64
	$\beta = 0.40$	1.82	0.79	0.00	4.77	1.04	1.72	0.96	0.23	14.50	10.45	1.41
	$\beta = 0.60$	2.96	1.56	0.00	8.33	1.83	3.03	2.03	0.25	22.64	16.05	2.42
	$\beta = 1.00$	5.49	3.52	0.00	16.97	3.73	6.30	4.99	0.24	39.74	27.19	4.74
	$m = 2$	0.53	0.13	0.00	2.21	0.16	0.41	0.40	0.04	15.29	12.64	0.21
	$m = 5$	1.70	0.67	0.00	5.61	1.12	1.93	1.27	0.30	16.78	11.84	1.55
	$m = 10$	2.84	1.52	0.00	8.03	1.89	3.07	2.04	0.25	18.09	11.98	2.46
	$m = 20$	4.09	2.61	0.00	10.38	2.58	4.25	2.94	0.17	19.36	12.46	3.38
	$n = 20$	4.63	2.29	0.01	10.19	2.56	4.03	2.34	0.32	19.23	14.60	3.34
	$n = 50$	2.37	1.32	0.00	6.61	1.49	2.49	1.76	0.20	17.29	12.25	1.97
	$n = 100$	1.37	0.83	0.00	5.14	1.01	1.80	1.42	0.14	16.66	11.31	1.34
	$n = 200$	0.78	0.49	0.00	4.30	0.71	1.33	1.14	0.10	16.34	10.75	0.95
$\delta = 100$	$\gamma = 0.00$	3.47	3.32	0.02	16.21	3.36	8.34	8.34	0.11	27.57	16.71	2.61
	$\gamma = 0.20$	3.30	2.71	0.04	14.82	0.64	4.09	4.09	0.12	28.09	17.18	0.74
	$\gamma = 0.40$	2.33	1.69	0.03	10.01	0.07	1.69	1.69	0.09	25.38	16.27	0.26
	$\gamma = 0.60$	1.76	1.11	0.01	6.50	0.03	0.75	0.75	0.06	22.10	14.70	0.14
	$\gamma = 0.80$	1.48	0.79	0.00	4.45	0.02	0.40	0.40	0.06	19.30	13.02	0.07
	$\gamma = 1.00$	1.25	0.51	0.01	3.20	0.01	0.25	0.25	0.04	17.19	11.76	0.04
	$\gamma = 2.00$	0.74	0.14	0.00	1.44	0.00	0.05	0.05	0.00	10.72	7.58	0.00
	$\beta = 0.10$	0.20	0.03	0.00	0.47	0.01	0.04	0.04	0.00	2.74	1.92	0.01
	$\beta = 0.20$	0.46	0.11	0.00	1.16	0.03	0.12	0.12	0.01	5.96	4.14	0.03
	$\beta = 0.40$	1.02	0.41	0.00	2.96	0.09	0.47	0.47	0.02	12.75	8.67	0.11
	$\beta = 0.60$	1.64	0.84	0.01	5.25	0.17	0.98	0.98	0.04	19.94	13.39	0.21
	$\beta = 1.00$	3.09	2.00	0.03	11.12	0.48	2.57	2.57	0.11	35.63	23.41	0.54
	$m = 2$	0.49	0.10	0.00	2.02	0.10	0.40	0.40	0.04	15.09	12.60	0.13
	$m = 5$	0.97	0.38	0.00	3.54	0.13	0.67	0.67	0.02	15.32	10.46	0.19
	$m = 10$	1.48	0.78	0.01	4.85	0.17	0.94	0.94	0.03	15.51	9.42	0.18
	$m = 20$	2.20	1.45	0.02	6.35	0.22	1.33	1.33	0.05	15.70	8.76	0.22
	$n = 20$	2.59	1.26	0.02	6.15	0.15	1.20	1.20	0.05	15.28	10.42	0.18
	$n = 50$	1.31	0.71	0.01	4.20	0.14	0.85	0.85	0.04	15.41	10.33	0.17
	$n = 100$	0.78	0.46	0.00	3.43	0.15	0.70	0.70	0.03	15.42	10.26	0.18
	$n = 200$	0.46	0.29	0.00	2.98	0.18	0.60	0.60	0.02	15.50	10.21	0.18

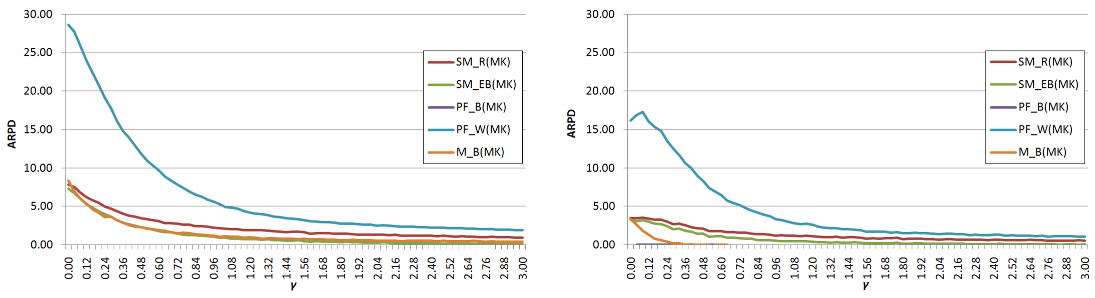

Figure 5.4: ARPD of the heuristics versus parameter gamma for makespan minimisation. On the left, no initial availability is considered and on the right an initial $\delta = 100$ is taken into account.

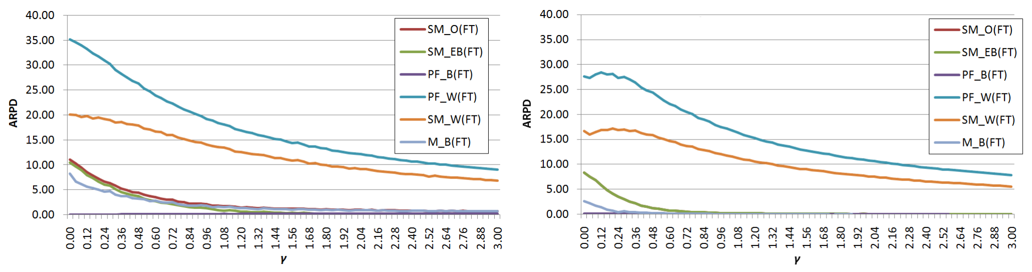

Figure 5.5: ARPD of the heuristics versus parameter gamma for total flowtime minimisation. On the left, no initial availability is considered and on the right an initial $\delta = 100$ is taken into account.

- Regardless the effects of other parameters, for high values of γ, solving the equivalent SMSP problem or the original PFSP yield a similar solution (i.e. ARPD obtained by SM_B(FT), SM_R(MK), SM_EB(i) and SM_B(FT) are very close to the ARPD of PF_B(i)).

- Additionally, for high values of γ, the worst solutions found in the $Fm|prmu|\sum C_j$ problem by PF_W(FT) are also similar to the solutions found for the equivalent $1||\sum C_j$ problem by SM_W(FT).

- The ARPD found by SM_R(MK) for the equivalent $1||C_{max}$ problem is always between the best and worst ARPD found by heuristics PF_B(MK) and PF_W(MK). The distance between the three curves heavily decreases with the increase of γ which explains the trivial behaviour of the $Fm|prmu|C_{\max}$ for those cases.

- The initial availability (δ) has a strong influence over the ARPD of the curves as seen in Figure 5.4 and 5.5. Thereby, e.g. the ARPD of SM_EB(i) goes close to PF_B(i) regardless the other parameters (m, n or β) from γ around 30%.

Attending to the dominance rules of Section 5.3 and 5.3, the number of machines and the bounds of the processing times play an essential role in the comparison between the PFSP and the SMSP problems. This influence is also empirically shown in this section. Thereby, the Figure 5.6 shows the evolution of the ARPD compared with the γ parameter for different number of machines. The ARPD curves clearly

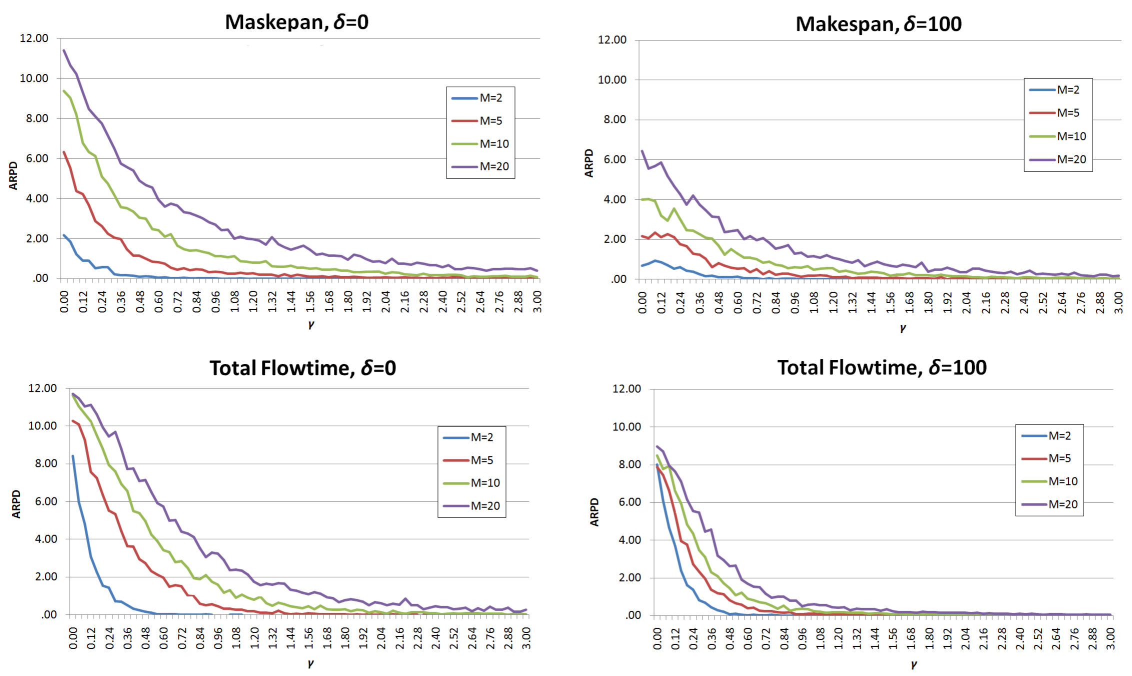

Figure 5.6: ARPD of the SM_EB(i) heuristic for different values of parameter m

decrease with the decrease of the number of the machine in the shop. Note that for clarity only the SM_EB(i) heuristics are represented although the behaviour is also similar for the other heuristics. As the ARPDs for the PF_B(i) heuristics are always approximately zero, then values closes to zero in the SM_EB(i) heuristics indicate the proximity in the solutions found for the original PFSP problem and for the equivalent SMSP problem. Thereby, e.g. it can be seen that the ARPD of the SM_EB(MK) heuristic for $\delta = 100$ is always less than 1, regardless the value of γ.

Regarding the β parameter, its influence over the ARPD is shown in Figure 5.7 for the heuristics SM_EB(i). The curves also present a high decrease in ARPD when the β parameter decrease. In fact, from $\gamma = 8$, the ARPD for the curve $\beta = 10$ is always less than 1 regardless the objective or the value of δ.

Boundary lines between the PFSP and the SMSP

In previous Sections, we have proved the relationship between both scheduling problems and have shown that the ARPDs of several heuristics (designed for the original PFSP and for reduced scheduling problems) tend to be similar for high values of γ, δ and n, and for low values of m and β. In this section, we analyse the conditions which have to be approximately fulfilled in order that the reduced SMSP is (roughly) equivalent to the original PFSP. Firstly, let us consider that both problems are similar when the differences in the ARPDs of the heuristics to solve both problems (i.e. $PF\_B(i)$ and $SM\_EB(i)$) are lesser than 0.5%.

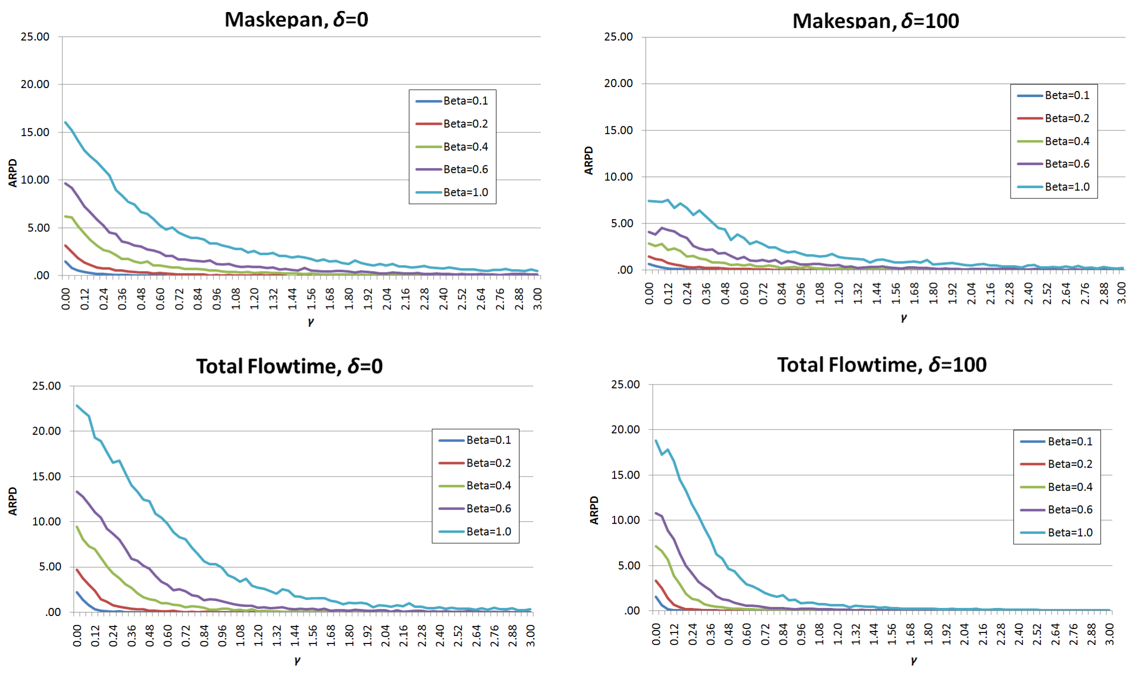

Figure 5.7: ARPD of the SM_EB(i) heuristic for different values of parameter *beta*

The experiments of this Section are carried out under an exhaustive set of 608,000 instances (which contain some more values of the parameter β in comparison with previous testbed):

- $n \in \{20, 50, 100, 200\}$.
- $m \in \{2, 5, 10, 20\}$.
- $\epsilon = 50$.
- $\beta \in \{0.04, 0.08, \ldots, 0.96, 1.00\}$.
- $\gamma \in \{0.00, 0.04, 0.08, \ldots, 2.96, 3.00\}$.
- $\delta \in \{0, 100\}$.

In this set, there are 40 instances with different values of n for each combination of m, β, γ and δ. Let us denote by $ARPD'_{m,\beta,\gamma,\delta}$ the average RPD of those 40 instances for each value of m, β, γ and δ as well as by $\gamma^*_{m,\beta,\delta}$, the first value of γ for which the $ARPD'_{m,\beta,\gamma,\delta} < 0.5$ for the instances with parameters m, β and γ. Values of $\gamma^*_{m,\beta,\delta}$ are graphically shown in Figure 5.8 and 5.9 for makespan and flowtime minimisation respectively. On the left sides of both figures values for $\delta = 0$ are shown while on the right sides values for $\delta = 100$ are shown. Additionally, for each value of m, a linear trend line is represented. Thereby, those lines represent approximately the boundary lines (difference of ARPD less than 0.5%) of

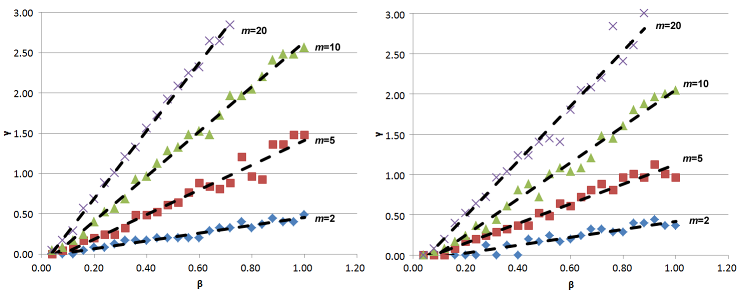

Figure 5.8: Boundary lines between the PFSP and the SMSP for makespan minimisation. On the left, no initial availability is considered and on the right an initial $\delta = 100$ is taken into account.

Table 5.2: Regions where solving PFSP is very similar to the SMSP for makespan minimisation

m	$\delta = 0$	$\delta = 100$
2	$\gamma \geq 0.48 \cdot \beta - 3.08 \cdot 10^{-2}$, $(R^2 = 0.962)$	$\gamma \geq 0.49 \cdot \beta - 7.12 \cdot 10^{-2}$, $(R^2 = 0.890)$
5	$\gamma \geq 1.53 \cdot \beta - 12.12 \cdot 10^{-2}$, $(R^2 = 0.960)$	$\gamma \geq 1.22 \cdot \beta - 9.88 \cdot 10^{-2}$, $(R^2 = 0.972)$
10	$\gamma \geq 2.79 \cdot \beta - 14.64 \cdot 10^{-2}$, $(R^2 = 0.994)$	$\gamma \geq 2.26 \cdot \beta - 20.36 \cdot 10^{-2}$, $(R^2 = 0.984)$
20	$\gamma \geq 4.22 \cdot \beta - 15.58 \cdot 10^{-2}$, $(R^2 = 0.997)$	$\gamma \geq 3.43 \cdot \beta - 20.26 \cdot 10^{-2}$, $(R^2 = 0.976)$

both decision problems for a value of m and δ, i.e. for a given β , m and δ it represents the first value of γ for which the ARPD between the heuristics PF_B(i) and SM_EB(i) is lower than 0.5%. The R^2 of each trend line is mostly close to 0.99. By means of those trend lines, regions with relative similar ARPD between heuristics to solve the PFSP and the reduced SMSP are shown in Table 5.2 and 5.3 for makespan and total flowtime minimisation respectively.

Note that these boundary lines are obviously exact over the proposed set of instances but they are an approximation for other benchmarks or for processing times following different distributions. Thereby, they can be useful for the decision makers to give an idea of solving their manufacturing layouts, since variables γ and β can be easily approximated by an sample of the processing times of the shop. Let $\bar{\mu}_1$ and $\bar{\mu}_2$ be the sample means of the processing times on the saturated machine and non-saturated machine respectively. Additionally, let $\bar{\sigma}_s^2$ be the unbiased sample variance. Then, using the definition of the mean and the variance for the uniform distribution, the following expressions approximate the variables used in this study:

- $\bar{\mu}_1 \simeq \epsilon \longrightarrow \epsilon \simeq \bar{\mu}_1$
- $\bar{\mu}_2 \simeq \epsilon \cdot (1 + \gamma) \longrightarrow \gamma \simeq \frac{\bar{\mu}_2}{\epsilon} - 1$
- $\bar{\sigma}_s^2 \simeq \frac{(2 \cdot \epsilon \cdot \beta + 1)^2 - 1}{12} \longrightarrow \beta \simeq \frac{\sqrt{\bar{\sigma}_s^2 \cdot 12 + 1} - 1}{2 \cdot \epsilon}$

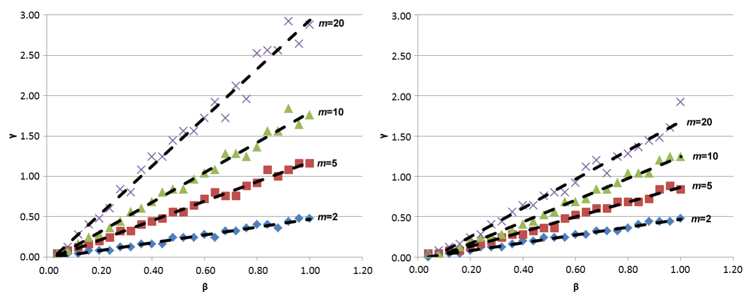

Figure 5.9: Boundary lines between the PFSP and the SMSP for flowtime minimisation. On the left, no initial availability is considered and on the right an initial $\delta = 100$ is taken into account.

Table 5.3: Regions where solving PFSP is very similar to the SMSP for flowtime minimisation

m	$\delta = 0$	$\delta = 100$
2	$\gamma \geq 0.48 \cdot \beta - 1.48 \cdot 10^{-2}$, $(R^2 = 0.974)$	$\gamma \geq 0.49 \cdot \beta - 1.56 \cdot 10^{-2}$, $(R^2 = 0.989)$
5	$\gamma \geq 1.21 \cdot \beta - 3.40 \cdot 10^{-2}$, $(R^2 = 0.991)$	$\gamma \geq 0.92 \cdot \beta - 5.08 \cdot 10^{-2}$, $(R^2 = 0.987)$
10	$\gamma \geq 1.86 \cdot \beta - 6.56 \cdot 10^{-2}$, $(R^2 = 0.989)$	$\gamma \geq 1.35 \cdot \beta - 10.12 \cdot 10^{-2}$, $(R^2 = 0.989)$
20	$\gamma \geq 3.01 \cdot \beta - 7.92 \cdot 10^{-2}$, $(R^2 = 0.984)$	$\gamma \geq 1.79 \cdot \beta - 10.76 \cdot 10^{-2}$, $(R^2 = 0.982)$

5.4 Controllable processing times

Introduction

Properly speaking, resource-dependent processing times have been usually classified depending on the level of skill of the assigned resources, and/or on the amount of resources. In the former case, an operation is performed by a resource (typically an employee) with a given level of skill or experience, and the processing times of that operation is different depending on such level (examples can be found in [87], [36], [33], [70] and [200]). In the latter case, the processing time of an operation changes with the amount of resources assigned to the operation. The term "controllable processing times" has traditionally been used in the literature to reflect this case. This section focuses on this second type. Regarding the type of resources, the classical classification of resources proposed by [8] and [184] is adopted here whereas resources are classified from the viewpoint of renewability and divisibility.

With respect to renewability:

- A resource is denoted as renewable if only its total usage is constrained at every moment, i.e. once a resource has been used by an operation, it may be assigned to another operation.
- A resource is denoted as non-renewable if its total consumption is constrained i.e. once it is consumed, it cannot be allocated to other operation.
- A resource is denoted as doubly constrained if both previous aspects are considered.

With respect to the divisibility:

- A resource is called discrete if it can be allocated to the operations in discrete amounts, i.e. we have a finite set of resources.
- A resource is denoted continuous if it can be assigned to the operations in a continuous amount between an interval.

Controllable processing times have been widely analyzed in the scheduling literature (see reviews in [133] and [182]). Additionally, there are contributions in related decision problems such as resource allocation or software development size team. Processing times depending on the amount of resources have been used in many operation research problems e.g. for single-machine/flowshop scheduling, resource allocation problems, multi-mode resource-constrained project scheduling problem (MRCPSP), etc., and both for discrete and continuous resources. However, to the best of our knowledge, there is no analysis or detailed discussion regarding the different types of relations between processing times and the amount of resources used in controllable-processing-times-based scheduling problems.

Notation

Firstly, it is necessary to clarify the notation to be used here. We have tried to accommodate the terms while maintaining the original notation as far as possible.

Let us assume a job composed of several operations that have to be performed using an amount of resources u. Depending on u, p the processing time of the operation may change, i.e. $p = p(u)$. Additionally, let us define the effort, $e = e(u)$ as the amount of resource-hours or resource-months (depending on the unit of p) that an operation needs to be carried out.

Let us now define S as the size of the operation or workload, which indicates the amount of work that has to be performed to complete the operation. S is measured in the unit of the work. It is an attribute of each operation that will be assumed constant in this section. Thus, the productivity of the operation, Pr, can be written as a function of the size of the operation and the effort (see e.g. [68], [91] and [113]):

$$Pr = \frac{S}{e(u)} = Pr(u) \tag{5.12}$$

An increase in the productivity indicates an increase in the operation size if the effort is constant, or a decrease in the effort when the size of the operation remains the same. As it can be seen in (5.12), the productivity only depends on the amount of resources, u. The amount of resources for which the maximum productivity, $\max(Pr)$, is achieved is denoted as u^* and can be obtained by maximizing the productivity or minimizing the effort. More specifically, we intend to find u^* for which $Pr(u^*) \geq Pr(u)$, $\forall u \neq u^*$. Since S is assumed constant, it is clear that $e(u^*) \leq e(u)$, $\forall u \neq u^*$.

The tuple $(u^*, p(u^*))$, with maximum productivity is denoted as the *productive configuration* or *productive points* of the problem.

Properties

Next, we present two basic properties which must hold. By doing so, two regions are distinguished to establish the limits of the area where each configuration (u, p) can take place. Second, a general law for productive processes is introduced in order to analyze the relations.

Property 5.4.1. *Assuming that the same amount of resources is available for each period, the processing time of the operation must fulfill: $p \geq constant/u$, i.e. the processing time must be over an ideal boundary which corresponds to an inverse proportional relation between p and u as defined in Figure 5.10.*

Proof. If the same amount of resources is employed throughout the duration of the operation, the effort can be written as the amount of resources times the processing time of the operation (5.13):

$$e = p(u) \cdot u \tag{5.13}$$

Substituting in the expression (5.12):

$$Pr(u) = \frac{S}{p(u) \cdot u} \tag{5.14}$$

As shown in (5.13), maximizing the productivity is equivalent to minimize the effort, therefore:

$$\frac{\partial p(u \cdot u)}{\partial u} = 0 = \frac{\partial p(u)}{\partial u} + p(u) \longrightarrow \frac{\partial p(u)}{\partial u} = -\frac{p(u)}{u} \tag{5.15}$$

Thus, solving the differential equation, the point(u, p) reaching the maximal productivity is:

$$p(u) = \frac{k}{u} \tag{5.16}$$

□

Corollary 5.4.1. *Each tuple (u, p) under the ideal boundary, $p(u) \leq k/u$, is infeasible (see Figure 5.11) otherwise, $p(u) \geq k/u$, is feasible.*

Proof. Trivial in view of Property 5.4.1. □

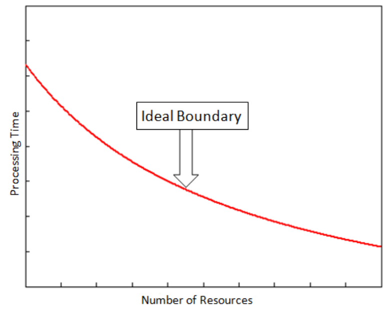

Figure 5.10: Ideal Boundary.

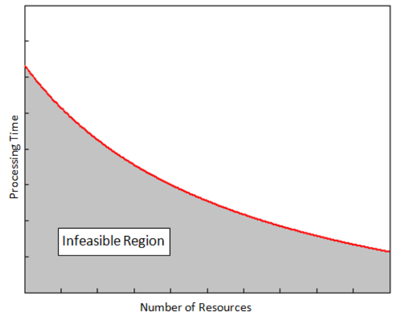

Figure 5.11: Infeasible Region.

For a tuple (u_1, p_1) in the ideal boundary, any other tuple (u, p) placed in the region "Dominated Region" (in Figure 5.12) is dominated by (u_1, p_1), since the latter achieves less processing times with less amount of resources. Note that points that are non-dominated by others are labelled as "efficient" according to the discussion shown in e.g. [29] and [66]. Hence, any tuple (u, p) candidate to be chosen as configuration of the problem must be located outside both the infeasible region and the dominated region, as we can see in Figure 5.13. In general, at least one tuple with minimal $u \cdot p$ must exist representing the most productive configuration to perform the operation, i.e. the productive configuration $(u^*, p(u^*))$. Since the goal pursued by companies is to minimize both the processing times and the amount of resources, different trade-offs can be established, which leads to a number of non-dominated solutions forming a Pareto frontier. Note that the points over this frontier are dominated and cannot be considered as possible configurations for the operation, i.e. non-dominated processing times must be a non-increasing function of the amount of resources assigned to the operation, $dp(u)/du \leq 0$ for each $u \in [\underline{u}, \bar{u}]$.

Property 5.4.2. *Given some amount of resources $u^{'}$, the fulfillment of the law of diminishing marginal returns is the same as the fulfillment of $\frac{d(1/p(u))}{du}(u_2) < \frac{d(1/p(u))}{du}(u_1)$, $\forall u_2 > u_1 > u^{'}$.*

Proof. The law of diminishing marginal returns establishes that, given some amount of resources denoted as $u^{'}$, the output of a productive process increases at a decreasing rate when the amount of recourse

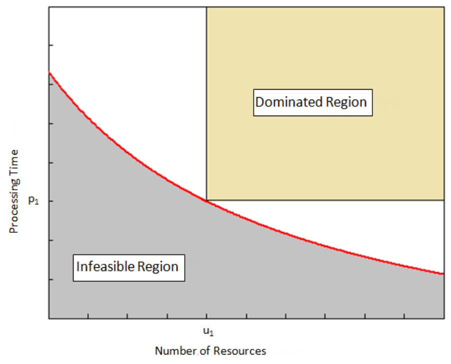

Figure 5.12: Dominated Region.

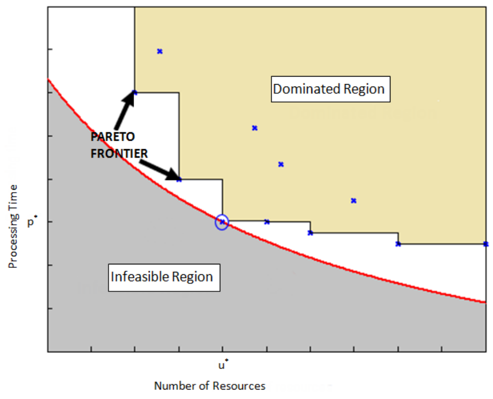

Figure 5.13: Example for many configurations p-u.

increases (see e.g. [7] and [145]). Considering the output $Y = S/(p(u))$ as the amount of the operation size performed in each time period and the input $X = u$, the law of diminishing marginal returns can be written as:

$$Y(X_2) > Y(X_1) \text{ and } \frac{dY}{dX}(X_2) < \frac{dY}{dX}(X_1), \forall X_2 > X_1 > X^{'} \tag{5.17}$$

Substituting Y and X in the expression (5.17):

$$\frac{d\left(\frac{S}{p(u)}\right)}{du}(u_2) < \frac{d\left(\frac{S}{p(u)}\right)}{du}(u_1) \longrightarrow \frac{d\left(\frac{1}{p(u)}\right)}{du}(u_2) < \frac{d\left(\frac{1}{p(u)}\right)}{du}(u_1), \forall u_2 > u_1 > u^{'} \tag{5.18}$$

□

Corollary 5.4.2. *Given some amount of resources $u^{'}$, the processing times of the operation must satisfy $\frac{d^2\left(\frac{1}{p(u)}\right)}{du^2} < 0$, $\forall u^{'} < u < \bar{u}$.*

Proof. Trivial in view of Property 5.4.2. □

This property, together with the previous one, is used in this section to analyze the different relations used in production management. Although in production management the output must satisfy the law of diminishing marginal returns, there is no such condition for renewable discrete resource (manpower). However, there are several results based on the experimentation for manpower in the literature. Among them, [131] established that the u-productivity graphics must be similar to an inverted U-shaped where it is assumed that there is only a single productive configuration with maximum productivity defined by the tuple $(u^*, p(u^*))$ ([6]). The productivity decreases for $u > u^*$, this scenario is denoted as EC in the following,due to the fact that there is too much coordination and communication if more employees are assigned ([143] and [185]), and that these difficulties in communication increase with the size of the team ([167] and [51]). It is also assumed that there is a decrease in productivity due to lack of specialization, denoted as LS, in small teams if fewer employees are assigned (i.e. for $u < u^*$). This fact is confirmed by [35], who also cite other difficulties such as making trade-off decisions, or managing error backlog.

The main relations between processing times and amount of resources are presented and classified in Table 5.4 where b, d, k, ec, f, g, h and i are constants.

We summarize the main characteristic of each relationship in Table 5.5. Columns 2 and 3 indicate the environment where each relation has been used, while the type of resource used is shown in column 4. The next two columns indicate the amount of productive points in the expression and their position, respectively. 7th and 8th columns are related to the fulfillment of the diminishing marginal returns law

Table 5.4: Main relations between processing times and amount of resources

Relationship	Expression
Linear relation	$p(u) = \bar{p} - b \cdot u$ with $\underline{u} \leq u \leq \bar{u}$
Convex relation	$p = \left(\frac{d}{u}\right)^k$
Convex relation + Constant	$p = b + \left(\frac{d}{u}\right)^k$ with $\underline{u} \leq u \leq \bar{u}$
Convex + Communication	$p = \frac{d}{u} + ec \cdot u \cdot (u-1)$
Hyperbola	$\left(\frac{e-h}{f}\right)^2 - \left(\frac{u-i}{g}\right)^2 = 1$
Multimode	—
Piecewise Linear	—

and the inverted U-shaped, respectively. In the last column, the number of constants necessary to fulfill each $p-u$ relation has been represented. Note that a higher value of the number of constants means more difficulty to configure the model.

Table 5.5: Relations between processing times and amount of resources used in the literature.

p(u)	Type of Resource	# Productive Points	Productive Points	Diminishing Marginal Returns Law	Inverted U-shaped	# Constants
Linear	Generic	1 or 2	Endpoints	Unfulfilled	≈ EC or LS	2
Convex $k = 1$	Generic	All	Every point	Unfulfilled	Unfulfilled	1
Convex $k < 1$	Generic	1	Left endpoint	Partially ($u^{'}$=Left endpoint)	≈ EC	2
Convex $k = 0.5+$ constant	Non-Renewable and Continuous	1	Left endpoint	Partially ($u^{'}$=Left endpoint)	≈ EC	3
Convex $k = 1+$ constant	Non-Renewable and Continuous	1	Left endpoint	Partially ($u^{'}$=Left endpoint)	≈ EC	3
Convex $k = 2+$ constant	Non-Renewable and Continuous	1	$d/\sqrt{b}$	Fulfilled ($u^{'} = \frac{d}{\sqrt{2 \cdot b}}$)	≈ Fulfilled	3
Convex $k = 1+$ Polynomial	Renewable and Discrete	1	Left endpoint	—	≈ EC	2
Multimode	Discrete	n	Chosen by the decision maker	Fulfilled	Fulfilled	n
Piecewise Linear	Non-Renewable and Continuous	n	Point in the convex curve	Fulfilled	Fulfilled	n
Hyperbola	Renewable and Discrete	1	Chosen by the decision maker	—	Fulfilled	4

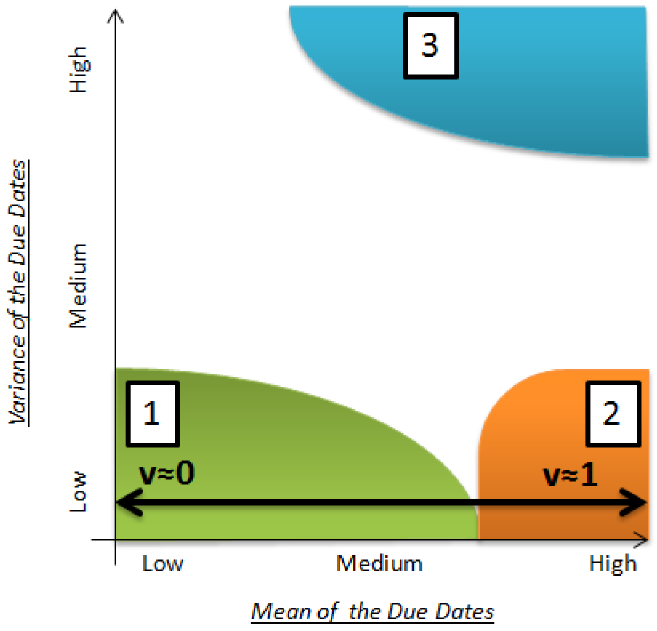

Figure 5.14: Location of the problem based on the mean and variance of the due dates.

5.5 Conclusions

Due dates

According to the previous analysis in Section 5.2, the $Fm|prmu|\sum T_j$ on an instance is bounded by three different problems depending on v and on the variance of the due dates of the jobs, as shown in Figure 5.14. Roughly speaking, high values of the mean and variance of the due dates correspond to a problem where the EDD rule is optimal (see Region 3 of Figure 5.14). Low values of the mean and variance determine a problem similar to $Fm|prmu|\sum C_j$ (see Region 1 of Figure 5.14). Finally, high values of the mean of the due dates combined with a low variance correspond to a trivial problem where each sequence is optimal (Region 2). The interesting region to be analysed for the $Fm|prmu|\sum T_j$ problem is the region between 1, 2 and 3, since otherwise we would be solving a different decision problem.

Regarding the $Fm|prmu|\sum E_j + T_j$, it is bounded by the $Fm|prmu|\sum C_j$ in case of tight due dates, and the $Fm|prmu| - \sum C_j$ problem in case of loose due dates. Typically, the good performance of a constructive heuristic is due to the fact that the objective computed in the iterations of the algorithm is similar to the objective function of the problem. Thereby, when minimising e.g. total flowtime in the PFSP, the choice of a partial sequence fulfilling the minimisation of total flowtime clearly seems to have a good performance when the sequence is completed. This is a consequence of having a regular measure as objective. Since this is not the case for the $Fm|prmu|\sum E_j + T_j$ due to its relationships with $Fm|prmu|\sum C_j$ and $Fm|prmu|-\sum C_j$ for extreme values of the due dates, the algorithm may not work well. In fact, Properties 5.2.2 and 5.2.2 confirm this fact and show how the objective of the constructive heuristics in their iterations could be distorted: A partial sequence could have loose due dates but, once completed, these due dates would become tight and thus, the algorithm would solve a completely different objective during its iterations than the objective function. This fact could also explain the good performance of composite heuristics as compared to constructive heuristics (as discussed in Chapter 9). In order to overcome the aforementioned problems, efficient heuristics for $Fm|prmu|\sum E_j + T_j$ should be designed according to the following ideas:

- They should be very fast in order to work as soon as possible with complete sequences. In this manner, it is easier to identify whether the instance has loose due dates, or tight ones.
- They should incorporate an analysis of both sequenced and non-sequenced jobs in each iteration.
- They should avoid the use of local search procedures operating with non-complete sequences.

Processing times

Regarding conclusions about the processing times of the jobs in the PFSP, on the one hand, theoretical results prove that both problems (the PFSP and the SMSP) are equivalent under several conditions. Although those conditions are hardly present in a real manufacturing environment (mostly in shops with several machines), they are sufficient but not necessary conditions and they only give an idea of the relationship between both problems. On the other hand, the empirical comparison stresses the high relationship between both problems. It has been showed that the increase of γ (related to the dispersion of the processing times) and n and the decrease of β (related to the predominance of the most loaded machine) and m makes the PFSP to be more similar to a SMSP. In fact, for low values of β and/or m, procedures for the equivalent SMSP are able to find similar or even better solutions than the heuristics to solve the original PFSP. In order to empirically establish the frontier between both problem, an extensive set of instances with 608,000 instances has been generated. Then, we have determine several boundary lines depending on the number of machines in the shop. Given a configuration in the shop (number of machines, initial machine availabilities, objective to be solved, length or variation of the processing times on the machines), these lines show the value of γ causing the difference of ARPD between the heuristics of both problems to be less than 0.5% on the analysed set of instances.

The relation between both scheduling problems remarks the importance of the pre-processing of the processing times of the problems as well as the importance of the right choice of the scheduling problem to be solved which do not have to necessary have the original machine environment of the shop. Additionally, it explains the behaviour found in the paper of [206] and [144] where the $Fm|prmu|C_{\max}$ has been found to be easily solvable for structured instances and for machines with initial availabilities respectively.

Controllable processing times

The following conclusions can be obtained for the controllable processing times regarding the productive point, the law of diminishing marginal returns and the renewable discrete resources. Beginning with the productive configuration, it is found that:

- The productive configuration obtained by the relations convex relation with $k = 0.5$, convex relation with $k = 0.5+$ constant, convex relation with $k = 1+$ constant or the convex plus communication is a unique productive point which is necessarily the left endpoint. Hence, feasible points are only on the right of that point. Thereby, regarding manpower, only the excess of communication can be modelled.

- The productive configuration is placed either in one endpoint or eventually in both endpoints by the linear relation.

- Convex relation with $k = 1$ corresponds with the ideal boundary.

- Control over the productive point is only allowed for non-renewable resources by the convex relation with $k = 2+$ constant and for renewable resources by hyperbola and multi-mode.

Regarding the law of diminishing marginal returns, the following aspects can be summarized:

- The relations that partially satisfy the law of diminishing marginal returns (only for the decreasing part of the law) are the convex relation with $k < 1$, the convex relation with $k = 0.5+$ constant, the convex relation with $k = 1+$ constant since the $u^{'}$ must be placed in the left endpoint.

- The convex relation with $k = 2+$ constant fulfills the diminishing marginal returns law for any amount of resources bigger than $d/\sqrt{3 \cdot b}$, i.e. $u^{'} = d/\sqrt{3 \cdot b}$.

- The linear relation, the convex relation with $k = 1$ and the piecewise linear relation does not fulfill this law.

With respect to renewable discrete resource:

- The convex relation with $k = 1$ does not reflect the reality for manpower.

- The inverted U-shaped is only fulfilled by the hyperbola and the convex relation with $k = 2$ plus a constant and eventually, by the multimode and the piecewise linear relation.

- The lack of specialization can be also eventually approximated by the linear relation.

Part III

SOLUTION PROCEDURES

Chapter 6

PFSP to minimise makespan

6.1 Introdution

In this section, dealing with specific objectives SO2, SO3 and SO4, we propose a new tie-breaking mechanism that outperforms existing ones both in the NEH and in the Iterated Greedy, as well as an extensive computational evaluation of algorithms. The rationale of our proposed tie-breaking mechanism is relatively simple, as it seems intuitive that lower values of the total idle time would mean less delays in the processing of the jobs, which would eventually lead to a better utilization of the machines and to a shortest makespan value once all jobs have been positioned. The challenge is to calculate these idle times in an efficient manner, particularly taking into account that Taillard's acceleration provides a very fast mechanism to evaluate the subsequences which is at the core of the excellent performance of NEH and Iterated Greedy. Our proposal is to use an ersatz of the idle times that can be calculated in parallel to the evaluation of the makespan of the subsequences and thus not adding computational complexity to the algorithms.

This chapter is organized as follows: Section 6.2 is devoted to explain the proposed tie-breaking mechanism. In Section 6.3, our proposal is compared against existing tie-breaking mechanisms when embedded in the NEH, and in the IG_RS_{LS} and IG_{RIS} algorithms. In Section 6.4 an exhaustive computational evaluation is performed comparing the proposed mechanisms with the most promising heuristics and metaheuristics in the literature. Finally, in Section 6.5, the main conclusions are discussed.

6.2 The proposed tie-breaking mechanism

The tie-breaking mechanism presented in this chapter (denoted by TB_{FF} in the following) is related to the minimisation of total idle times. According to [42], the definition of machine idletime is not unambiguous and at least three different ways have been used:

- The idletime considering front delays (time before first job) and back delays (time after the last job on the machine).
- Excluding front and back delays.
- Including front delays and excluding back delays.

In this chapter, we assume the third definition of the idle time. Therefore, it_i the idle time of machine i can be calculated according to the expression $it_i = C_{in} - \sum_{j=1}^{n} p_{ij}$ and the total idle time by $it = \sum_{i=1}^{m} it_i$. If we denote by Δ_{ij} the idle time in machine i induced between the completion of job in position j and the beginning of job in position $j+1$, then Δ_{ij} can be written in terms of equations (2.2) as follows:

$$\Delta_{ij} = (e_{i,j+1} - p_{i,j+1}) - e_{ij} \tag{6.1}$$

The first two terms in the right side of the equation indicate the starting time of job in position $j+1$, therefore subtracting the completion time of job in position j yields the idle time between jobs in positions j and $j+1$ in machine i. Clearly, $it_i = \sum_{j=0}^{n-1} \Delta_{ij}$, and therefore $it = \sum_{i=1}^{m} \sum_{j=0}^{n-1} \Delta_{ij}$

In order to explain the tie-breaking mechanism, let us assume that we have a subsequence of $k-1$ jobs (see Figure 6.1). Then, an unscheduled job r is going to be inserted in all positions in the subsequence in order to select the position yielding the minimum makespan. If ties occur, then the position whose insertion yields the minimum total idle time is to be selected. Note that, if the unscheduled job is to be inserted in position l, then $g_{i,l-1}$ the cumulative idle times on machine i induced by jobs prior to position $l-1$ is:

$$g_{i,l-1} = \sum_{j=0}^{l-2} \Delta_{ij} = \sum_{j=1}^{l-1} [(e_{ij} - p_{ij}) - e_{i,j-1}] \tag{6.2}$$

Analogously, h_{il} the cumulative idle times on machine i induced by jobs after position l is:

$$h_{il} = \sum_{j=l}^{k-2} \Delta_{ij} = \sum_{j=l}^{k-2} [(e_{i,j+1} - p_{i,j+1}) - e_{ij}] \tag{6.3}$$

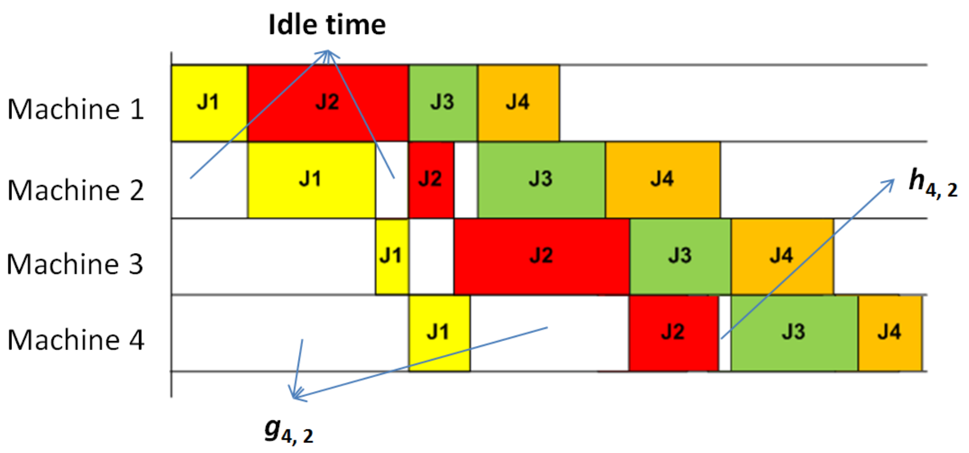

Figure 6.1: Sequence of jobs before inserting the new job in position l

It is clear that, for each machine i, when an unscheduled job r (with t_{ir} its processing time on machine i) is inserted in position l (see example in Figure 6.2), $g_{i,l-1}$ remains the same. However, this does not happen for h_{il}, which would have to be recomputed. Unfortunately, doing so would substantially increase the computation time since Taillard's acceleration cannot be employed to calculate the new idle times. As a consequence, we suggest using an estimation of the idle time as tie-breaking indicator, based on the assumption that the new Δ_{ij} values for jobs in positions $l+1$ to k are not very different from the old ones. Therefore, when inserting an unscheduled job, e_{ij}, q_{ij} and f_{il} are used according to equations (2.2-2.4) in order to obtain the makespans for each position. Then, the position yielding the minimum makespan is selected. In case of ties, we calculate an estimation of the new idle time denoted by $it^{'}(l)$ for each position l for which the tie occurs, and selects the position l yielding the minimum makespan for which $it^{'}(l)$ is minimum:

$$it'(l) = \sum_{i=1}^{m} \left(g_{i,l-1} + h_{il} + \Delta^{'}_{i,l-1} + \Delta^{'}_{il} \right) \tag{6.4}$$

The first term in Equation (6.4) denotes the idle time in machine i caused by the jobs prior to position $l-1$. This value has been already obtained, as it has not been modified by the insertion of the job. The second term is the idle time in machine i caused by jobs in (old) positions l to $k-1$ (now positions $l+1$ to k once the job is inserted). As stated before, after the insertion of job l, this is not anymore the idle time of the new sequence, but we will assume that they are the same (hence the estimation). Finally, the insertion of the job in position l induces a new idle time between the job in position $l-1$ and the new job (denoted by $\Delta^{'}_{i,l-1}$), and between the new job and the job in the old position l ($l+1$ after the insertion), denoted by $\Delta^{'}_{i,l}$. Both terms can be easily calculated from the data obtained from Taillard's acceleration:

$$\Delta^{'}_{i,l-1} = (f_{il} - t_{ir}) - e_{i,l-1} \tag{6.5}$$

and

$$\Delta^{'}_{i,l} = \max\{f^{'}_{i-1,l} - f_{il}, 0\} \tag{6.6}$$

where $f^{'}_{i-1,l}$ is the completion time on machine i of the job which before was in position l (after inserting the new job, it corresponds to the job placed in position $l+1$) and can be computed as follows:

$$f^{'}_{il} = \max\{f_{il}, f^{'}_{i-1,l}\} + p_{il}, i = 1 \ldots m \tag{6.7}$$

and $f^{'}_{0l} = 0$ being $p_{i,l}$ the processing time of the job that before was in position l.

Note that Equation (6.4) can be simplified by means of the idle time, $\Delta_{i,l-1}$, between the job in position $l-1$ and l:

$$it'(l) = \sum_{i=1}^{m} \left(g_{i,l-1} + h_{il} + \Delta_{i,l-1} - \Delta_{i,l-1} + \Delta^{'}_{i,l-1} + \Delta^{'}_{il} \right) \tag{6.8}$$

$$it'(l) = \sum_{i=1}^{m} (g_{i,l-1} + h_{il} + \Delta_{i,l-1}) + \sum_{i=1}^{m} \left(\Delta^{'}_{i,l-1} + \Delta^{'}_{il} - \Delta_{i,l-1} \right) \tag{6.9}$$

Equation (6.9) can be decomposed into two terms, i.e.:

$$it'(l) = C + it^{''}(l) \tag{6.10}$$

where $C = \sum_{i=1}^{m} (g_{i,l-1} + h_{il} + \Delta_{i,l-1})$ is a constant that does not depend on the tie-breaking l, and $it^{''}(l)$ is:

$$it^{''}(l) = \sum_{i=1}^{m} \left(\Delta^{'}_{i,l-1} + \Delta^{'}_{il} - \Delta_{i,l-1} \right) = \sum_{i=1}^{m} \left(f_{il} - e_{il} + p_{il} - t_{ir} + \max\{f^{'}_{i-1,l} - f_{il}, 0\} \right) \tag{6.11}$$

where it has been used that $\Delta_{i,l-1} = (e_{il} - p_{il}) - e_{i,l-1}$, see Equation (6.1). Additionally, since $\sum_{i=1}^{m} t_{ir}$ is the same regardless the position l where the job is inserted, we can define $it^{'''}(l)$ a more concise indicator equivalent to $it^{''}(l)$ as follows:

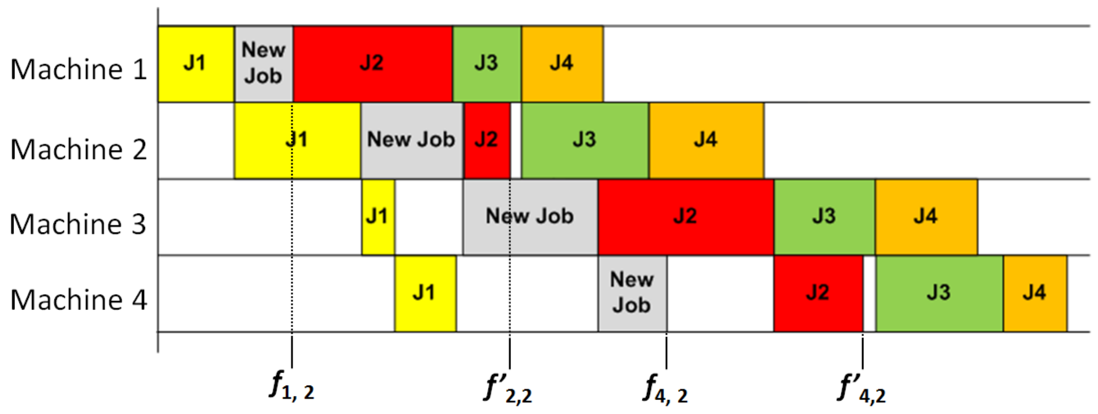

Figure 6.2: Sequence of jobs after inserting the job in position l

$$it^{'''}(l) = \sum_{i=1}^{m} (f_{il} - e_{il} + p_{il} + \max\{f^{'}_{i-1,l} - f_{il}, 0\}) \tag{6.12}$$

Therefore, the proposal breaks the ties according to the minimisation of $it^{''}(l)$ –or, equivalently, to the minimisation of $it^{'''}(l)$. The pseudo code of this tie-breaking mechanism for the NEH is shown in Figure 6.3. Note that the idle time $it^{''}(l)$ is forced to be zero for the last job to be inserted, i.e. no tie-breaking mechanism is considered for the last job to be inserted. It can be easily checked that the insertion of tie-breaking mechanism does not alter the complexity of the algorithm, i.e. it remains $O(n^2 \cdot m)$. Analogously, this mechanism can be easily incorporated in the constructive and in the local search phase of IG_RS_{LS} [174], and in the IG_{RIS} by [138].

6.3 Computational comparison of tie-breaking mechinisms

The tie-breaking mechanisms described in the previous section have been coded in C# and embedded into the NEH and the two versions of the Iterated Greedy. As for initial ordering in the NEH, the non-increasing order of the sum of the processing times has been adopted. This is the initial order of the original NEH and it has been chosen because, on one hand, it is the most widely-employed mechanism and the results are easier to compare with the rest of the literature. On the other hand, this allows focusing exclusively on insertion tie-breaking mechanisms and removes the possible influence of more elaborated initial ordering rules such as the ones discussed above. Nevertheless, we also provide the results using these more advanced initial orderings.

The computational experiments are carried out on an Intel Core i7-930, 2.8GHz, 16GB RAM under Windows 7. This section is divided into two parts depending on which heuristic (NEH or Iterated Greedy

```
$\pi^{'}$ ⟵ Sort in decreasing order of sum of processing times $p_{ij}$;
$\pi$ ⟵ $\pi^{'}_1$;
for $k = 2$ to $n$ do
    $r$ ⟵ $\pi_k{}'$;
    Determine the values of $e_{ij}$, $q_{ij}$ and $f_{il}$ from Taillard's acceleration (see equations 2.2, 2.3, and 2.4);
    Determine minimal makespan resulted of testing the job $r$ in all possible positions of $\pi$;
    $bp$ ⟵First position where the makespan is minimal;
    $tb$ ⟵ Number of positions with minimal makespan (i.e. number of ties);
    $ptb$ ⟵ Array (of length $tb$) with the positions where the makespan is minimal;
    $it_{bp}$ is the idletime corresponding to the $bp$ and set to a very large number;
    if $tb > 1$ and $k < n$ then
        for $l = 1$ to $tb$ do
            $it^{''} \longleftarrow 0$;
            if $ptb[l] = k$ then
                for $i = 2$ to $m$ do
                    $it^{''} \longleftarrow it^{''} + f_{i,k} - e_{i,k-1} - t_{i,r}$;
                end
            else
                $f^{'}_{1,ptb[l]} \longleftarrow f_{1,ptb[l]} + p_{1,ptb[l]}$;
                for $i = 2$ to $m$ do
                    $it'' \longleftarrow it^{''} + f_{i,ptb[l]} - e_{i,ptb[l]} + p_{i,ptb[l]} - t_{i,r} + \max\{0, f^{'}_{i-1,ptb[l]} - f_{i,ptb[l]}\}$;
                    $f^{'}_{i,ptb[l]} \longleftarrow \max\{f^{'}_{i-1,ptb[l]}, f_{i,ptb[l]}\} + p_{i,ptb[l]}$;
                end
            end
            if $it_{bp} > it^{''}$ then
                $bp \longleftarrow ptb[l]$;
                $it_{bp} \longleftarrow it^{''}$;
            end
        end
    end
    $\pi$ ⟵ Array obtained by inserting job $r$ in position $bp$ of $\pi$;
end
```

Figure 6.3: Our Tie-Breaking Method

Table 6.1: Average relative percentage deviation of NEH implemented with tie-breaking mechanisms

Instance	TB_{FS}	TB_D	TB_{KK1}	TB_{KK2}	TB_{FF}
20 x 5	3.300	2.655	2.638	2.729	2.293
20 x 10	4.601	4.661	4.488	4.312	4.152
20 x 20	3.731	3.443	3.683	3.407	3.305
50 x 5	0.727	0.497	0.586	0.588	0.922
50 x 10	5.073	5.082	5.022	4.875	5.150
50 x 20	6.648	6.091	6.274	6.412	6.207
100 x 5	0.527	0.459	0.354	0.397	0.378
100 x 10	2.215	2.065	1.829	1.771	2.182
100 x 20	5.345	5.235	5.417	5.284	5.021
200 x 10	1.258	1.182	1.179	1.166	0.984
200 x 20	4.408	3.901	4.243	4.232	4.037
500 x 20	2.066	1.779	2.080	2.020	1.776
Average	3.325	3.088	3.149	3.099	3.034

Algorithm) the tie breaks are implemented.

Comparison of tie-breaking mechanisms for the NEH

The performance of the NEH with the tie-breaking mechanisms by [35] (denoted as TB_D), [83] (labelled TB_{KK1}), [84] (denoted as TB_{KK2}) and our proposal, as well as with the original tie-breaking mechanism of the NEH (labelled TB_{FS} in the following) are compared using the benchmark $\mathcal{B}_1$. Note that, although in [86] it was established that Dong's tie-breaking mechanism outperformed the two suggested by Kalczynski&Kamburowski for the NEH, we nevertheless include them to test them against the proposal and to make the comparison homogeneous with that of the IG (for which none of the mechanisms' performance was tested).

For each instance, the $RPD2$ is computed with respect to the best known solution according to Expression (3.2), where O^{ih} is the solution obtained for instance i by the NEH algorithm using the j tie-breaking mechanism while UB is the best known solution or the lowest known upper bound value for the instance. The Average $RPD2$ ($ARPD2$) values are obtained by averaging $RPD2$ for each instance size or for the whole testbed. The results in Table 6.1 show that the $ARPD2$ found by the original NEH is 3.325 while each other tie break yields better $ARPD2$, being the value of 3.034 the best one, obtained by our tie-breaking proposal.

Since we use the same test bed for all tie-breaking mechanisms, being each one a version of the same algorithm, the random variables ($ARPD2$) are related and the hypothesis of independence can be rejected. Therefore, a paired samples t-test (shown in Table 6.2) can be used to compare the results. Note that paired samples t-test is a usual test to establish the statistical significance of the differences in the perfor-

Table 6.2: Paired samples t-test for NEH using Taillard's benchmark

Algorithm	Mean	SEM	IC - Lower	IC - Upper	t	Significance
TB_{FS} - TB_{FF}	0.291	0.806	0.146	0.437	3.958	0.000
TB_{FS} - TB_{D}	0.237	0.852	0.084	0.391	3.054	0.003
TB_{FS} - TB_{KK1}	0.176	0.736	0.043	0.308	2.626	0.010
TB_{FS} - TB_{KK2}	0.226	0.711	0.098	0.354	3.490	0.001
TB_{D} - TB_{FF}	0.054	0.770	-0.086	0.193	0.764	0.446
TB_{KK1} - TB_{FF}	0.115	0.842	-0.037	0.268	1.502	0.136
TB_{KK2} - TB_{FF}	0.066	0.877	-0.093	0.224	0.819	0.415

Table 6.3: Paired samples t test for NEH using the extended benchmark

Algorithm	Mean	SEM	IC - Lower	IC - Upper	t	Significance
TB_{FS} - TB_{FF}	0.226	0.496	0.177	0.274	9.117	0.000
TB_{FS} - TB_{D}	0.169	0.486	0.121	0.216	6.943	0.000
TB_{FS} - TB_{KK1}	0.102	0.466	0.056	0.148	4.375	0.000
TB_{FS} - TB_{KK2}	0.126	0.472	0.080	0.173	5.351	0.000
TB_{D} - TB_{FF}	0.057	0.450	0.013	0.101	2.545	0.011
TB_{KK1} - TB_{FF}	0.124	0.490	0.076	0.172	5.057	0.000
TB_{KK2} - TB_{FF}	0.099	0.410	0.059	0.140	4.849	0.000

mance of algorithms for flowshop scheduling problems in Taillard's testbed (see e.g. [60, 191, 34]). In view of the values of the significance levels, it can be stated that each tie-breaking mechanism is statistically significant with respect to TB_{FS}. However, no statistical significance among the rest of the tie-breaking mechanisms can be found due to the small size of the benchmark, a fact also noted by [84] when proposing their tie-breaking mechanisms. Therefore, in line with these authors, an extended test-bed of 400 instances with $n \in \{50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$, and $m \in \{5, 10, 15, 20\}$, with 10 replications for each combinations of n and m is generated with the processing times uniformly distributed in the interval $[1, 99]$. A paired-samples t-test (shown in Table 6.3) was performed, indicating that our proposal is statistically significant with respect to the other tie-breaking mechanisms, being 0.01 the maximum p-value found.

Results of the NEH algorithm with the proposed tie-breaking mechanism using different initial orders are shown in Table 6.4 for the Taillard's testbed. As explained above, three different initial orders outperforming the original non-ascending order of the sum of their processing times have been proposed in the literature by [84] (denoted as $KK1 - Init$), by [35] (denoted as $AvgDev - Init$); and by [85] (denoted as $KK2 - Init$). All three were implemented in order to obtain the best initial order for the NEH using our tie-breaking mechanism. The $ARPD2$ using the initial order $AvgDev$ was 2.897 being the best initial order for Taillard's testbed, a result in line with those obtained by [86].

Table 6.4: Average relative percentage deviation of NEH implemented with our tie-breaking mechanisms and different initial order

Instance	$KK1 - Init$	$KK2 - Init$	$AvgDev - Init$	$Original$
20 x 5	2.484	2.372	2.559	2.293
20 x 10	4.919	4.453	3.543	4.152
20 x 20	3.265	3.509	3.331	3.305
50 x 5	0.555	0.791	0.749	0.922
50 x 10	4.865	4.861	4.905	5.150
50 x 20	6.139	7.026	5.812	6.207
100 x 5	0.379	0.321	0.412	0.378
100 x 10	1.961	2.057	1.719	2.182
100 x 20	5.284	5.114	5.147	5.021
200 x 10	1.030	0.899	0.987	0.984
200 x 20	3.712	3.895	3.885	4.037
500 x 20	1.726	1.650	1.713	1.776
Average	3.027	3.079	2.897	3.034

Comparison of the tie-breaking mechanisms in the iterated greedy

As explained before, the iterated greedy has two parameters (T, d) to be set. [174] conducted a full factorial design to determine both parameters, resulting $d = 4$ and $T = 0.4$ as the best combination. Therefore, these values are used in our implementation. Two versions of the iterated greedy are analysed: IG_RS_{LS} as in [174] and IG_{RIS} as proposed by [138]. The tie-breaking mechanism analysed in Section 6.2 was integrated in these Iterated Greedy Algorithms, together with our proposal. In order to compare them, the same test bed as in [174] was employed, i.e. Taillard's benchmark using 5 replicates for each instance to increase the power of the analysis.

The termination criterion considered for both versions of the Iterated Greedy is the CPU time. In line with most papers, this time t depends on the amount of jobs and machines, i.e. $t = n \cdot (m/2) \cdot 30$, $t = n \cdot (m/2) \cdot 60$ and $t = n \cdot (m/2) \cdot 90$ milliseconds (see e.g. [174], or [195]). $ARPD2$ results for each version of the iterated greedy algorithm and for each tie-breaking mechanism are shown in Tables 6.5, 6.6 and 6.7 for each stopping time, respectively. The results show that the $ARPD2$ for IG_RS_{LS} with our tie-breaking mechanism is the best for every stopping time, being the average results 0.461, 0.376 and 0.350 respectively. Kalczynski & Kamburowski's tie-breaking mechanism II also yields good $ARPD2$ results: 0.518, 0.446 and 0.418 respectively. Both mechanisms performs better than the original iterated greedy algorithm. Nevertheless, it is to note that Dong's tie-breaking mechanism and Kalczynski & Kamburowski's tie-breaking mechanism I give worse results when included in IG_RS_{LS}.

Furthermore, a paired-samples t- test was carried out in order to analyse the different mechanisms (see Table 6.8). Our tie-breaking mechanism was found to be statistically significant with respect to every

Table 6.5: Average relative percentage deviation of iterated greedy algorithms implemented with tie breaks and $n \cdot (m/2) \cdot 30$ milliseconds as stopping criterion

	IG_RS_{LS}					IG_{RIS}				
Instance	TB_{FS}	TB_D	TB_{KK1}	TB_{KK2}	TB_{FF}	TB_{FS}	TB_D	TB_{KK1}	TB_{KK2}	TB_{FF}
20 x 5	0.045	0.066	0.076	0.049	0.076	0.037	0.041	0.076	0.053	0.041
20 x 10	0.055	0.052	0.087	0.080	0.104	0.080	0.096	0.099	0.064	0.057
20 x 20	0.092	0.095	0.085	0.066	0.114	0.081	0.093	0.098	0.090	0.092
50 x 5	0.007	0.039	0.021	0.003	0.017	0.006	0.020	0.024	0.007	0.006
50 x 10	0.724	0.754	0.842	0.707	0.566	0.683	0.651	0.787	0.666	0.621
50 x 20	1.199	1.188	1.228	1.191	1.134	1.160	1.066	1.195	1.149	1.173
100 x 5	0.005	0.066	0.030	0.013	0.014	0.005	0.067	0.018	0.013	0.013
100 x 10	0.274	0.383	0.415	0.215	0.226	0.258	0.301	0.336	0.202	0.219
100 x 20	1.624	1.446	1.789	1.624	1.346	1.547	1.365	1.770	1.542	1.387
200 x 10	0.317	0.477	0.284	0.140	0.155	0.267	0.361	0.263	0.161	0.148
200 x 20	1.656	1.401	1.925	1.466	1.239	1.549	1.287	1.898	1.478	1.248
500 x 20	0.767	0.724	1.033	0.668	0.542	0.728	0.621	0.987	0.626	0.530
Average	0.564	0.558	0.651	0.518	0.461	0.534	0.497	0.629	0.504	0.461

Table 6.6: Average relative percentage deviation of iterated greedy algorithms implemented with tie breaks and $n \cdot (m/2) \cdot 60$ milliseconds as stopping criterion

	IG_RS_{LS}					IG_{RIS}				
Instance	TB_{FS}	TB_D	TB_{KK1}	TB_{KK2}	TB_{FF}	TB_{FS}	TB_D	TB_{KK1}	TB_{KK2}	TB_{FF}
20 x 5	0.024	0.039	0.041	0.041	0.032	0.032	0.041	0.041	0.032	0.041
20 x 10	0.043	0.031	0.049	0.057	0.059	0.038	0.042	0.064	0.032	0.046
20 x 20	0.067	0.042	0.047	0.049	0.057	0.052	0.070	0.066	0.060	0.071
50 x 5	0.004	0.009	0.016	0.003	0.007	0.000	0.026	0.010	0.004	0.001
50 x 10	0.529	0.615	0.692	0.595	0.441	0.549	0.524	0.626	0.584	0.478
50 x 20	1.044	1.005	1.047	0.978	1.048	1.011	0.940	1.060	1.008	1.012
100 x 5	0.008	0.056	0.011	0.006	0.006	0.006	0.028	0.006	0.006	0.009
100 x 10	0.218	0.228	0.310	0.170	0.149	0.173	0.214	0.223	0.184	0.111
100 x 20	1.423	1.317	1.643	1.449	1.118	1.394	1.145	1.589	1.402	1.245
200 x 10	0.250	0.397	0.217	0.092	0.093	0.174	0.271	0.188	0.113	0.101
200 x 20	1.407	1.217	1.819	1.313	1.049	1.407	1.125	1.754	1.401	1.036
500 x 20	0.720	0.627	0.992	0.602	0.453	0.650	0.519	0.958	0.573	0.473
Average	0.478	0.465	0.573	0.446	0.376	0.457	0.412	0.549	0.450	0.385

Table 6.7: Average relative percentage deviation of iterated greedy algorithms implemented with tie breaks and $n \cdot (m/2) \cdot 90$ milliseconds as stopping criterion

	IG_RS_{LS}					IG_{RIS}				
Instance	TB_{FS}	TB_D	TB_{KK1}	TB_{KK2}	TB_{FF}	TB_{FS}	TB_D	TB_{KK1}	TB_{KK2}	TB_{FF}
20 x 5	0.015	0.024	0.041	0.041	0.041	0.041	0.041	0.039	0.041	0.041
20 x 10	0.035	0.022	0.072	0.026	0.024	0.025	0.040	0.048	0.057	0.024
20 x 20	0.038	0.030	0.049	0.048	0.035	0.028	0.054	0.028	0.041	0.051
50 x 5	0.000	0.007	0.007	0.002	0.004	0.003	0.007	0.011	0.001	0.003
50 x 10	0.517	0.567	0.649	0.532	0.438	0.493	0.526	0.583	0.555	0.453
50 x 20	0.918	0.953	0.978	0.874	0.858	0.902	0.837	0.912	0.925	0.935
100 x 5	0.006	0.053	0.008	0.008	0.001	0.006	0.037	0.008	0.004	0.003
100 x 10	0.213	0.253	0.293	0.183	0.169	0.199	0.187	0.239	0.139	0.155
100 x 20	1.261	1.193	1.485	1.388	1.096	1.274	1.048	1.516	1.350	1.106
200 x 10	0.169	0.388	0.180	0.080	0.078	0.155	0.241	0.171	0.069	0.061
200 x 20	1.337	1.184	1.706	1.276	1.026	1.278	1.049	1.704	1.344	0.987
500 x 20	0.674	0.611	0.933	0.558	0.428	0.605	0.488	0.920	0.543	0.412
Average	0.432	0.441	0.533	0.418	0.350	0.417	0.380	0.515	0.422	0.353

other tie-breaking mechanism for every value of t considered, being 0.017 the highest p-value. Regarding the rest of the tie-breaking mechanisms, Kalczynski & Kamburowski's tie-breaking mechanism II was found to be statistically significant with respect to the original IG_RS_{LS} for $t = n \cdot (m/2) \cdot 30$ and $t = n \cdot (m/2) \cdot 60$ but not for $t = n \cdot (m/2) \cdot 90$, being 0.118 the p-value for this case. On the other hand, Kalczynski & Kamburowski's tie-breaking mechanism I was found statistically worse, with p-value of 0.000. Finally, no statistically significance was found for any t between Dong's tie-breaking mechanism and the original IG_RS_{LS}. Regarding IG_{RIS}, very similar results were found (results are shown in Tables 6.5, 6.6 and 6.7 for the different values of t). On the one hand, the proposed tie-breaking mechanism yields again the best $ARPD2$, being 0.461 for $t = n \cdot (m/2) \cdot 30$ milliseconds, 0.385 for $t = n \cdot (m/2) \cdot 60$ milliseconds, and 0.353 $t = n \cdot (m/2) \cdot 90$ milliseconds. On the other hand, Kalczynski & Kamburowski's tie-breaking mechanism I is again the one with the worst results.

It is worth to highlight that the proposed tie-breaking mechanism performs better than existing mechanisms when embedded in the iterated greedy than when integrated in the NEH. Note that the fact that a tie-breaking mechanism performs efficiently for the NEH does not imply the same for the iterated greedy. This is due to the fact that, in the NEH, the insertion is performed in all steps (i.e. from an one-job sequence until the n jobs have been scheduled), while the construction phase of the iterated greedy is performed only for the last d steps (beginning with a sequence of $N - d$ jobs). Therefore, a tie-breaking mechanism should have a good performance in the last steps of the insertion phase in order to be efficient when embedded in the iterated greedy algorithm.

Table 6.8: Paired samples t test for IG_RS_{LS} using the benchmark of Taillard

CPU time	Algorithm	Mean	SEM	IC - Lower	IC - Upper	t	Significance
$n \cdot (m/2) \cdot 30$	TB_{FS} - TB_{FF}	0.103	0.277	0.080	0.125	9.078	0.000
	TB_{D} - TB_{FF}	0.097	0.283	0.074	0.119	8.361	0.000
	TB_{KK1} - TB_{FF}	0.190	0.335	0.163	0.217	13.889	0.000
	TB_{KK2}- TB_{FF}	0.057	0.235	0.038	0.076	5.961	0.000
	TB_{FS} - TB_{D}	0.006	0.286	-0.017	0.029	0.522	0.602
	TB_{FS} - TB_{KK1}	-0.087	0.259	-0.108	-0.067	-8.266	0.000
	TB_{FS} - TB_{KK2}	0.045	0.222	0.028	0.063	5.002	0.000
$n \cdot (m/2) \cdot 60$	TB_{FS} - TB_{FF}	0.102	0.266	0.081	0.124	9.402	0.000
	TB_{D} - TB_{FF}	0.089	0.272	0.067	0.111	8.034	0.000
	TB_{KK1} - TB_{FF}	0.197	0.361	0.169	0.226	13.397	0.000
	TB_{KK2}- TB_{FF}	0.070	0.250	0.050	0.090	6.880	0.000
	TB_{FS} - TB_{D}	0.013	0.263	-0.008	0.034	1.205	0.229
	TB_{FS} - TB_{KK1}	-0.095	0.267	-0.117	-0.074	-8.746	0.000
	TB_{FS} - TB_{KK2}	0.032	0.251	0.012	0.052	3.132	0.002
$n \cdot (m/2) \cdot 90$	TB_{FS} - TB_{FF}	0.082	0.243	0.063	0.102	8.280	0.000
	TB_{D} - TB_{FF}	0.091	0.254	0.070	0.111	8.753	0.000
	TB_{KK1} - TB_{FF}	0.184	0.311	0.159	0.209	14.455	0.000
	TB_{KK2}- TB_{FF}	0.068	0.221	0.050	0.086	7.554	0.000
	TB_{FS} - TB_{D}	-0.009	0.252	-0.029	0.012	-0.846	0.398
	TB_{FS} - TB_{KK1}	-0.102	0.249	-0.121	-0.082	-10.006	0.000
	TB_{FS} - TB_{KK2}	0.014	0.219	-0.004	0.031	1.564	0.118

6.4 An extensive computational evaluation

In this section, we perform a comprehensive comparison between the proposed mechanisms and the most promising algorithms in the literature. To the best of our knowledge, the last computational evaluation for the $Fm|prmu|C_{\max}$ was presented by [172], who carried out an exhaustive computational evaluation of the heuristics and metaheuristics published until 2004 for the PFSP to minimize makespan. A total of 18 heuristics and 7 metaheuristics were implemented and tested under the same conditions. Among them, two of these methods turned out to be the most efficient ones: The NEH heuristic [127] was clearly the most efficient among the constructive heuristics for the problem, and the Iterated Local Search [187] presented itself as the most efficient metaheuristic for the problem.

All experiments have been carried out on a computational cluster formed by 30 blade servers. Each server contains two Intel XEON E5420 processors running at 2.5 GHz and 16 Gbytes of RAM memory. However, the specific tests are performed on virtual machines running on this cluster. Each virtual machine runs Microsoft Windows 7 64 bit operating system and has one virtual processor and 2 GBytes of RAM. Several benchmarks have been used (see e.g. [190, 10, 158, 206, 30, 71]) in the literature to perform comparisons between algorithms. Among them, the most extended one is the benchmark from [190], i.e. $\mathcal{B}_1$. More recently, [198] proposed a more exhaustive orthogonal benchmark, $\mathcal{B}_2$. This benchmark was shown to have more discriminant power than that of [190]. In this section, both benchmarks are used to compare the algorithms.

In this computational evaluation, we use the $ARPD2$ indicator to measure the quality of the solutions and both $ARPT2$ and $ACPU$ indicators to measure the computational effort of the algorithms. Note that, despite the problems when using the $ACPU$ indicator to compare algorithms, it is included in the evaluation in order for one to be able to reproduce the original comparisons of the authors since all reviewed and implemented heuristics consider the $ACPU$ indicator. By means of these two indicators, let us denote a method as efficient in terms of $ARPT2$ ($ACPU$) when there is no other method with both less $ARPD2$ and less $ARPT2$ ($ACPU$).

Regarding the algorithms implemented in the computational evaluation, numerous algorithms have been proposed in the literature since the last computational evaluation of [172]. As a matter of fact, the number of metaheuristics is staggering and new proposals do not cease to appear. Therefore, only a selected number of them have been implemented with a cutoff date of December 2014.

Among the heuristics of Section 4.2, the FRB1 heuristic has been statistically improved by several heuristics (e.g. FRB4_6, FRB4_8) in the same paper. Additionally, the tie-breaking mechanisms of [35], [83], [84] as well as the original one of [127] are statistically outperformed by the tie-breaking mechanism proposed in Chapter 6 and therefore, heuristics NEHD, NEH1, NEHKK1 and NEH are removed from the analysis. A total of 19 remaining heuristics, are reimplemented here under the same conditions. They are: RAER, RAER-di, KKER, KKER-di, NEHR, NEHR-di, NEMR, NEMR-di, NEH-di, NEH1-di, NEHKK1-di, NEHKK2, NEHD-di, NEHFF, CL_{WTS}, FRB2, FRB3, FRB4_{k} ($k \in \{2, 4, 6, 8, 10, 12\}$) and FRB5 (these implemented heuristics are indicated in bold in Table 4.1). Note that, although the recent heuristic NEHI was initially discarded due to the fact that it was available online after December 2014, it also seems to be clearly inefficient according to the $ARPD2$ and average computational times (around 25 times bigger than the original NEH) shown in that paper (as compared to FRB4_{10} or FRB4_{12} for example). Note that there are two possible interpretations of RCT, the idle-time- based tie-breaking mechanism proposed by [162]. The authors state that this mechanism can be implemented in $O(n^2m^2)$. However, as explained in Section 6.3, it can be implemented in $O(n^3m)$ if the idle time between jobs is calculated only for the ties. Thereby, the complexity is $O(E \cdot n^2m)$ due to the need to evaluate a complete sequence for each iteration E times. Clearly, since the maximum number of tie-breaks is the number of jobs in the partial sequence, the complexity of this interpretation is $O(n^3m)$. In this Thesis, this latter interpretation is employed as it yields a lower computational effort for the benchmark $\mathcal{B}_1$, i.e. the constant affecting the complexity of the original interpretation is higher than that of the second one for each instance of the testbed.

Regarding metaheuristics, the decision about which ones to select is not trivial due to the large amount of existing methods. More precisely, only algorithms fulfilling the two following requirements are considered:

- $ARPD < 0.4$ (on T1 or T2, see Table 4.2) or
- $ARPD < 0.6$ and t parameter ≤ 90 (on T1).

In other words, we are demanding that for a metaheuristic to be selected it either has to have a good solution quality ($ARPD < 0.4$), or a reasonable solution quality in short-medium computational times ($ARPD < 0.6$ and t parameter ≤ 90). 11 metaheuristics fulfil these requirements: EXTS by [186]; HGA_RMA by [173]; MSSA by [132]; IG_RS$_{\mathrm{LS}}$ by [174]; IG$_{\mathrm{RIS}}$ by [138]; DDE$_{\mathrm{RLS}}$ by [138]; 3XTS by [37]; EDA$_{\mathrm{ACS}}$ by [195]; PSO by [218]; IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$) in Chapter 6; IG$_{\mathrm{RIS}}$(TB$_{\mathrm{FF}}$) by in Chapter 6. Among them, EXTS and HGA_RMA, are discarded since they are outperformed in statistically and/or sound comparisons by [37] and [174] respectively. Additionally, the H-CPSO algorithm by [77] has been implemented due to its promising results despite being outperformed by [138] under different stopping criteria and conditions. Furthermore, we have tried to implement the MSSA metaheuristic proposed by [132] without success, and after several unsuccessful attempts to make contact with the authors, we have removed this algorithm from the computational evaluation. Finally, metaheuristic HCS by [99] has also been included in the comparison since the $ARPD$ is very close to 0.4 and has not been shown to be outperformed by any other metaheuristic. Hence, a total of 10 metaheuristics have been chosen (these metaheuristics are indicated in bold in Tables 4.2 and 4.3).

When reimplementing the algorithms, doubts relating to the implementation were transmitted to the corresponding authors of the papers. All questions were successfully answered by the authors with the exception of [218], where no answer was received after several tries. Other specifics considered in order to carry out a fair comparison of the algorithms (apart from these of Section 3.4) are the following:

- The order of the instances was randomly chosen in the experiments to avoid systematic errors in the tests.
- The algorithms to be run in each instance are similarly randomized.
- For each instance, ten independent runs were performed for each heuristic to better fit the required CPU time (the average CPU time is kept).
- For each instance, five independent runs were carried out for each metaheuristic keeping the average values.

The results of these experiments –that have required a total CPU time effort of 348.74 days– are presented in the next two sections.

Constructive and improvement heuristics

The 19 heuristics implemented in this evaluation are first compared under the classic benchmark set of Taillard with 120 instances. The overall results are summarised in Table 6.10. The second, third and fourth columns represent the $ARPD2$, $ACPU$ and $ARPT2$ values for each algorithm in the set of instances of Taillard. $ARPD2$ values range from 3.89 (RAER heuristic) to 1.48 (FRB5 improvement heuristic) while $ARPT2$ values range from 0.02 to 7.23. Results are graphically shown in Figures 6.4 and 6.5 where the y-axis represents the $ARPD2$ for each heuristic and x-axis, respectively, represents $ACPU$ and $ARPT2$ in logarithmic scale. Although results obtained for the different time indicators are, in general, similar, there are also differences in the performance of the heuristics. Therefore, considering $ACPU$ as a measure of the computational effort as compared to $ARPT2$, FRB4_2 is faster than KKER-di, NEHR-di and RAER-di in addition to the CL$_{\text{WTS}}$ being slower than the FRB2 heuristic. According to indicators $ARPD2$ and $ARPT2$, the efficient heuristics are NEHKK2, NEHFF, NEHR-di (this last one would not be efficient considering $ARPD2$ and $ACPU$), FRB4_2, FRB4_4, FRB4_6, FRB4_{10}, FRB4_{12}, FRB3 and FRB5 (shown with a black circle in Figure 6.5). To be able to compare heuristics with different stopping criteria, they are grouped into clusters as a function of similar $ARPT2$ values (see Figure 6.5). Then, the heuristics of each cluster are compared with the best heuristic in terms of $ARPD2$ of that cluster, i.e. NEHFF, FRB4_2, FRB4_4, FRB4_6 and FRB4_{12}, respectively, for clusters 1, 2, 3, 4 and 5. The hypotheses to statistically check the efficiency of the heuristics are shown in Table 6.9, ordered by these clusters of heuristics. Since each heuristic is based on the original NEH algorithm and the same set of instances is used, the hypotheses of independence of the random variables (RDI) can be rejected (see third and fourth columns in Table 6.9). We use Holm's procedure [73] where p-values are calculated using the non-parametric Wilcoxon signed-rank test (see [138] for similar tests). Holm's procedure orders the p-values of the hypotheses in non-decreasing order (let us denote by i the position of the heuristic in that order) where the hypotheses are rejected if the p-value is lower than $\alpha/(k-i+1)$ (with k the number of hypotheses). Results are shown in Table 6.9. Considering a level of confidence of 0.05, several hypotheses of the NEHFF heuristic (cluster 1) have not been rejected (see e.g. NEHFF vs NEHR or NEHFF vs NEH-di). Additionally, there is no statistical significance to state that FRB4_6 and FRB4_{12} outperform FRB4_8 and FRB2 respectively.

A similar Pareto set is found when the heuristics are compared under the new set of instances $\mathcal{B}_2$. Average results are shown in Table 6.10. The last three columns represent the $ARPD2$, $ACPU$ and $ARPT2$ of each heuristic in that set of instances. Clearly, heuristics of complexity $O(n^3m)$ (CL_{WTS}, FRB2, FRB3 and FRB5) need proportionally more computational effort since this set of instances consid-

Table 6.9: Hypotheses, analysis of dependence and Holm's procedure on $\mathcal{B}_1$

Clusters	Comparison	Analysis of Dependence		Wilcoxon		Holm's Procedure		
		Correlation	Sig.	Sig.	Reject?	i	$\alpha/(k-i+1)$	Reject?
Cluster 1 (green)	NEHFF vs NEHKK2	0.891	0.000	0.015	R	11	0.0083	
	NEHFF vs NEH-di	0.923	0.000	0.054		14	0.0167	
	NEHFF vs NEHKK1-di	0.895	0.000	0.001	R	8	0.0056	R
	NEHFF vs NEHR	0.893	0.000	0.055		15	0.0250	
	NEHFF vs NEH1-di	0.910	0.000	0.021	R	12	0.0100	R
	NEHFF vs KKER	0.884	0.000	0.010	R	10	0.0071	
	NEHFF vs NEMR	0.869	0.000	0.006	R	9	0.0063	R
	NEHFF vs RAER	0.830	0.000	0.000	R	1	0.0031	R
Cluster 2 (blue)	$FRB4_2$ vs RAER-di	0.842	0.000	0.000	R	2	0.0033	R
	$FRB4_2$ vs NEHR-di	0.880	0.000	0.000	R	3	0.0036	R
	$FRB4_2$ vs KKER-di	0.877	0.000	0.000	R	4	0.0038	R
	$FRB4_2$ vs NEHD-di	0.860	0.000	0.000	R	5	0.0042	R
	$FRB4_2$ vs NEMR-di	0.864	0.000	0.000	R	6	0.0045	R
Cluster 3 (orange)	$FRB4_4$ vs CL_{WTS}	0.868	0.000	0.000	R	7	0.0050	R
Cluster 4 (red)	$FRB4_6$ vs $FRB4_8$	0.937	0.000	0.937		16	0.0500	
Cluster 5 (yellow)	$FRB4_{12}$ vs FRB2	0.927	0.000	0.041	R	13	0.0125	

Table 6.10: Summary of heuristics

Algorithm	$\mathcal{B}_1$			$\mathcal{B}_1$		
	ARPD2	ACPU	ARPT2	ARPD2	ACPU	ARPT2
NEHKK2	3.09	0.02	0.12	3.21	0.47	0.02
NEHFF	2.90	0.02	0.13	2.95	0.46	0.02
NEH-di	3.03	0.04	0.20	3.18	0.91	0.04
NEH1-di	3.11	0.04	0.20	3.15	0.91	0.04
NEHKK1-di	3.15	0.04	0.20	3.19	0.93	0.04
RAER	3.89	0.06	0.20	3.46	0.88	0.04
NEHR	3.05	0.06	0.21	3.16	0.93	0.04
KKER	3.15	0.06	0.21	3.15	0.93	0.04
NEMR	3.16	0.10	0.31	3.22	1.64	0.07
RAER-di	3.53	0.13	0.40	3.33	1.71	0.07
NEHR-di	2.85	0.13	0.40	3.02	1.82	0.07
KKER-di	2.86	0.12	0.42	3.00	1.79	0.07
NEHD-di	2.84	0.16	0.48	2.86	2.06	0.08
$FRB4_2$	2.33	0.11	0.48	2.57	2.81	0.13
NEMR-di	2.97	0.18	0.52	3.05	2.53	0.10
$FRB4_4$	2.13	0.18	0.68	2.31	4.65	0.20
CL_{WTS}	3.02	0.86	0.73	3.11	26.63	0.68
$FRB4_6$	1.91	0.25	0.89	2.17	6.42	0.28
$FRB4_8$	1.95	0.31	1.06	2.07	8.09	0.35
$FRB4_{10}$	1.87	0.37	1.20	1.97	9.87	0.43
$FRB4_{12}$	1.79	0.42	1.34	1.94	11.42	0.49
FRB2	1.93	0.64	1.68	1.74	37.97	1.40
FRB3	1.61	5.08	3.61	1.32	198.31	4.34
FRB5	1.48	14.59	7.23	1.04	753.56	14.36

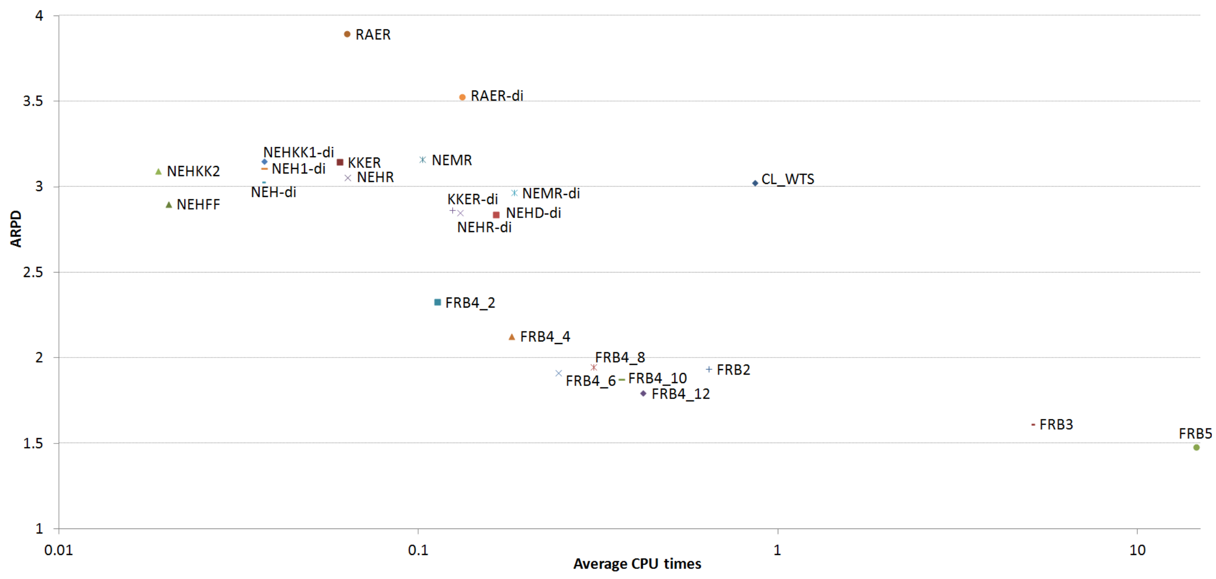

Figure 6.4: $ARPD2$ versus $ACPU$ of heuristics in logarithmic scale on $\mathcal{B}_1$.

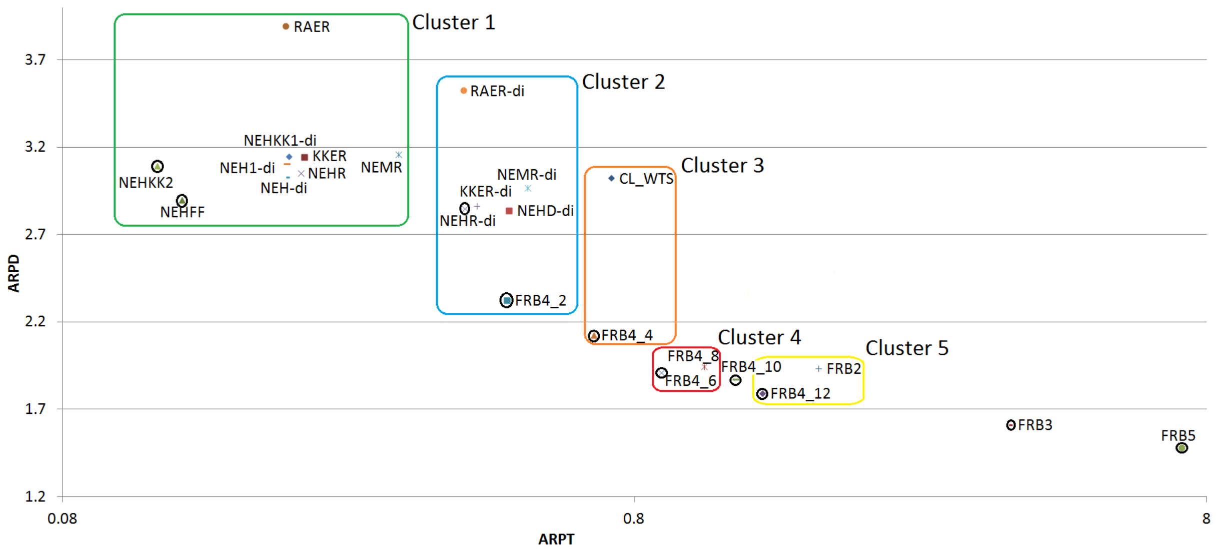

Figure 6.5: $ARPD2$ versus $ARPT2$ of heuristics in logarithmic scale on $\mathcal{B}_1$.

Table 6.11: Hypotheses, analysis of dependence and Holm's procedure on $\mathcal{B}_2$

	Comparison	Analysis of Dependence		Wilcoxon		Holm's Procedure		
		Correlation	Sig.	Sig.	Reject?	i	$\alpha/(k-i+1)$	Reject?
Cluster 1 (green)	NEHFF vs NEHKK2	0.950	0.000	0.000	R	1	0.0036	R
	NEHFF vs NEH-di	0.954	0.000	0.000	R	2	0.0038	R
	NEHFF vs NEHKK1-di	0.952	0.000	0.000	R	3	0.0042	R
	NEHFF vs NEHR	0.946	0.000	0.000	R	4	0.0045	R
	NEHFF vs NEH1-di	0.939	0.000	0.000	R	5	0.0050	R
	NEHFF vs KKER	0.952	0.000	0.000	R	6	0.0056	R
	NEHFF vs RAER	0.945	0.000	0.000	R	7	0.0063	R
Cluster 2 (blue)	FRB4$_2$ vs NEMR	0.943	0.000	0.000	R	8	0.0071	R
	FRB4$_2$ vs RAER-di	0.946	0.000	0.000	R	9	0.0083	R
	FRB4$_2$ vs NEHR-di	0.958	0.000	0.000	R	10	0.0100	R
	FRB4$_2$ vs KKER-di	0.953	0.000	0.000	R	11	0.0125	R
	FRB4$_2$ vs NEHD-di	0.948	0.000	0.000	R	12	0.0167	R
	FRB4$_2$ vs NEMR-di	0.952	0.000	0.000	R	13	0.0250	R
Cluster 3 (orange)	FRB4$_{12}$ vs CL$_{\mathrm{WTS}}$	0.942	0.000	0.000	R	14	0.0500	R

ers higher values of n and m than on $\mathcal{B}_1$. This increase in computational effort also results in a decrease in the $ARPD2$ of the heuristics with the exception of CL_{WTS}. Results are graphically shown in Figure 6.6 comparing $ARPD2$ versus $ACPU$ and in Figure 6.7 comparing $ARPD2$ versus $ARPT2$. In terms of $ARPD2$ and $ARPT2$, efficient heuristics are shown with a black circle in Figure 6.7. Note that regarding the NEH-based heuristics of [162] with direct and inverse approach, the best $ARPD2$ is now found by the NEHD-di heuristic instead of the NEHR-di. In order to compare the heuristics, we group them according to their $ARPT2$ (see Figure 6.7) and perform the same Holm's procedure [73] (hypotheses of independence can be rejected again). Note that heuristics FRB4$_k$ are not compared together since all are the same heuristics with a different input parameter. Results are shown in Table 6.11. Each p-value is 0.000 and all hypotheses are rejected using Holm's procedure. Thus, according to $ARPD2$ and $ARPT2$, statistically there is no reason to affirm that the NEHFF, FRB4$_{\mathrm{k}}$, FRB2, FRB3, FRB5 heuristics are not efficient heuristics within each cluster.

Metaheuristics

In Section 4.2, 10 metaheuristics were defined as the most promising according to the results shown in their papers. In this section, these metaheuristics are compared under the set of instances $\mathcal{B}_1$ and $\mathcal{B}_2$. Each metaheuristic is stopped using the same stopping criterion based on CPU time. More specifically, three different stopping criteria are applied, $t \cdot n \cdot m/2$ milliseconds with $t \in [30, 60, 90]$, which depends on the number of jobs and machines. Results are shown in Table 6.12. For both sets of instances, the best metaheuristics are those based on the Iterated Greedy (IG_RS$_{\mathrm{LS}}$) proposed by [174], see the results found by IG_RS$_{\mathrm{LS}}$, IG$_{\mathrm{RIS}}$, IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$) and IG$_{\mathrm{RIS}}$(TB$_{\mathrm{FF}}$) for example. These results are also confirmed by the DDE$_{\mathrm{RLS}}$, a discrete differential evolution algorithm which uses similar phases. Regarding $\mathcal{B}_1$, the $ARPD2s$ of Iterated Greedy metaheuristics for $t = 90$ is between 0.28 and 0.38 which clearly outperforms

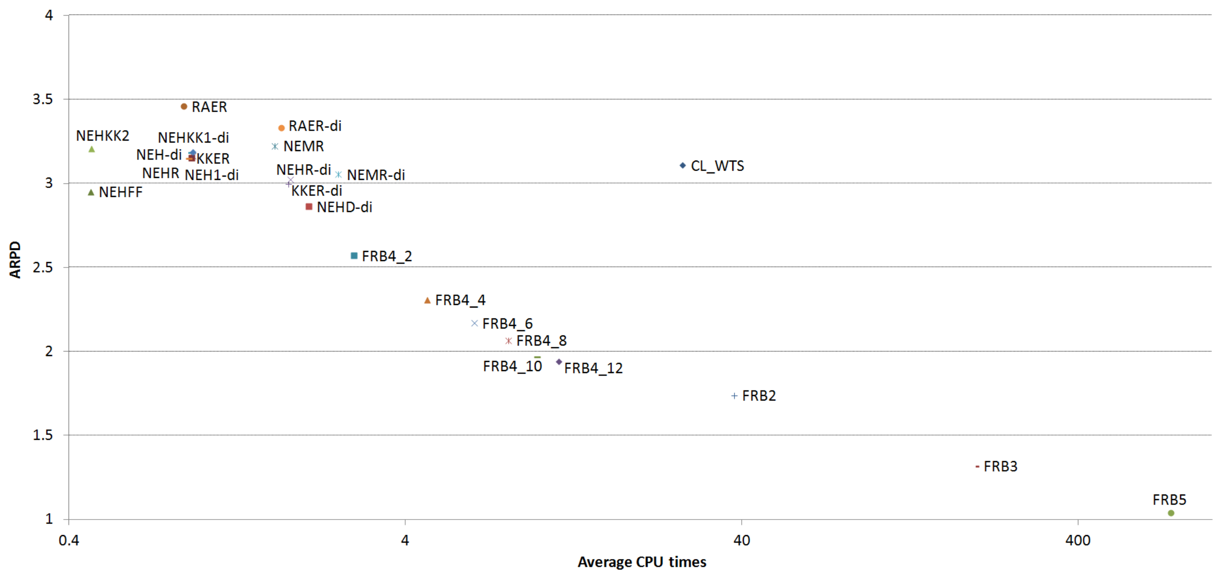

Figure 6.6: $ARPD2$ versus $ACPU$ of heuristics in logarithmic scale on $\mathcal{B}_2$.

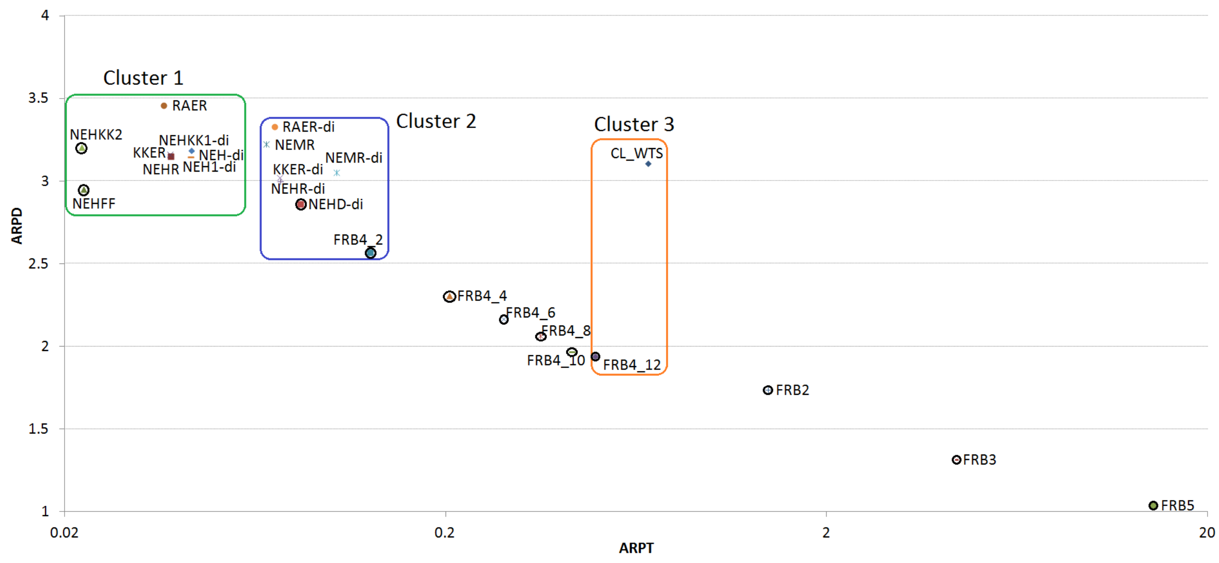

Figure 6.7: $ARPD2$ vs $ARPT2$ of heuristics in logarithmic scale on $\mathcal{B}_2$.

non IG-based metaheuristics (the $ARPD2s$ of 3XTS, H-CPSO, HCS and PSO are, respectively, 1.24, 0.70, 1.35 and 0.84 for $t = 90$). The best $ARPD2$ value is obtained by IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$), with 0.37, 0.32 and 0.37 for $t = 30$, $t = 60$ and $t = 90$ on Taillard's instances respectively. Let us highlight the fast convergence behaviour of the IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$) where the $ARPD2$ obtained for $t = 30$ is lower than or equal to every other metaheuristic for $t = 90$. Metaheuristics are compared with IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$) using the non-parametric Mann-Whitney test (see Table 6.13). With the exception of the IG-based algorithms IG_RS$_{\mathrm{LS}}$, IG$_{\mathrm{RIS}}$ and IG$_{\mathrm{RIS}}$(TB$_{\mathrm{FF}}$), each p-value on $\mathcal{B}_1$ is less than or equal to 0.032 regardless the value of t.

Regarding the benchmark $\mathcal{B}_2$, the superiority of the IG-based algorithms is more clear, as $\mathcal{B}_2$ include a wider range of values of n and m. Thereby, the differences between the $ARPD2$ values of the metaheuristics greatly increase with respect to the IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$) metaheuristic (see the difference of $ARPD2$ between 3XTS and IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$) is 0.96 on $\mathcal{B}_1$ and 2.10 on $\mathcal{B}_2$ for $t = 90$ for example). Statistical significance has been found for all metaheuristics with the exception of IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$) 0.000 being the maximum p-value (see Table 6.13). In view of the results, although there are many papers proposing metaheuristics, no metaheuristic statistically outperforms the original Iterated Greedy Algorithm (IG_RS$_{\mathrm{LS}}$) of [174] on $\mathcal{B}_1$, while only the Iterated Greedy variants proposed here statistically outperform IG_RS$_{\mathrm{LS}}$ on $\mathcal{B}_2$.

We have already discussed that many metaheuristics have been published since the last computational evaluation and review proposed by [172] (see Tables 4.2 and 4.3) and since the original Iterated Greedy algorithm proposed by [174]. On one hand, in view of Tables 4.2 and 4.3 only 11 metaheuristics have promising results in terms of quality of solutions and computational effort. On the other hand, in view of the results in this section, only the original IG_RS$_{\mathrm{LS}}$ and the IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$) algorithms are state-of-the-art methods. It follows that many metaheuristics were not state-of-the-art even at the time on their publication, a fact that strongly highlights the need for a review and framework for computational evaluation such as the one proposed here.

Comparison of heuristics with metaheuristics

Traditionally, researchers have focused either on finding efficient heuristics, or on obtaining the best metaheuristic for the problem. The former are implemented to find a good fast solution and/or a good initial seed sequence for the problem, while the latter are typically implemented to find better solutions using longer CPU times. As a consequence, typically both heuristics and metaheuristics have been separately evaluated and compared. In this section, we analyse both heuristics and metaheuristics together, as there

Table 6.12: Summary of $ARPD2s$ of the metaheuristics

Metaheuristic	Ref.	$\mathcal{B}_1$			$\mathcal{B}_2$		
		t=30	t=60	t=90	t=30	t=60	t=90
IG_RS$_{\text{LS}}$	[174]	0.47	0.40	0.37	0.96	0.77	0.67
IG$_{\text{RIS}}$	[138]	0.49	0.42	0.38	0.85	0.67	0.56
DDE$_{\text{RLS}}$	[138]	0.52	0.47	0.43	0.92	0.77	0.69
3XTS	[37]	1.64	1.34	1.24	2.89	2.65	2.47
H-CPSO	[77]	0.84	0.75	0.70	1.65	1.41	1.28
EDA$_{\text{ACS}}$	[195]	0.60	0.51	0.47	1.43	1.25	1.16
HCS	[99]	1.55	1.42	1.35	2.54	2.35	2.27
PSO	[218]	1.09	0.95	0.84	2.51	2.14	1.93
IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$)	Our proposal	0.37	0.32	0.28	0.60	0.46	0.37
IG$_{\text{RIS}}$(TB$_{\text{FF}}$)	Our proposal	0.42	0.34	0.31	0.61	0.47	0.38

Table 6.13: Comparison of metaheuristics using Mann-Whitney tests

Comparison	$\mathcal{B}_1$ (Sig.)			$\mathcal{B}_2$ (Sig.)		
	t=30	t=60	t=90	t=30	t=60	t=90
IG$_{\text{RIS}}$ vs IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$)	0.114	0.130	0.132	0.000	0.000	0.000
IG_RS$_{\text{LS}}$ vs IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$)	0.371	0.331	0.297	0.000	0.000	0.000
DDE$_{\text{RLS}}$ vs IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$)	0.011	0.007	0.013	0.000	0.000	0.000
3XTS vs IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$)	0.000	0.000	0.000	0.000	0.000	0.000
H-CPSO vs IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$)	0.000	0.000	0.000	0.000	0.000	0.000
EDA$_{\text{ACS}}$ vs IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$)	0.018	0.023	0.032	0.000	0.000	0.000
HCS vs IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$)	0.000	0.000	0.000	0.000	0.000	0.000
PSO vs IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$)	0.000	0.000	0.000	0.000	0.000	0.000
IG$_{\text{RIS}}$(TB$_{\text{FF}}$) vs IG_RS$_{\text{LS}}$(TB$_{\text{FF}}$)	0.533	0.526	0.556	0.638	0.816	0.711

are several heuristics requiring long CPU times and vice versa. Therefore, each heuristic is compared with one of the best metaheuristics, i.e. the iterated greedy IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$). In order to have a fair comparison, the metaheuristic is stopped at the CPU time used by each heuristic. These comparisons are performed using the set of instances $\mathcal{B}_1$ and $\mathcal{B}_2$. A summary of the results is shown in Table 6.14 as well as in Figures 6.8 and 6.9 for these benchmarks, respectively, where the dotted lines represent logarithmic trend lines for the heuristics and the red squares represent all values obtained by the IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$) metaheuristic. Note that the IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$) metaheuristic starts with the sequence obtained by the NEHFF heuristic and therefore, the NEHKK2 and NEHFF heuristics are not included in the comparison as they need shorter CPU times. For all other heuristics, the metaheuristic outperforms them in terms of $ARPD2$. All compared heuristics are outperformed by the IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$) metaheuristic, especially when compared on $\mathcal{B}_2$. The statistical significance of these comparisons is established by means of the non-parametric Mann-Whitney test since the normality and homoscedasticity assumptions are not fulfilled. Although no statistical significance is found for many of the comparisons on the Taillard instances (see heuristics by [150] which have $ARPD2$ values similar to or even better than those obtained by the IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$) metaheuristic for several problem sizes for example), on $\mathcal{B}_2$ each hypothesis is rejected, 0.001 being the highest p value. This Section highlights the exceptional performance of IG-based algorithms for short periods of time and also serves to classify IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$) as a state-of-the-art method for constructive and improvement heuristics.

Table 6.14: Comparison between heuristics and the best metaheuristic

Algorithm	$\mathcal{B}_1$							$\mathcal{B}_2$						
	Original Heuristics			IG_RS$_{LS}$(TB$_{FF}$)			Mann-Whitney	Original Heuristics			IG_RS$_{LS}$(TB$_{FF}$)			Mann-Whitney
	ARPD2	*ACPU*	*ARPT2*	*ARPD2*	*ACPU*	*ARPT2*	Sig.	*ARPD2*	*ACPU*	*ARPT2*	*ARPD2*	*ACPU*	*ARPT2*	Sig.
NEHKK2	3.09	0.02	0.12	—	—	—	—	3.21	0.47	0.02	—	—	—	—
NEHFF	2.90	0.02	0.13	—	—	—	—	2.95	0.46	0.02	—	—	—	—
NEH-di	3.03	0.04	0.20	2.53	0.06	0.22	0.045	3.18	0.91	0.04	2.55	1.42	0.06	0.000
NEH1-di	3.11	0.04	0.20	2.50	0.06	0.26	0.031	3.15	0.91	0.04	2.55	1.42	0.06	0.000
NEHKK1-di	3.15	0.04	0.20	2.52	0.06	0.23	0.020	3.19	0.93	0.04	2.55	1.47	0.06	0.000
RAER	3.89	0.06	0.20	2.50	0.09	0.27	0.000	3.46	0.88	0.04	2.53	1.61	0.07	0.000
NEHR	3.05	0.06	0.21	2.51	0.08	0.34	0.300	3.16	0.93	0.04	2.53	1.62	0.07	0.000
KKER	3.15	0.06	0.21	2.50	0.08	0.27	0.015	3.15	0.93	0.04	2.53	1.63	0.07	0.000
NEMR	3.16	0.10	0.31	2.39	0.12	0.34	0.002	3.22	1.64	0.07	2.43	2.11	0.09	0.000
RAER-di	3.53	0.13	0.40	2.36	0.15	0.42	0.000	3.33	1.71	0.07	2.44	2.04	0.09	0.000
NEHR-di	2.85	0.13	0.40	2.35	0.15	0.42	0.037	3.02	1.82	0.07	2.44	2.16	0.09	0.000
KKER-di	2.86	0.12	0.42	2.34	0.14	0.47	0.037	3.00	1.79	0.07	2.43	2.12	0.09	0.000
NEHD-di	2.84	0.16	0.48	2.30	0.17	0.50	0.028	2.86	2.06	0.08	2.41	2.42	0.10	0.000
NEMR-di	2.97	0.18	0.52	2.28	0.20	0.49	0.004	3.05	2.53	0.10	2.27	2.90	0.12	0.000
CL$_{WTS}$	3.02	0.86	0.73	2.05	0.84	0.73	0.000	3.11	26.63	0.68	1.60	24.84	0.64	0.000
FRB4$_2$	2.33	0.11	0.48	2.11	0.13	0.53	0.371	2.57	2.81	0.13	2.18	3.36	0.15	0.000
FRB4$_4$	2.13	0.18	0.68	1.98	0.19	0.68	0.445	2.31	4.65	0.20	1.98	5.16	0.22	0.000
FRB4$_6$	1.91	0.25	0.89	1.83	0.24	0.94	0.709	2.17	6.42	0.28	1.88	6.86	0.29	0.000
FRB4$_8$	1.95	0.31	1.06	1.75	0.30	1.15	0.329	2.07	8.09	0.35	1.80	8.47	0.35	0.000
FRB4$_{10}$	1.87	0.37	1.20	1.70	0.34	1.34	0.338	1.97	9.87	0.43	1.74	10.10	0.41	0.001
FRB4$_{12}$	1.79	0.42	1.34	1.67	0.37	1.46	0.497	1.94	11.42	0.49	1.69	11.57	0.47	0.000
FRB2	1.93	0.64	1.68	1.61	0.61	1.68	0.054	1.74	37.97	1.40	1.39	35.29	1.35	0.000
FRB3	1.61	5.08	3.61	1.40	5.06	3.76	0.174	1.32	198.31	4.34	1.11	197.65	4.32	0.000
FRB5	1.48	14.59	7.23	1.21	14.58	7.38	0.133	1.04	753.56	14.36	0.82	753.03	14.34	0.000

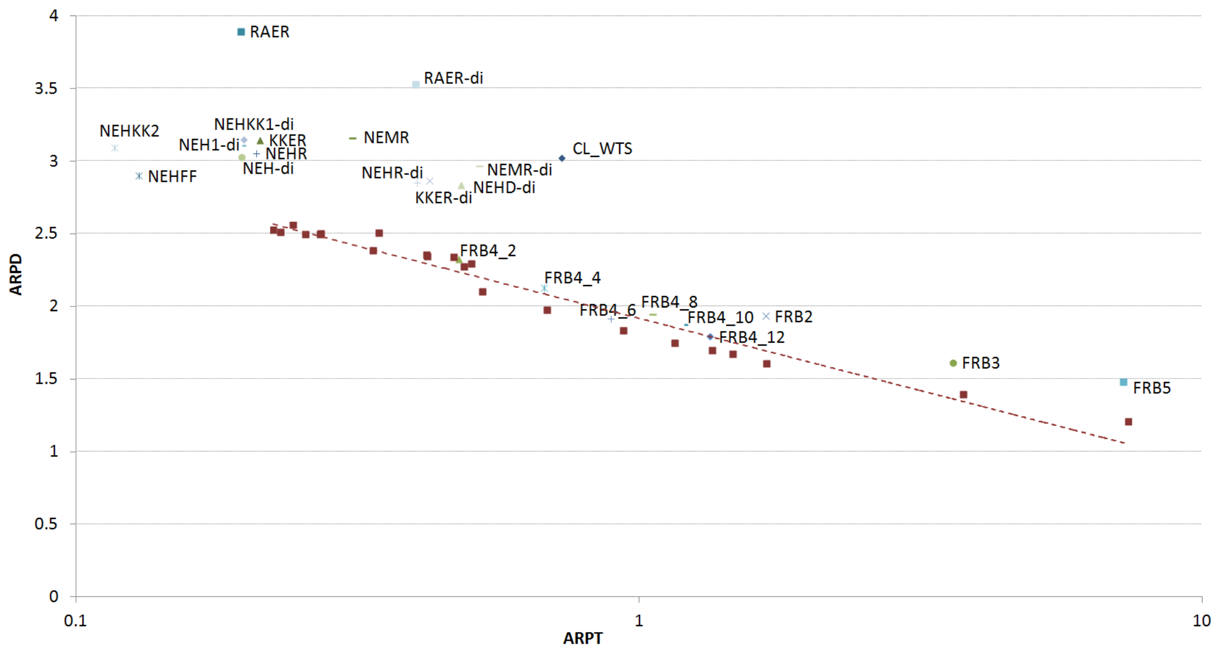

Figure 6.8: Heuristics versus $IG\_RS_{LS}(TB_{FF})$ on the set of instances $\mathcal{B}_1$. X-axis (variable $ARPT2$) is shown in logarithmic scale.

6.5 Conclusions

In this chapter, we have presented a new tie-breaking mechanism based on an estimation of the idle times of the different subsequences in order to pick the one with the lowest value of the estimation. This tie-breaking mechanism can be incorporated into the most efficient approximate procedures for the flowshop scheduling problem with makespan objective, resulting in statistically significant better results than existing tie-breaking mechanisms.

The proposed tie-breaking mechanism has been compared with the most promising tie-breaking mechanisms and algorithms in the literature. Since the last reviews in 2005, a large number of heuristics and metaheuristics have been proposed for the permutation flowshop scheduling problem to minimize makespan. Most of them are compared with other non-efficient algorithms and/or under uncomparable conditions. Thus, it was not clear which algorithms were state-of-the-art. In this chapter, an exhaustive evaluation of algorithms for the permutation flowshop is proposed, with special attention being paid to conducting a fair comparison of algorithms. The most promising ones, i.e. a total of 29 algorithms (19 constructive heuristics and 10 metaheuristics), have been implemented and compared under the same conditions. The comparisons have been done using the benchmarks $\mathcal{B}_1$ and $\mathcal{B}_2$. On one hand, the metaheuristics are compared under three different stopping criteria to analyse the evolution of the each algorithm with the computational effort. On the other hand, the comparison of (constructive and im-

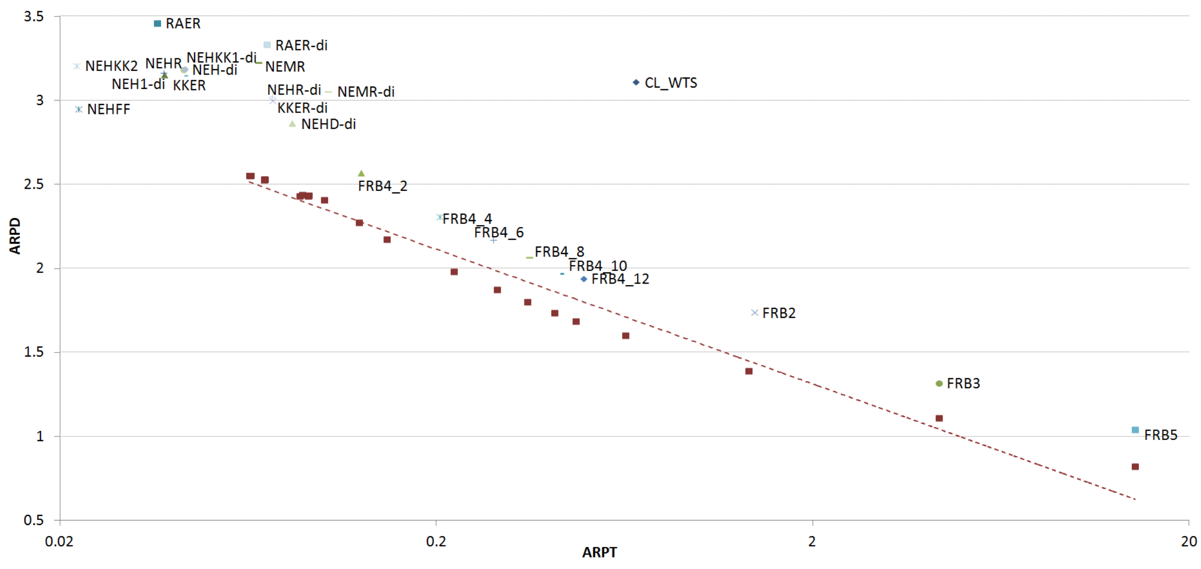

Figure 6.9: Heuristics versus IG_RS$_{\mathrm{LS}}$(TB$_{\mathrm{FF}}$) on the set of instances $\mathcal{B}_2$. X-axis (variable $ARPT2$) is shown in logarithmic scale.

provement) heuristics has been performed using two relative indicators to measure the quality of the solution and the computational effort in order to identify the efficient ones. Statistical analyses of the quality of the solutions have been carried out to study the efficiency of the heuristics as well as to compare the metaheuristics. Additionally, each heuristic has been compared with the best metaheuristic under the stopping criterion of the heuristic to analyze tentative best seed sequences for the metaheuristics. Therefore, we believe that this contribution may represent a starting point for future researchers who attempt to propose new algorithms for the permutation flowshop scheduling problem with makespan objective.

Among all coded metaheuristics, IG-based algorithms have been clearly identified as the most efficient metaheuristics for the problem. This fact is further confirmed since other well-performing metaheuristics also incorporate some part of the IG algorithm (see metaheuristics EDA_ACS or DDE_RLS for example). In particular, the proposed iterated greedy with the proposed tie-breaking mechanisms is the most efficient one. Additionally, the difference in solution quality between IG-based algorithms and other methods is even greater in the new set of instances $\mathcal{B}_2$ which also consider a higher number of jobs and machines, a fact which explains why some metaheuristics tested on just a subset of the instances $\mathcal{B}_1$ were found to be efficient ones at their time.

Regarding heuristics, most have been identified and classified as variations of the NEH algorithm. Among the 19 coded algorithms, only 5 heuristics (NEHFF, FRB4$_{\mathrm{k}}$, FRB2, FRB3 and FRB5) could be classified as efficient. Nevertheless, when they are compared with the best metaheuristic under the stopping criteria of the heuristic, all efficient heuristics have been outperformed by the metaheuristic, with the exception of NEHFF since that heuristic is the initial solution of the metaheuristic. Hence, this fact clearly indicates a way of proceeding when future new heuristics are proposed in the literature. From now, constructive and improvement heuristics should be directly compared either with the best metaheuristic under the same stopping criterion or with NEHFF with at least the same computational effort, as it might turn out that a few iterations of a good metaheuristic already give better results.

Chapter 7

PFSP to minimise total flowtime

7.1 Introduction

In this chapter, we propose a new heuristic that improves the results with respect to that by [108] both in terms of quality of the solutions and in CPU time (Objective SO5). Starting with this heuristic, we also propose an advance population-based constructive heuristic. With these new two heuristics, a completely new efficient Pareto set is obtained. The heuristics are tested on an extensive computational evaluation, comparing them against the set of 14 efficient heuristics (see Section 4.3).

The rest of the chapter is organised as follows: Section 7.2 analyses some issues related with the performance evaluation of the different heuristics for the problem. In Section 7.3 and 7.4, two new set of heuristics are presented for the problem. The computational evaluations are carried out in Section 7.5. Finally, conclusions are discussed in Section 7.6.

7.2 Implemented heuristics

As mentioned in the previous chapter, a great number of heuristics have been proposed for the problem. For a detailed presentation and evaluation of all these heuristics, we refer the interested reader to 4.3, and we will describe here only a sub-set which are found to be state-of-the-art and consequently are the ones used in this chapter for comparison.

From the conclusion of Section 4.5, it can be seen that LR is a key heuristic of complexity $O(n^3 \cdot m)$, playing a role similar to that of the NEH for makespan minimisation. For this latter problem, Taillard's accelerations (Section 2.2) showed that the complexity of the NEH can be reduced from $O(n^3m)$ to $O(n^2m)$ by using an acceleration mechanism, but unfortunately, such mechanism cannot be used to

Algorithm	LR	NEH	iRZ	FPE	VNS	Complexity
Raj		X				O($n^3 \cdot m$)
RZ			X			O($n^3 \cdot m$)
LR(x)	X					O($x \cdot n^3 \cdot m$)
RZ-LW			X			O($k \cdot n^3 \cdot m$)
LR-NEH(x)	X	X				O($x \cdot n^3 \cdot m$)
LR(n/m)-FPE(n)	X			X		O(n^4)
IC1	X		X			O($k \cdot n^3 \cdot m$)
IC2	X		X	X		O($k \cdot n^3 \cdot m$)
IC3	X		X	X (r)		O($k \cdot n^3 \cdot m$)
PR1(x)	X	X	X			O($x \cdot k \cdot n^3 \cdot m$)
PR2(x)	X	X			X	O($x \cdot k \cdot n^3 \cdot m$)
PR3(x)	X	X	X			O($x \cdot k \cdot n^3 \cdot m$)
PR4(x)	X	X			X	O($x \cdot k \cdot n^3 \cdot m$)

Table 7.1: Efficient heuristics [137] as variation/adaptation of primary procedures.

minimize flowtime. The only acceleration proposed is due to [96], who reported savings in the CPU time around 30-50%. Nevertheless, the complexity of the NEH remains the same and thus a way to reduce the complexity of efficient approximate algorithms for flowtime to $O(n^2m)$ has remained elusive.

We can employ the data from [137] to calculate the corresponding values of $ARPT1$ for each heuristic. The results are shown in Table 7.2, and are represented in two axis in Figure 7.1. The set of efficient heuristics according to the proposed approach is: Raj, $LR(1)$, RZ, $RZ-LW$, $LR-NEH(5)$, $LR-NEH(10)$, $LR-FPE$, $IC1$, $IC2$, $IC3$, $PR1(5)$, $PR1(10)$ and $PR1(15)$.

It can be checked in Table 7.2 that the alternative representation of efficiency is more complete in the sense that ten heuristics of the thirteen heuristics considered efficient using $ARPT1$ are indeed efficient for six or more problem sizes, whereas only three heuristics with less than six efficient sizes are included. Furthermore, the data in [137] expressed the CPU time with two decimals, therefore for some heuristics the CPU time is 0.00 in some problem sizes, and in this case it is not possible to establish a realistic trade-off between CPU time and $ARPD1$. More specifically, heuristics RZ and $RZ-LW$ should have several more efficient sizes due to the fact that their CPU time is 0.00 for the first 3-5 instance sizes (this may also happen with as $IC1$ or $IC2$, among others). It is worth noting that, using CPU time, six heuristics globally efficient are efficient for six or more problem sizes while there are eight heuristics which have less than six sizes for which they are efficient. The average number of efficient sizes for the fourteen efficient heuristics using CPU time is only 4.00, as compared to an average of 6.08 using $ARPT1$.

Furthermore, in order to re-assure that no heuristic is excluded by using the proposed indicator, we conduct a series of experiments to extend the comparison between heuristics $PR1$ and $PR2$. Note that, according to the results, $PR1$ seems to outperform $PR2$, but the latter is extremely efficient for the biggest instances (i.e. $100 \times 10, 200 \times 20$, and 500×20). In addition, the improvement phase of VNS used in

Algorithm	$ARPD1$	$ARPT1$	#Efficient Size $ACPU$	#Efficient Size $ARPT1$
Raj	5.02	-1.00	**7**	**7**
LIT	8.26	-0.96	0	0
SPD1	17.37	-0.97	0	0
SPD2	16.56	-0.97	1	1
RZ	2.65	-0.97	**3**	**3**
WY	2.83	-0.67	0	0
LR(1)	3.13	-0.99	**7**	**7**
LR(n/m)	2.29	-0.89	2	2
LR(n)	2.09	0.27	0	0
NEH	4.03	-0.99	1	1
FL	1.99	-0.41	0	0
RZ-LW	1.29	-0.82	4	**4**
FL-LS	1.22	0.11	0	0
LR-NEH(5)	1.84	-0.94	**8**	**8**
LR-NEH(10)	1.75	-0.90	**6**	**6**
LR-NEH(15)	1.72	-0.78	**3**	3
LR-FPE	1.14	-0.81	**7**	**7**
LR-BPE	1.23	-0.80	5	5
IH7	1.43	-0.25	0	0
IH7-FL	1.30	-0.22	0	0
C1-FL	1.72	-0.35	0	0
C2-FL	0.95	0.26	1	1
IC1	0.81	-0.75	6	**6**
IC2	0.66	-0.62	8	**8**
IC3	0.62	-0.26	6	**6**
PR1(5)	0.50	-0.15	7	**7**
PR1(10)	0.39	0.79	4	**4**
PR1(15)	0.33	1.43	**6**	**6**
PR2(5)	0.51	0.54	**3**	3
PR2(10)	0.41	1.90	3	3
PR2(15)	0.36	2.93	**1**	1
PR3(5)	0.51	0.04	**4**	4
PR3(10)	0.46	0.91	2	2
PR3(15)	0.45	1.64	1	1
PR4(5)	0.54	0.69	**1**	1
PR4(10)	0.45	2.00	**0**	0
PR4(15)	0.41	2.98	**0**	0

Table 7.2: Summary of average results of the heuristics implemented in [137] using $ARPT1$. Last two columns show the number of problem sizes where each heuristic is efficient using both $ACPU$ and $ARPT1$. In bold it is indicated when the heuristic is efficient when averaged for the 120-instances using either CPU time or $ARPT1$.

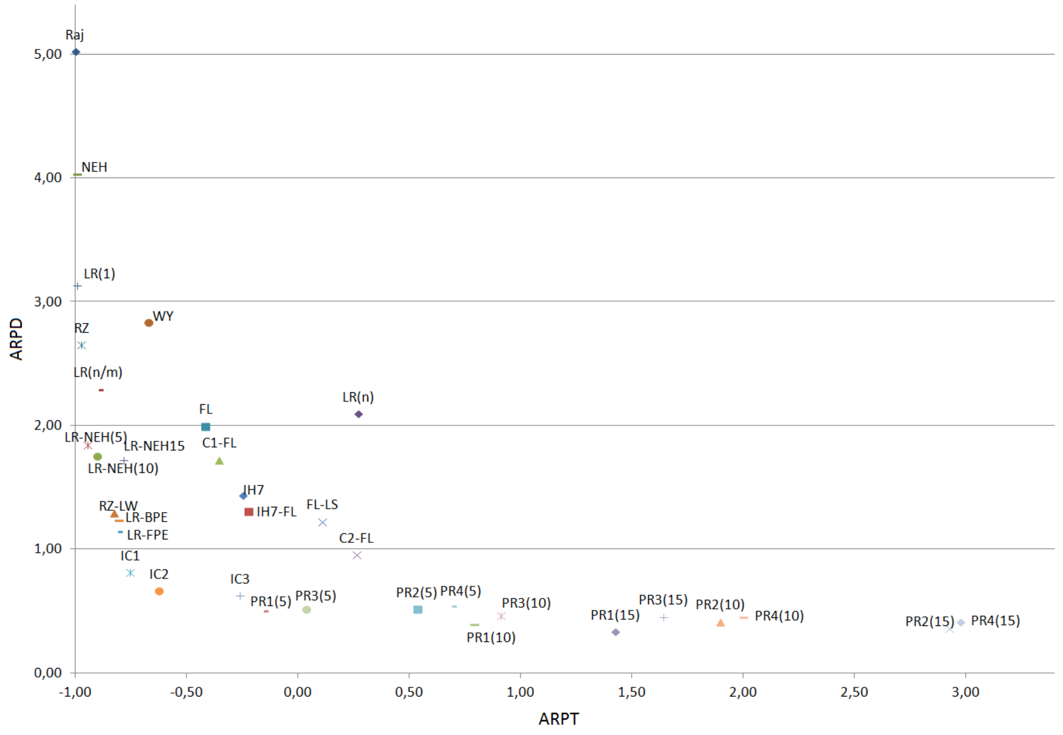

Figure 7.1: Pareto set using the $ARPT1$

[137] (which includes pairwise interchanges and insertion movements in an single position) is used as the first neighborhood. This may affect the performance of the PR2 heuristic, as more exhaustive insertion movements (such as iRZ) could be also considered as the first neighborhood so the performance of this so-obtained heuristic (labelled $PR2A$ in the following) is improved. Note however, that $PR2A$ would be much slower than $PR1$ and $PR2$.

Thus, these three heuristics ($PR1$, $PR2$ and $PR2A$) are further compared using Taillard's testbed. To obtain more points in the Pareto approximation, we extend the initial range of the stopping criteria and that of parameter x for the fastest heuristics (i.e. $PR1$ and $PR2$). More specifically, we test the following stopping criteria: $0.01 \cdot n \cdot m$, $0.05 \cdot n \cdot m$, and $0.1 \cdot n \cdot m$ for all three heuristics, and also $0.2 \cdot n \cdot m$ for $PR1$. Regarding the values of x, $x \in \{5, 10, 15, 20, 25\}$ is used for $PR1(x)$, $x \in \{5, 10, 15, 20\}$ is employed for $PR2(x)$, whereas $x \in \{5, 10, 15\}$ is used for $PR2A(x)$. A clear dominance of $PR1(x)$ over $PR2(x)$ and $PR2A(x)$ is obtained from these results (summarised in Figure 7.2, where the dotted lines represent quadratic polynomial trend lines for the heuristics). As a result, in the computational experiments in the following sections, $PR2(x)$ and $PR2A(x)$ will be excluded.

In view of the discussion and the results in this section, it seems clear that the evaluation of the performance of heuristics for the problem is not trivial, and that both the quality of the solutions and

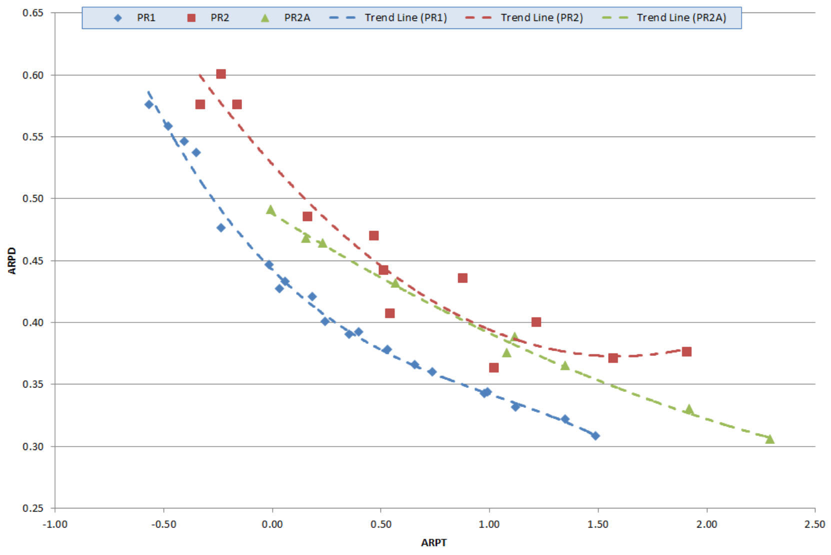

Figure 7.2: Comparison heuristics $PR1$, $PR2$ and $PR2A$

the computational effort should be taken into account. Building upon the work by [137], an indicator for measuring the computational effort has been proposed. This indicator, although not perfect, presents more consistency between the disaggregated (i.e. at instance size level) and aggregated (overall) results. Nevertheless, since the state-of-art evaluation of heuristics for flowtime (that of [137]) was done using CPU time as indicator, we report the subsequent results in this chapter using also their scheme.

7.3 A simple constructive heuristic

The proposed heuristic –denoted in the following as $FF(x)$– uses the idea present in LR of decreasing number of the evaluations of solutions, beginning with $n-1$ evaluations and finishing with 0. Thereby, the heuristic is composed of $n-1$ step with a maximum of $n-1$ evaluations. However, in contrast to the LR, we focus in the evaluation of each solution trying to reduce the complexity of the algorithm. When introducing a new job at the end of the sequence, there are three elements to be considered:

- *Idle time induced by the newly inserted job.* This idle time influences the next jobs to be inserted. Clearly, this influence decreases with each step (being 0 in the last step). Its calculation has a complexity $O(m)$ since only the completion time of the preceding job in each machine is required. For a given iteration k, this data is known from the previous iteration (or zero if it is the first job).

- *Completion time in machine m of the newly inserted job.* Its influence on the total flowtime is clear since the completion time of each job in machine m is included in the objective function. This data can be calculated within $O(m)$ using the completion time on each machine of the preceding job.

- *Completion time in machine m of the artificial job.* It seems that it influences the objective function in an indirect manner, as it is an indicator of the completion time in machine m of the yet unscheduled jobs. It is thus convenient to ensure that the unscheduled jobs will not have a very large completion time in machine m. However, the calculation of this completion time has a complexity of $n \cdot m$.

As in the *LR* heuristic, we intend that, once a job is scheduled in a position, it stays in this position, then choosing the adequate position of a job is critical. The problem thus lies in weighting the influence of the aforementioned elements. To do so, we use two parameters (a and b), to balance the first two elements, i.e. idle time and completion time of the newly inserted job. In contrast, we leave aside the third element (completion time of the artificial job), since its influence on the objective function is not as direct as the other two elements, and its consideration would increase the complexity of the algorithm to $n^3 \cdot m$.

More specifically, the proposed heuristic is as follows:

1. Sort the jobs according to a non descending order of indicator $\xi^{'}_{j0}$ (see equation 7.1), breaking ties in favor of jobs with lower $IT^{'}_{j,0}$ (see equation 7.2). Let us denote by I the so-obtained vector
2. Obtain x partial sequences π^i ($i = 1, \ldots, x$) of length 1, where the first (and only) job of sequence π^i is the job in position i in I. Store in U^i the jobs not scheduled in π^i.
3. For $k = 1$ to $n - 1$:
   (a) For each partial sequence π^i, remove from U^i the job for which the minimum value of $\xi^{'}_{j,k}$ (see equation 7.1) is found and place it in the last position of π^i.
4. Return the (final) sequence π^i yielding the lowest completion time.

Therefore, the proposed procedure begins with x sequences (π^i with $i \in [1, x]$) with only one job. The first job of each sequence π^i is the job in position i of a vector sorted in non descending order of indicator $\xi^{'}_{j0}$ (equation 7.1) breaking ties in favor of jobs with higher $IT^{'}_{j,0}$ (equation 7.2) and each final sequence π^i is obtained adding one by one jobs to the last position of the vector.

Let us denote by k the size of the vector in each step until the vector reaches the n jobs. To insert a new job j ($j \in U^i$) in each sequence π^i, one of the unscheduled jobs of each sequence, U^i, is removed according to an ascending index of a complexity m, $\xi^{'}_{j,k}$. This index is based on $IT^{'}_{j,k}$ the weighted idle time between the job in position k and the new job j to be inserted, and on the makespan of the sequence when inserting job j, $C_{m,j}$. For each job $j \in U^i$, $\xi^{'}_{j,k}$ is calculated as follows:

$$\xi^{'}_{j,k} = \frac{(n-k-2)}{a} \cdot IT^{'}_{j,k} + AT^{'}_{j,k} \tag{7.1}$$

Notation using LR	Notation using FF
LR(1)	FF(1)
LR-NEH(5)	FF-NEH(5)
LR-NEH(10)	FF-NEH(10)
LR(n/m)-FPE(n)	FF(n/m)-FPE(n)
IC1	FF-IC1
IC2	FF-IC2
IC3	FF-IC3
PR1(5)	FF-PR1(5)
PR1(10)	FF-PR1(10)
PR1(15)	FF-PR1(15)

Table 7.3: Notation for the heuristics using the proposed heuristic FF

where $AT^{'}_{j,k}$ and $IT^{'}_{j,k}$ are defined as follows:

$$IT^{'}_{j,k} = \sum_{i=2}^{m} \frac{m \cdot max\{C_{i-1,j} - C_{i,[k]}, 0\}}{i - b + k \cdot (m - i + b)/(n-2)} \tag{7.2}$$

$$AT^{'}_{j,k} = C_{m,j} \tag{7.3}$$

being a and b the aforementioned parameters to balance the influence of idle times and completion time of the newly inserted job. Note that by avoiding the calculation of the completion time of the artificial job p (C_{mp}), the complexity of the algorithm decreases from $n^3 \cdot m$ to $n^2 \cdot m$, a complexity n times lower than the fastest heuristics in the efficient set by [137].

As explained in Section 7.2, 10 of the 13 efficient heuristics using ARPT1 are based on LR. All of these 10 heuristics can be reimplemented using FF instead of LR. The notation for this set of heuristics is shown in Table 7.3. Additionally, due to the decrease in complexity, $FF(x)$ can be implemented for larger values of x. Note that $LR(n/m) - FPE(n)$ has a greater complexity than $LR(n/m)$, i.e. $O(n^4)$. Once LR is replaced by FF, $FF(n/m) - FPE(n)$ has a lower complexity and it can be interesting to perform the heuristic $FF(x) - FPE(y)$ for more values of both x and y since e.g. now $FF(1) - FPE(1)$ is also $O(n^2 \cdot m)$.

Prior to conducting these experiments, the best values for parameters a and b have to be found. To do so we carry out some computational experiments where different values are tried. After a first screening where different ranges of values were discarded, a multi-factor Analysis of Variance (ANOVA) was performed for $a \in \{1, 2, 3, 4\}$, $b \in \{0, 0.5, 1\}$ on calibration benchmark $\mathcal{B}_{C1}$. The results in Table 7.4 show that all parameters n, m, a and b are statistically significant. To determine the best level for each parameter, a Least Significant Difference (LSD) interval for each one is carried out (see Figure 7.3). Although from this figure it may seem that there is a monotonic trend for both a and b, further tests with

Source	Sum of Squares	Df	Mean Square	F-Ratio	p-Value
Main Effects					
n	52.169	4	13.042	20.755	0.000
m	26.724	2	13.362	21.264	0.000
a	7.711	3	2.570	4.090	0.007
b	10.763	2	5.381	8.564	0.000
Interaction					
$a * b$	0.114	6	0.019	0.030	1.000
$m * a$	6.087	6	1.015	1.614	0.139
$n * a$	6.063	12	0.505	0.801	0.647
$m * b$	5.548	4	1.387	2.207	0.066
$n * b$	13.679	8	1.710	2.721	0.006
$n * m$	110.808	8	13.851	22.042	0.000
Residual	1095.905	1744			
Total (corrected)	1335.571	1799			

Table 7.4: ANOVA for the parameters n, m, a, b

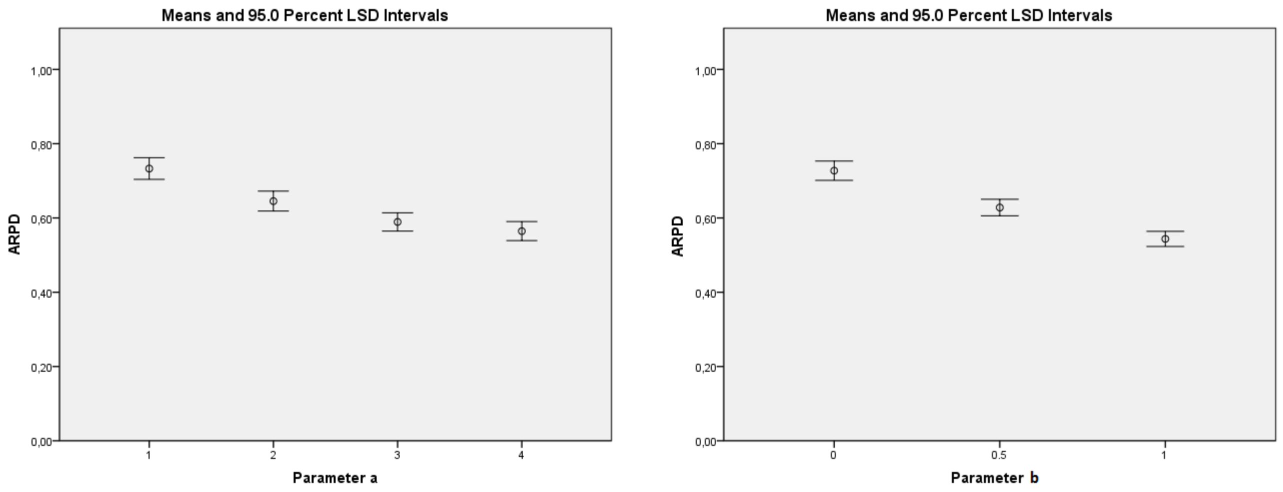

Figure 7.3: LSD intervals of the RPD1 for each level of the parameters a and b.

$a > 4$ and $b > 1$ did not produce better results. Therefore, $a = 4$ and $b = 1$ were used for the new set of heuristics $FF, FF - FPE, FF - NEH, FF - ICx, FF - PR1(x)$ in the next section.

7.4 A population-based constructive heuristic

In this section, we propose a Population-based Constructive Heuristic –denoted PCH–, for the PFSP to minimise total flowtime. PCH works with several individuals in parallel in each iteration. The number of individuals is controlled by the parameter x. The heuristic operates performing $n - 1$ iterations. At iteration k, each individual l ($l \in \{1, \dots, x\}$) is formed by a set, S_k^l, of k scheduled jobs (s_{jk}^l denotes the job placed in position j of individual l in iteration k). Consequently, for each individual l in iteration k there is a set U_k^l of $n - k$ unscheduled jobs. Let us denote u_{jk}^l the jth unscheduled job of individual l in iteration k.

For each iteration $k \in \{1, \ldots, n-1\}$, $|U_k^l|$ candidates can be obtained from each individual l by inserting each one of the jobs in U_k^l in position $k+1$ of S_k^l. In total, $(n-k) \cdot x$ candidates can be obtained. The idea is to retain the best x candidates as the next iteration individuals. However, comparing candidates may be or may be not straightforward depending on the specific situation:

- If the candidates to be compared have been obtained by appending different jobs in U_k^l to a same individual S_k^l, then their corresponding partial sequences are identical with the exception of the last job. Therefore, they can be compared in terms of the completion time of the added job, or of the new idle time induced by the added job.

- If the candidates to be compared have been obtained from different individuals –e.g. one candidate is the subsequence $(1, 2)$, and other candidate is subsequence $(2, 3)$–, both the scheduled and the unscheduled jobs are different for each candidate. In such case, it is useless to perform a straightforward comparison among candidates taking into account either the job to be inserted, or just the scheduled jobs.

Clearly, the key to select the best x candidates is to be able to compare partial sequences composed of different jobs. Since in iteration k, a candidate partial sequence is formed by individual S_k^l plus a job inserted in position $k+1$, both the individual and the inserted job would contribute to the value of the flowtime of a final sequence obtained from this candidate.

Regarding the contribution of the inserted job, there are two elements that largely influence the value of the sum of completion times in the complete sequence (see Section 7.3), i.e.: the weighted idle time induced by the new job u_{jk}^l inserted, and the completion time of the new job u_{jk}^l. Note that the evaluation of these elements can be done in $O(m)$.

Regarding the contribution of individual l in iteration k to the flowtime of the final sequence –denoted F_{kl} or forecast index in the following–, it is clear that such contribution is related to both scheduled and unscheduled jobs. On one hand, the contribution due to the scheduled jobs can be evaluated by means of a function of the idle times and completion times of the previous jobs. On the other hand, an "artificial" completion time, denoted as $CT\lambda_{kl}$, can be used to identify the contribution of the unscheduled jobs. Therefore, the forecast index will be used as an indicator to take into account the scheduled and unscheduled jobs of each individual. Note that this type of indicator is different that the traditional fitness function employed in the iterated population-based algorithms, since the latter always work with complete sequences, and thus different individuals contains the same jobs in different order. The calculation of F_{kl} is developed in Section 7.4.

Hence, steps of the constructive heuristic can be summarised as follows:

- Obtain initial individuals
- During n iterations:
  - Generate candidates
  - Evaluate candidates
  - Select the best x candidates
  - Update forecast index

These steps are elaborated in the next subsections.

Generation of initial individuals

Jobs are initially sorted in non descending order of indicator $\xi^{'}_{j,0}$ (see Section 7.3) breaking ties in favor of jobs with lower $IT^{'}_{j,0}$ as in FF heuristic. Let us denoted by α_i ($\alpha_i := \alpha_1, \alpha_2, ..., \alpha_n$) the component i of that sorted list. Hence, to obtain the first x individuals (consisting of one job), job in position l of the sorted list is placed in the first position of the partial sequence $s^l_{1,1}$ of the l individual ($s^l_{1,1} = \alpha_l$). The rest of the jobs form the unscheduled jobs of the individual, $u^l_{j,1}$ with $j \in \{1, \dots, n-1\}$ for each individual l.

Candidates generation

New candidates are formed by adding an unscheduled job at the end of the partial sequence of each individual. More specifically, from each individual $l \in \{1, \dots, x\}$, $n-k$ candidates are obtained at iteration k where each candidate j is obtained from individual l by adding the jobs in U^l_k at the end of the scheduled jobs.

Candidates evaluation

Once candidates are formed, they are evaluated. This evaluation is performed taking into account two factors:

- Influence from the individual: As already discussed, the influence of individual l in iteration k is measured by means of the forecast index F_{kl} which is explained further in Section 7.4.

- Influence from the inserted job: This influence is due to the new job inserted at the end of the scheduled jobs. This influence is measured by CT_{jkl} the completion time of the unscheduled job j, which is the additional completion time incurred when inserting job j in the individual, i.e.:

$$CT_{jkl} = C_{mj}$$

and by IT_{jkl} the weighted idle time induced by the insertion of job j (see 7.2).

Hence, in iteration k, given an individual l, the insertion of unscheduled job j is evaluated according to the following index:

$$B_{jkl} = F_{kl} + c \cdot CT_{jkl} + IT_{jkl} \cdot (n - k - 2) \tag{7.4}$$

Note that parameter c has been introduced in the expression in order to balance the completion time and the idle time of the new introduced job (in Section 7.4, the calibration of this parameter is addressed). Additionally, the idle time is weighted by $(n-k-2)$ to decrease its importance as indicator as the sequence contains more jobs.

Candidates selection

The procedure to select the candidates that would constitute the individuals of the next iteration is very simple. We adopt an elitist selection procedure where the x candidates with the lowest values of B are selected, i.e. in iteration k we look for the combination of j and l achieving the lowest values of B_{jkl} as defined in 7.4. The rest of candidates are removed from the population, and the chosen candidates are denoted as the individuals for the next iteration. Let us denote by $branch[l^{'}]$ and $job[l^{'}]$ the value of l and j respectively of the $l^{'}$th best B_{jkl} in iteration k.

Forecasting phase

The Forecast Index, F, is used to be able to compare candidates with different un- and sequenced jobs. It considers:

1. the idle time of each scheduled job in the individual,
2. the completion time of each scheduled job in the individual, and
3. the completion time of the unscheduled jobs in the individual.

The influence of 1) and 2) changes across the iteration of the algorithm. Recall that the influence of the idle time allows us to compare individuals with different jobs. In the first iterations there are few sequenced jobs, and these sequenced jobs may be quite different. Therefore, the idle time between jobs is expected to have a larger influence in the comparison between individuals, as compared to the sum of completion times (which is strongly schedule-dependent). In contrast, in the last iterations the individuals are almost complete sequences, so they are very similar in terms of scheduled jobs and therefore, a direct evaluation of the completion times of the jobs to compare individuals would be more related to the final objective. Thereby, in the expression of the forecast index, the cumulated idle time, denoted as SIT (11.10) would be reduced with the number of scheduled jobs (it is multiplied by $n - k - 2$), while the cumulated completion time, so-called SCT (11.12), would remain the same in the formation of the final individual. More specifically:

$$SIT_{k,l'} = \frac{n-b}{n} \cdot \left[SIT_{k-1,branch[l']} + IT_{job[l'],k,branch[l']} \cdot (n-k-2) \right], \forall k = \{1, \ldots, n-1\}, l^{'} = \{1, \ldots, x\} \tag{7.5}$$

$$SCT_{k,l'} = SCT_{k-1,branch[l']} + CT_{job[l'],k,branch[l']} + CT\lambda_{k,branch[l']}, \forall k = \{1, \ldots, n-1\}, l^{'} = \{1, \ldots, x\} \tag{7.6}$$

where $SIT_{0,l'} = SCT_{0,l'} = 0$, $\forall l^{'} = \{1, \ldots, x\}$ and $CT\lambda_{k,l}$ is the completion time of the individual l in the iteration k of an artificial job placed at the end of the sequence with processing times equal to the average processing times of the unscheduled jobs ($u^{l}_{j,k}$ $\forall j$).

Taking these indicators into account, the forecast index can be then defined as follows:

$$F_{k,l'} = a \cdot SCT_{k,l'} + SIT_{k,l'}, \forall k = \{1, \ldots, n-1\}, l^{'} = \{1, \ldots, x\}$$

where a, and b are parameters designed to better balance the components of the forecast index. Parameter a balances the influence of SIT and SCT. Parameter b is introduced in fraction $(n-b)/n$ of SIT in order to diminish the weight of idle time with the increase of iterations, given that 1) the idle time of the last jobs is less important than that of the first ones given the flowtime objective, and 2) the importance of the cumulated idle time as indicator also decreases as the number of scheduled jobs is higher.

The calibration of a and b is discussed in Section 7.4.

An example of the algorithm is presented in Figure 7.4. We use the third instance of the benchmark $\mathcal{B}_1$ but removing last 16 jobs and considering only the first 4 ones. Individuals are shown in lilac while

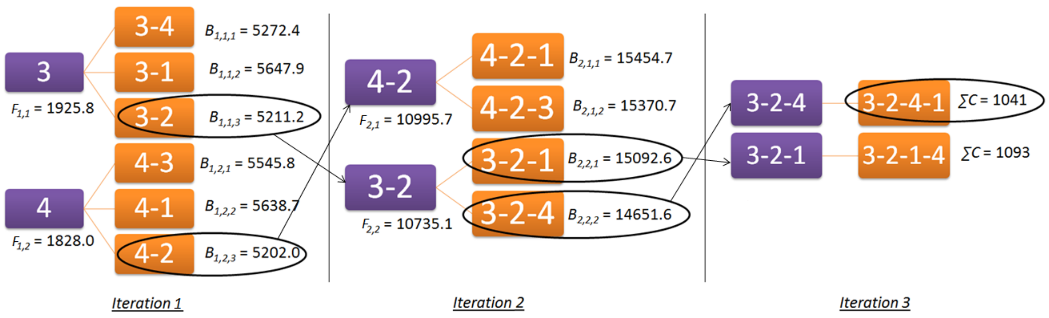

Figure 7.4: Example of PCH

Source	Significance
Parameter a	0.000
Parameter b	1.000
Parameter c	0.007

Table 7.5: Kruskal-Wallis for the parameters d, L and T

candidates are shown in orange.

The pseudo-code of the algorithm is shown in Figure 7.5.

Experimental parameter tuning

Parameters a, b and c have been included to better adjust the performance of the proposed heuristic. In this subsection, a full factorial design of experiments is performed to set up proper values for these parameters. For each of them, the following levels are tested

- $a \in \{1, 3, 5, 7, 9, 11, 13\}$
- $b \in \{0, 1, 2, 3, 4, 5, 6\}$
- $c \in \{1, 3, 5, 7, 9, 11, 13\}$

representing 343 combinations of values. Each combination is tested on calibration benchmark $\mathcal{B}_{C1}$. A non-parametric Kruskal-Wallis analysis is performed since normality and homoscedasticity assumptions required for ANOVA were not fulfilled. In the experiments, $x = n/10$ in order to avoid excessive CPU time requirements in the parameter tuning. Results are shown in Table 7.5, indicating that there are significant differences between the levels of parameters a and c, but not for parameter b. The best combination is obtained for $a = 9$, $b = 3$ and $c = 7$. These values are used for the PCH heuristic in the next section regardless the value of x.

```
Procedure PCH(x)
    //Initial Order
    Determination of $IT^{'}_{j,0}$, $CT^{'}_{j,0}$ and $\xi^{'}_{j0}$;
    $IT_{j,0,l} = IT^{'}_{j,0}$ and $CT_{j,0,l} = CT^{'}_{j,0}$ $\forall l$;
    $\alpha :=$ Jobs ordered according to non-decreasing $\xi^{'}_{j,0}$ breaking ties in favor of jobs with lower $IT^{'}_{j,0}$;
    Update $S^l_1$ ($s^l_{1,1} = \alpha_l$) $\forall l$ and $U^l_1$ with the remaining jobs.
    Determination of $CT\lambda_{0,l}$ $\forall l$. Note that the processing times of the artificial job for individual $l$ is
    equal to the average processing times of all jobs with the exception of $s^l_{1,1}$;
    for $l = 1$ to $x$ do
        $SIT_{1,l} = \frac{n-b}{n} \cdot \big(IT_{alpha[l],0,l} \cdot (n-0-2)\big)$;
        $SCT_{1,l} = CT_{alpha[l],0,l} + CT\lambda_{,0,l}$;
        $F_{1,l} := a \cdot SCT_{1,l} + SIT_{1,l}$;
    end
    for $k = 1$ to $n-1$ do
        //Candidates Creation
        Determination of $IT_{jkl}$, $CT_{jkl}$;
        //Candidates Evaluation
        $B_{jkl} := F_{kl} + c \cdot CT_{jkl} + IT_{jkl}$, $\forall l \in [1,x]$ and $\forall j \in [1, n-x]$;
        //Candidates Selection
        for $l^{'} = 1$ to $x$ do
            Determination of the $l^{'}$-th best candidate according to non-decreasing $B_{jkl}$ in iteration $k$.
            Denote by $branch[l^{'}]$ the value of the index $l$ of that candidate and by $job[l^{'}]$ the value of $j$;
        end
        //Forecasting Phase. Update of the Forecast Index
        for $l^{'} = 1$ to $x$ do
            Update $S^{l'}_{k+1}$ and $U^{l'}_{k+1}$ by removing job $u^{branch[l']}_{job[l'],k}$ from $U^{branch[l']}_k$ and including in $S^{branch[l']}_k$.
            Determination of $CT\lambda_{k,branch[l']}$ for new individual $l^{'}$ formed by the old individual $branch[l^{'}]$
            with job $job[l^{'}]$. Note that the processing times of the artificial job is equal to the average
            processing times of all unscheduled jobs ($U^{l'}_{k+1}$);
            $SIT_{k+1,l'} = \frac{n-b}{n} \cdot \Big(SIT_{k,branch[l']} + IT_{job[l'],k,branch[l']} \cdot (n-k-2)\Big)$;
            $SCT_{k+1,l'} = SCT_{k,branch[l']} + CT_{job[l'],k,branch[l']} + CT\lambda_{k,branch[l']}$;
            $F_{k+1,l'} = a \cdot SCT_{k+1,l'} + SIT_{k+1,l'}$;
        end
    end
    //Final evaluation
    Evaluate the flowtime of the sequenced jobs of each individual and return the least one.
end
```

Figure 7.5: PCH

7.5 Computational experience

Computational experience for the FF heuristic

In order to compare the performance of the FF heuristic, the efficient heuristics described in Section 7.2 are implemented and their results on the benchmark $\mathcal{B}_1$ are collected. Note that all experiments carry out in this chapter are run on an Intel Core i7-3770 PC with 3.4 GHz and 16GB RAM. As mentioned before, [96] showed that heuristics for the $Fm|prmu|\sum C_j$ with insertion and pair-wise exchanges (i.e. all efficient heuristic but LR) can be implemented reducing around 30-50% computational times. Thus, in order to conduct a fair comparison, this acceleration has been implemented in our codification of each insertion method of all heuristics.

Comparing the CPU time required by each heuristic with those in the chapter by [137], we found that the former were larger by a factor of 3.38 times on average. This can be explained due to the different programming languages used to implement the heuristics, to the different ways of coding the routines and to the different computer employed for the implementation. However, since the stopping criterion of some heuristics in [137] was set to $0.01 \cdot n \cdot m$ seconds, applying the same criterion in our slower procedures would penalise the performance of these heuristics, as they now require more time per iteration and thus will perform less iterations. Therefore, in order to conduct a fair comparison, we change the stopping criterion to $0.0338 \cdot n \cdot m$ so to have a similar number of iterations to that of [137].

The overall results of the experiments are summarised in Table 7.6, where the average results of each heuristic over all 120 instances are shown. The effect of replacing LR by FF is specifically highlighted in Table 7.7, showing the efficiency of the proposed heuristics. For instance, the average computational time of $FF(1)$ is just $0.02s$ while the average computational time for $LR(1)$ is $0.76s$. Not only the complexity of the algorithm has been reduced from $n^3 \cdot m$ to $n^2 \cdot m$, but the $ARPD1$ of $FF(1)$ is also lower as compared to the $ARPD1$ of $LR(1)$.

Regarding the detailed results, those obtained by the heuristics Raj, $LR(1)$, RZ, $RZ-LW$, $LR-NEH(5)$, $LR-NEH(10)$, $LR-FPE$, $IC1$, $IC2$ and $IC3$ are shown in Table 7.8 while the CPU times are shown in Table 7.11.

Heuristic	$ARPD1$	$ARPT1$	$ACPU$
LR(1)	3.01	-0.98	0.76
LR(n/m)-FPE(n)	1.02	-0.47	33.07
IC1	0.64	-0.29	41.93
IC2	0.54	-0.08	55.33
IC3	0.53	1.26	330.92
LR-NEH(5)	1.52	-0.75	6.69
LR-NEH(10)	1.44	-0.52	13.37
Raj	4.86	-0.99	0.29
RZ	2.32	-0.90	2.97
RZ-LW	1.13	-0.42	32.69
PR1(5)	0.37	1.93	58.87
PR1(10)	0.26	4.60	67.38
PR1(15)	0.21	7.06	68.54
FF(1)	2.76	-1.00	0.02
FF(2)	2.34	-0.99	0.05
FF(n/10)	1.95	-0.98	0.99
FF(n/m)	2.05	-0.98	0.54
FF(n)	1.83	-0.69	10.12
FF(1)-FPE(1)	2.36	-0.99	0.15
FF(1)-FPE(n/10)	1.81	-0.94	2.89
FF(1)-FPE(n)	1.23	-0.64	21.39
FF(2)-FPE(1)	1.97	-0.97	0.17
FF(2)-FPE(n/10)	1.55	-0.94	3.05
FF(2)-FPE(n)	1.07	-0.63	19.78
FF(15)-FPE(1)	1.58	-0.88	0.44
FF(15)-FPE(n/10)	1.29	-0.86	2.90
FF(15)-FPE(n)	0.96	-0.62	16.89
FF(n/10)-FPE(1)	1.63	-0.96	1.12
FF(n/10)-FPE(n/10)	1.33	-0.92	3.51
FF(n/10)-FPE(n)	0.94	-0.63	18.79
FF(n/m)-FPE(1)	1.70	-0.96	0.64
FF(n/m)-FPE(n/10)	1.37	-0.92	3.11
FF(n/m)-FPE(n)	1.01	-0.64	18.05
FF(n)-FPE(1)	1.53	-0.65	10.27
FF(n)-FPE(n/10)	1.25	-0.62	12.68
FF(n)-FPE(n)	0.94	-0.35	28.00
FF-IC1	0.62	-0.48	25.33
FF-IC2	0.56	-0.26	36.47
FF-IC3	0.55	1.13	300.93
FF-NEH(5)	1.40	-0.86	3.18
FF-NEH(10)	1.34	-0.72	6.33
FF-PR1(5)	0.34	1.37	48.60
FF-PR1(10)	0.24	3.68	58.48
FF-PR1(15)	0.19	5.45	63.03

Table 7.6: Summary of results of heuristics

Heuristic	$ARPD1$	Avg. Time	$ARPT1$		Heuristic	$ARPD1$	Avg. Time	$ARPT1$
LR(1)	3.01	0.76	-0.98	$\rightarrow$	FF(1)	2.76	0.02	-1.00
LR(n/m)-FPE(n)	1.02	33.07	-0.47	$\rightarrow$	FF(n/m)-FPE(n)	1.01	18.05	-0.64
IC1	0.64	41.93	-0.29	$\rightarrow$	FF-IC1	0.62	25.33	-0.48
IC2	0.54	55.33	-0.08	$\rightarrow$	FF-IC2	0.56	36.47	-0.26
IC3	0.53	330.92	1.26	$\rightarrow$	FF-IC3	0.55	300.93	1.13
LR-NEH(5)	1.52	6.69	-0.75	$\rightarrow$	FF-NEH(5)	1.40	3.18	-0.86
LR-NEH(10)	1.44	13.37	-0.52	$\rightarrow$	FF-NEH(10)	1.34	6.33	-0.72
Raj	4.86	0.29	-0.99		—	—	—	—
RZ	2.32	2.97	-0.90		—	—	—	—
RZ-LW	1.13	32.69	-0.42		—	—	—	—
PR1(5)	0.37	58.87	1.93	$\rightarrow$	FF-PR1(5)	0.34	48.60	1.37
PR1(10)	0.26	67.38	4.60	$\rightarrow$	FF-PR1(10)	0.24	58.48	3.68
PR1(15)	0.21	68.54	7.06	$\rightarrow$	FF-PR1(15)	0.19	63.03	5.45

Table 7.7: Comparisons between composite heuristics which include LR and FF heuristics

Instance	LR(1)	LR(n/m)-FPE(n)	IC1	IC2	IC3	LR-NEH(5)	LR-NEH(10)	Raj	RZ	RZ-LW	PR1(5)	PR1(10)	PR1(15)	FF(1)	FF(2)	FF(n/10)	FF(n/m)	FF(n)
20 x 5	2.70	1.47	0.72	0.67	0.66	1.93	1.93	5.57	2.39	1.05	0.30	0.22	0.08	3.13	2.58	2.58	2.32	2.30
20 x 10	3.14	1.42	1.02	0.84	0.84	2.07	2.07	4.67	2.24	0.88	0.56	0.24	0.20	3.33	3.18	3.18	3.18	2.89
20 x 20	3.12	1.69	1.08	1.05	1.08	2.00	1.96	4.60	1.72	0.81	0.45	0.26	0.03	3.31	2.74	2.74	3.31	2.54
50 x 5	2.17	0.79	0.50	0.41	0.41	1.08	1.07	4.47	2.15	1.22	0.26	0.24	0.22	1.81	1.50	1.43	1.42	1.42
50 x 10	5.21	1.06	0.80	0.74	0.84	1.94	1.93	5.64	2.78	1.41	0.28	0.16	0.16	3.02	2.52	2.04	2.04	2.04
50 x 20	3.65	1.65	0.82	0.54	0.54	2.22	2.02	5.66	2.23	1.59	0.40	0.26	0.14	2.86	2.84	2.55	2.84	2.55
100 x 5	1.15	0.31	0.27	0.25	0.25	0.68	0.63	4.21	2.05	1.34	0.28	0.23	0.22	1.03	0.61	0.54	0.54	0.54
100 x 10	2.64	0.65	0.47	0.28	0.25	1.19	1.09	4.77	2.56	1.05	0.32	0.24	0.17	2.31	2.08	1.72	1.72	1.67
100 x 20	4.42	1.42	0.85	0.75	0.63	2.24	1.98	5.48	2.53	0.96	0.48	0.27	0.27	4.66	3.96	2.79	3.14	2.39
200 x 10	2.42	0.39	0.30	0.24	0.23	0.68	0.64	4.26	2.68	1.16	0.17	0.14	0.14	1.89	1.46	1.18	1.18	1.18
200 x 20	3.75	1.02	0.49	0.41	0.31	1.48	1.37	4.69	2.37	1.21	0.39	0.34	0.34	3.87	3.18	1.87	2.06	1.64
500 x 20	1.77	0.43	0.37	0.29	0.26	0.74	0.64	4.24	2.16	0.87	0.55	0.55	0.55	1.90	1.48	0.79	0.81	0.79
*ARPD*1	3.01	1.02	0.64	0.54	0.53	1.52	1.44	4.86	2.32	1.13	0.37	0.26	0.21	2.76	2.34	1.95	2.05	1.83

Table 7.8: RPD1 of heuristics I

	FF(x)-FPE(y)																	
	1-1	1-n/10	1-n	2-1	2 - n/10	2 - n	15-1	15-n/10	15-n	n/10-1	n/10-n/10	n/10-n	n/m-1	n/m-n/10	n/m-n	n-1	n-n/10	n-n
20 x 5	2.01	1.80	1.19	1.81	1.76	1.11	1.44	1.40	1.24	1.81	1.76	1.11	1.48	1.44	1.28	1.44	1.40	1.24
20 x 10	2.61	2.22	1.65	2.29	2.18	1.56	1.90	1.75	1.48	2.29	2.18	1.56	2.29	2.18	1.56	1.90	1.75	1.48
20 x 20	2.74	2.48	1.84	2.19	1.97	1.60	2.25	2.30	1.65	2.19	1.97	1.60	2.74	2.48	1.84	2.25	2.30	1.65
50 x 5	1.65	1.20	0.80	1.33	0.92	0.66	1.27	0.96	0.77	1.31	1.03	0.79	1.27	0.96	0.77	1.27	0.96	0.77
50 x 10	2.47	1.99	1.57	2.18	1.74	1.40	1.81	1.57	1.18	1.77	1.56	1.18	1.77	1.56	1.18	1.81	1.57	1.18
50 x 20	2.56	2.31	1.57	2.41	2.17	1.56	2.24	1.88	1.36	2.24	1.88	1.36	2.41	2.17	1.56	2.24	1.88	1.36
100 x 5	0.96	0.86	0.47	0.57	0.41	0.21	0.50	0.41	0.25	0.50	0.41	0.25	0.50	0.41	0.25	0.50	0.41	0.25
100 x 10	2.05	1.27	0.98	1.81	1.15	0.86	1.55	0.95	0.77	1.55	0.98	0.80	1.55	0.98	0.80	1.52	0.97	0.76
100 x 20	4.18	3.23	1.82	3.45	2.76	1.45	2.40	1.87	1.15	2.50	2.02	1.15	2.72	2.03	1.28	2.15	1.72	1.20
200 x 10	1.70	0.88	0.62	1.34	0.65	0.43	1.10	0.61	0.48	1.08	0.59	0.49	1.08	0.59	0.49	1.08	0.59	0.49
200 x 20	3.55	2.41	1.40	2.84	2.07	1.38	1.64	1.18	0.80	1.64	1.18	0.80	1.83	1.26	0.84	1.47	0.98	0.67
500 x 20	1.80	1.10	0.79	1.40	0.88	0.58	0.91	0.58	0.34	0.72	0.43	0.23	0.74	0.42	0.23	0.72	0.43	0.23
*ARPD*1	2.36	1.81	1.23	1.97	1.55	1.07	1.58	1.29	0.96	1.63	1.33	0.94	1.70	1.37	1.01	1.53	1.25	0.94

Table 7.9: RPD1 of heuristics II

	FF-ICH1	FF-ICH2	FF-ICH3	FF-NEH(5)	FF-NEH(10)	FF-PR1(5)	FF-PR1(10)	FF-PR1(15)
20 x 5	1.16	1.07	1.07	1.54	1.54	0.33	0.20	0.13
20 x 10	0.87	0.97	0.97	2.22	2.22	0.47	0.34	0.10
20 x 20	1.08	1.08	1.07	2.03	1.95	0.30	0.18	0.12
50 x 5	0.46	0.32	0.32	1.11	1.11	0.37	0.29	0.29
50 x 10	0.52	0.62	0.60	1.53	1.53	0.27	0.21	0.14
50 x 20	0.91	0.86	0.86	1.80	1.80	0.27	0.22	0.12
100 x 5	0.26	0.12	0.11	0.63	0.61	0.20	0.17	0.14
100 x 10	0.60	0.36	0.40	1.21	1.07	0.26	0.16	0.14
100 x 20	0.60	0.56	0.54	1.95	1.72	0.43	0.24	0.17
200 x 10	0.39	0.26	0.23	0.92	0.84	0.30	0.18	0.18
200 x 20	0.46	0.39	0.34	1.25	1.11	0.28	0.19	0.19
500 x 20	0.14	0.08	0.04	0.66	0.61	0.55	0.55	0.51
$ARPD1$	0.62	0.56	0.55	1.40	1.34	0.34	0.24	0.19

Table 7.10: RPD1 of heuristics III

Instance	LR(1)	LR(n/m)-FPE(n)	IC1	IC2	IC3	LR-NEH(5)	LR-NEH(10)	Raj	RZ	RZ-LW	PR1(5)	PR1(10)	PR1(15)	FF(1)	FF(2)	FF(n/10)	FF(n/m)	FF(n)
20 x 5	0.00	0.00	0.01	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.02	0.04	0.06	0.00	0.00	0.00	0.00	0.00
20 x 10	0.00	0.00	0.00	0.01	0.01	0.00	0.01	0.00	0.00	0.01	0.03	0.07	0.12	0.00	0.00	0.00	0.00	0.00
20 x 20	0.00	0.01	0.01	0.02	0.02	0.01	0.02	0.00	0.00	0.01	0.06	0.13	0.20	0.00	0.00	0.00	0.00	0.01
50 x 5	0.00	0.05	0.07	0.09	0.11	0.03	0.05	0.00	0.01	0.06	0.29	0.57	0.89	0.00	0.00	0.00	0.01	0.03
50 x 10	0.01	0.08	0.10	0.15	0.19	0.05	0.09	0.00	0.02	0.09	0.61	1.24	1.82	0.00	0.00	0.01	0.01	0.08
50 x 20	0.01	0.12	0.19	0.28	0.37	0.09	0.18	0.00	0.04	0.16	1.09	2.25	3.37	0.00	0.00	0.01	0.00	0.12
100 x 5	0.02	0.56	0.70	0.82	1.13	0.17	0.34	0.01	0.08	0.44	2.19	4.48	7.00	0.00	0.00	0.03	0.05	0.24
100 x 10	0.04	0.86	1.10	1.55	2.75	0.32	0.64	0.02	0.14	0.96	5.32	11.08	17.01	0.01	0.01	0.06	0.05	0.55
100 x 20	0.08	1.48	2.22	2.89	6.01	0.65	1.28	0.03	0.28	1.65	10.99	23.11	35.28	0.01	0.02	0.13	0.06	1.16
200 x 10	0.27	10.10	11.65	14.73	35.94	2.47	4.94	0.12	1.11	8.89	44.48	72.83	71.87	0.02	0.03	0.34	0.34	3.32
200 x 20	0.58	14.77	19.27	25.48	101.91	5.04	10.01	0.23	2.19	15.89	102.84	148.47	147.15	0.03	0.07	0.68	0.35	6.51
500 x 20	8.11	368.83	467.78	617.92	3822.54	71.43	142.83	3.07	31.76	364.16	538.48	544.33	537.68	0.22	0.44	10.69	5.62	109.42
Average	0.76	33.07	41.93	55.33	330.92	6.69	13.37	0.29	2.97	32.69	58.87	67.38	68.54	0.02	0.05	0.99	0.54	10.12
ARPT1	-0.98	-0.47	-0.29	-0.08	1.26	-0.75	-0.52	-0.99	-0.90	-0.42	1.93	4.60	7.06	-1.00	-0.99	-0.98	-0.98	-0.69

Table 7.11: Computational Times of heuristics I

	FF(x)-FPE(y)																	
	1-1	1-n/10	1-n	2-1	2 - n/10	2 - n	15-1	15-n/10	15-n	n/10-1	n/10-n/10	n/10-n	n/m-1	n/m-n/10	n/m-n	n-1	n-n/10	n-n
20 x 5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20 x 10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01
20 x 20	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.01	0.01
50 x 5	0.00	0.01	0.03	0.00	0.01	0.03	0.01	0.01	0.03	0.00	0.01	0.03	0.01	0.01	0.03	0.04	0.04	0.05
50 x 10	0.00	0.01	0.06	0.01	0.01	0.06	0.02	0.02	0.06	0.01	0.02	0.06	0.01	0.02	0.06	0.07	0.07	0.12
50 x 20	0.01	0.02	0.14	0.01	0.02	0.12	0.04	0.05	0.14	0.02	0.03	0.13	0.01	0.02	0.12	0.14	0.15	0.24
100 x 5	0.01	0.04	0.22	0.01	0.04	0.26	0.04	0.06	0.25	0.03	0.06	0.30	0.06	0.09	0.32	0.27	0.29	0.51
100 x 10	0.02	0.10	0.58	0.03	0.12	0.67	0.08	0.14	0.52	0.08	0.15	0.74	0.07	0.15	0.69	0.54	0.62	1.16
100 x 20	0.04	0.23	1.26	0.06	0.24	1.35	0.15	0.27	1.07	0.13	0.28	1.47	0.08	0.27	1.41	1.11	1.22	2.00
200 x 10	0.10	1.01	5.32	0.12	0.99	6.21	0.32	1.02	4.39	0.48	1.37	5.84	0.50	1.42	6.03	4.59	5.23	9.77
200 x 20	0.20	2.16	14.15	0.29	2.21	13.39	0.63	2.00	10.29	0.94	2.57	13.33	0.56	2.32	12.60	8.16	9.79	19.23
500 x 20	1.41	31.07	234.85	1.53	32.91	215.32	3.95	31.27	185.93	11.74	37.67	203.61	6.37	33.07	195.33	108.26	134.71	302.87
Average	0.15	2.89	21.39	0.17	3.05	19.78	0.44	2.90	16.89	1.12	3.51	18.79	0.64	3.11	18.05	10.27	12.68	28.00
ARPT1	-0.99	-0.94	-0.64	-0.97	-0.94	-0.63	-0.88	-0.86	-0.62	-0.96	-0.92	-0.63	-0.96	-0.92	-0.64	-0.65	-0.62	-0.35

Table 7.12: Computational Times of heuristics II

	FF-ICH1	FF-ICH2	FF-ICH3	FF-NEH(5)	FF-NEH(10)	FF-PR1(5)	FF-PR1(10)	FF-PR1(15)
20 x 5	0.00	0.00	0.01	0.00	0.00	0.02	0.05	0.06
20 x 10	0.01	0.01	0.01	0.00	0.01	0.03	0.06	0.10
20 x 20	0.01	0.01	0.01	0.00	0.01	0.05	0.10	0.16
50 x 5	0.05	0.06	0.08	0.01	0.03	0.23	0.44	0.66
50 x 10	0.09	0.12	0.16	0.03	0.05	0.44	0.89	1.32
50 x 20	0.18	0.22	0.27	0.05	0.10	0.87	1.71	2.64
100 x 5	0.32	0.52	0.89	0.09	0.18	1.84	3.74	5.36
100 x 10	0.77	1.30	2.71	0.18	0.35	3.75	7.66	11.75
100 x 20	1.95	2.71	5.18	0.32	0.65	8.29	16.28	24.29
200 x 10	7.93	11.00	49.87	1.21	2.40	36.42	67.75	72.55
200 x 20	18.85	24.30	100.12	2.34	4.58	72.73	144.41	146.06
500 x 20	273.86	397.42	3451.85	33.92	67.65	458.55	458.72	491.43
Average	25.33	36.47	300.93	3.18	6.33	48.60	58.48	63.03
ARPT1	-0.48	-0.26	1.13	-0.86	-0.72	1.37	3.68	5.45

Table 7.13: Computational Times of heuristics III

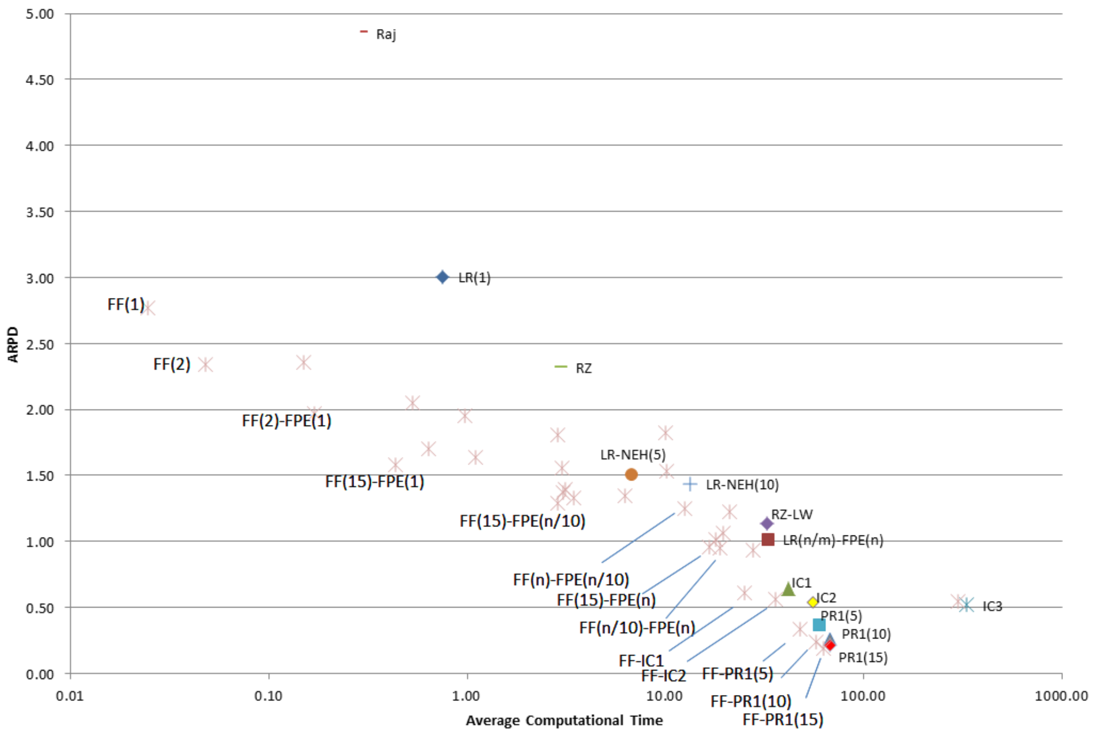

Figure 7.6: $ARPD1$ versus average computational times. X-axis (Average computational time) is shown in logarithmic scale. Noted that only the efficient heuristics of the new set of heuristics are named.

Different values of parameters x and y of the heuristic $FF(x)-FPE(y)$ have been analysed according to the literature (see e.g. [108, 137]). More specifically, $x \in \{1, 2, 15, n/10, n/m, n\}$ and $y \in \{1, n/10, n\}$ have been employed. Regarding the parameters of the heuristics $FF - NEH(x)$ and $PR1(a)$, the same values than in [137] (i.e. $x \in \{5, 10\}$ and $a \in \{5, 10, 15\}$) are chosen since the analysis of these parameters was already performed by these authors. Detailed $ARPD1$ results are shown in Tables 7.8, 7.9 and 7.10, while computational results are shown in Tables 7.11, 7.12 and 7.13.

Graphically, the new efficient set of heuristics using the average computational time is shown in Figure 7.6 while new efficient heuristics using $ARPT1$ as time reference are shown in Figure 7.7. For the former, the Pareto set is formed by the following heuristics: $FF(1)$, $FF(2)$, $FF(2)-FPE(1)$, $FF(15)-FPE(1)$, $FF(15)-FPE(n/10)$, $FF(15)-FPE(n)$, $FF(n/10)-FPE(n)$, $FF(n)-FPE(n)$, $FF-IC1$, $FF-IC2$, $FF-PR1(5)$, $FF-PR1(10)$ and $FF-PR1(15)$. For the latter, the efficient frontier is: $FF(1)$, $FF(2)$, $FF(n/10)$, $FF(n/m)$, $FF(2)-FPE(n/10)$, $FF(15)-FPE(n/10)$, $FF(n/10)-FPE(1)$, $FF(n/10)-FPE(n/10)$, $FF(n/10)-FPE(n)$, $FF(n/m)-FPE(n)$, $FF-IC1$, $FF-IC2$, $IC2$, $IC3$, $FF-PR1(5)$, $FF-PR1(10)$ and $FF-PR1(15)$. It represent a total of 17 heuristics in the new Pareto set. 15 of these heuristics correspond to the new set of heuristics presented in this chapter.

Seven paired samples t-test were carried out in order to compare the new set of efficient heuristic to

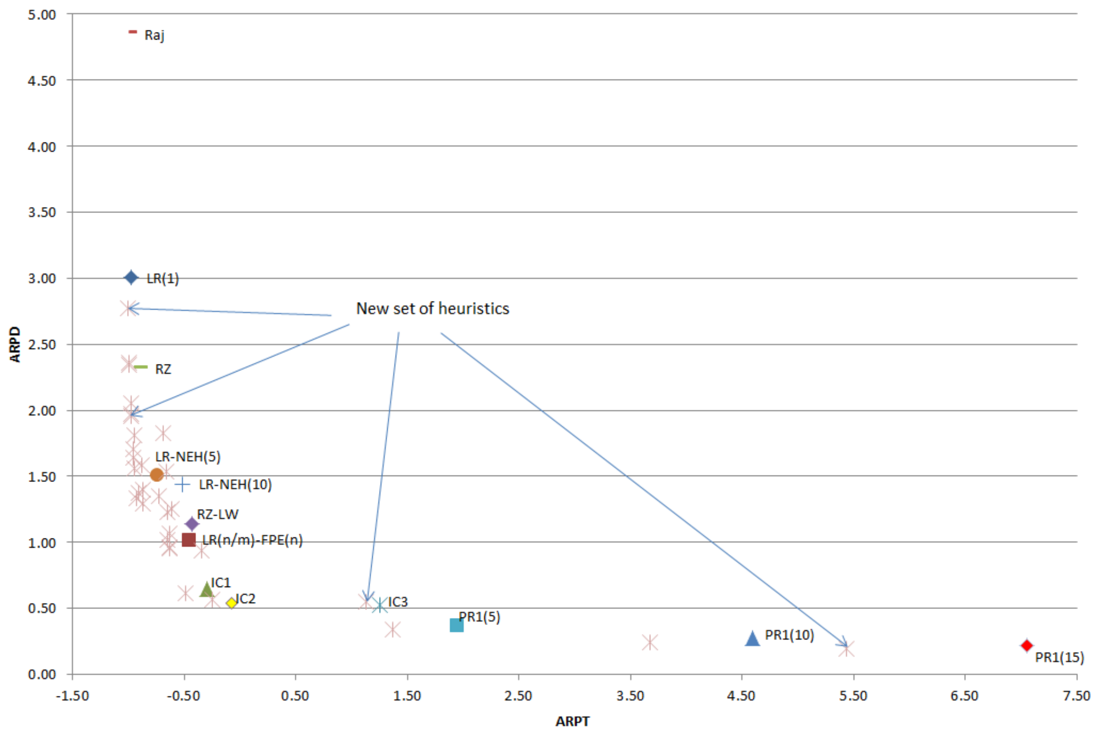

Figure 7.7: $ARPD1$ versus $ARPT1$.

Heuristics	Mean	SEM	IC - Lower	IC - Upper	t	Significance
Raj vs FF(1)	2.096	1.798	1.770	2.421	12.767	0.000
LR(1) vs FF(2)	0.668	1.507	0.396	0.941	4.857	0.000
LR(n/m)-FPE(n) vs FF-IC1	0.404	0.667	0.283	0.524	6.628	0.000
LR-NEH(5) vs FF(15)-FPE(n/10)	0.233	0.709	0.105	0.361	3.598	0.000
RZ vs FF(n/10)-FPE(n/10)	0.989	1.156	0.780	1.198	9.374	0.000
RZ-LW vs FF-IC1	0.510	0.830	0.360	0.660	6.738	0.000
LR-NEH(10) vs FF(n/10)-FPE(n)	0.500	0.711	0.371	0.628	7.699	0.000

Table 7.14: Paired samples t-test using Taillard's benchmark.

the old one. Comparisons were always performed between algorithms with higher ARPT1, i.e.: Raj vs $FF(1)$, $LR(1)$ vs $FF(2)$; $LR(n/m) - FPE(n)$ vs $FF - IC1$; $LR - NEH(5)$ vs $FF(15) - FPE(n/10)$; RZ vs $FF(n/10) - FPE(n/10)$; $RZ - LW$ vs $FF - IC1$ and $LR - NEH(10)$ vs $FF(n/10) - FPE(n)$. The results of the analysis are shown in Table 7.14. Statistically significant differences were found for each comparison being 0.000 the maximum p-value found in the analysis. The Least Significant Difference (LSD) intervals for each heuristics are shown in Figure 7.8.

Computational experience for the PCH heuristic

The proposed heuristic is compared with the current set of efficient heuristics formed by $FF(1)$, $FF(2)$, $FF(n/10)$, $FF(n/m)$, $FF(2) - FPE(n/10)$, $FF(15) - FPE(n/10)$, $FF(n/10) - FPE(1)$, $FF(n/10) -$

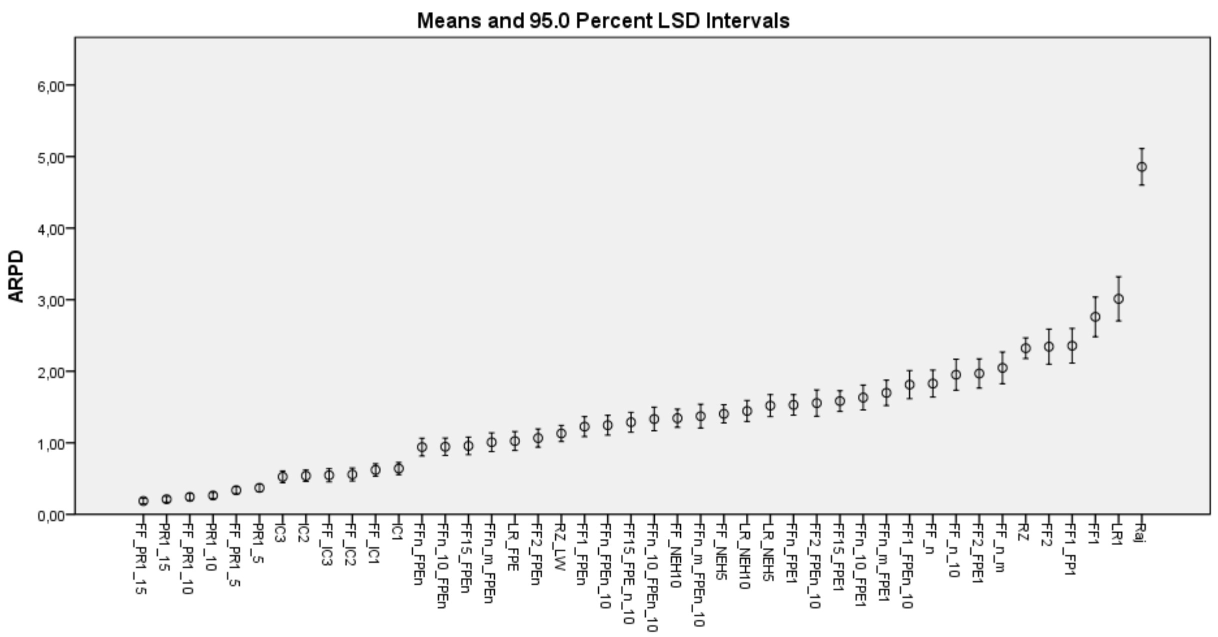

Figure 7.8: LSD intervals of the RPD1 for each analysed heuristics.

$FPE(n/10)$, $FF(n/10) - FPE(n)$, $FF(n/m) - FPE(n)$, $FF - IC1$, $FF - IC2$, $FF - IC3$, $IC2$, $IC3$, $FF - PR1(5)$, $FF - PR1(10)$ and $FF - PR1(15)$.

Experiments have been performed for the 120 instances of the benchmark $\mathcal{B}_1$. Additionally, parameter x of the proposed heuristic must be set. As shown in Section 7.4, x indicates the number of individuals in each iteration and therefore, it is directly proportional to the CPU time required by the heuristic. For $x > n$, additional indications in the first iteration of the algorithm would have to be provided (i.e. at least it should be indicated which is the first job of the last $x - n$ individuals after the first iteration), so here we restrict to $x \in \{1, n\}$. More specifically, we use the values of x also used in the literature for the LR heuristic, i.e. $x \in \{2, 5, 10, 15, n/10, n/m, n\}$ (see e.g. [108] and [137]). Note that $x = 1$ has been removed of the analysis since $PCH(1)$ is equal to $FF(1)$ (with a different combination of parameters), so it is already included in the computational evaluation.

The comparison of heuristics is performed in terms of quality of the solutions and computational effort. On the one hand, the former is commonly evaluated by means of the Relative Percentage Deviation $RPD1$. On the other hand, the most common indicator for the computational effort is the average CPU time, $ACPU$. However, in Section 7.2 detected that this indicator presents several problems when used to evaluate heuristics with different stopping criteria, and proposed the (Relative Percentage Time) RPT indicator to overcome these problems.

The $RPD1$ values obtained for each algorithm are shown in Tables 7.15 and 7.16. The last file indicates the average value, i.e. the $ARPD1$ for each algorithm. As it can be seen, the $ARPD1s$ of the actual set of efficient heuristics ranges from 3.84 to 1.22, where the best one (1.22) is found by FF-PR1(15). Regarding PCH, the worst $ARPD1$ is 2.51 while the best one is 0.19. In order to be able to perform a fair comparison between the heuristics, CPU times (in seconds) are summarised in Tables 7.17 and 7.18 (the last two files represent the average CPU time and the $ARPT1$ respectively). The average values are indicated in Table 7.19 and graphically shown in Figure 7.10 using the $ARPT$ as measure of the computational effort, as well as in Figure 7.9 using the average CPU time.

Using $ARPT1$ as a measure of the computational effort, the actual set of efficient heuristics is updated by including a complete new set of heuristics, all of them including PCH for different values of x. As it can be seen, our proposal PCH outperforms existing efficient heuristics, according to the following conclusions:

- $PCH(2)$ (with $ARPD1 = 2.51$) improves heuristics $FF(n/m)$, $FF(n/10)$ and $FF(n/10) - FPE(1)$ with $ARPD1s$ equal to 3.11, 3.02 and 2.70 respectively, while using less $ARPT1$.
- $PCH(n/m)$, $PCH(5)$ and $PCH(n/10)$ with $ARPD1$ 1.46, 1.35 and 1.21 respectively outperform $FF(2) - FPE(n/10)$ and $FF(n/10) - FPE(n/10)$ using less $ARPT1$.
- $PCH(10)$, with $ARPD1$ and $ARPT1$ equal to 0.88 and -0.87, clearly outperforms $FF(15) - FPE(n/10)$, which has an $ARPD1$ of 2.35 and an $ARPT1$ of -0.83.
- $PCH(15)$ ($ARPD1 = 0.64$) outperforms with less computational effort $FF(n/10) - FPE(n)$, $FF(n/m) - FPE(n)$, $FF - IC1$ and $FF - IC2$ which have a minimal $ARPD1$ of 1.61.
- The best heuristic, $PCH(n)$, with $ARPD1 = 0.19$ clearly outperforms heuristics $IC2$, $FF - IC3$, $IC3$, $FF - PR1(5)$, $FF - PR1(10)$ and $FF - PR1(15)$.

In order to establish the statistical significance of these results, Holm's procedure [73] is used where each hypothesis is analysed using a non-parametric Mann-Whitney test (see e.g. [138]). In Holm's procedure, the hypotheses are sorted in non-descending order of the p-values found in the Mann-Whitney test. The hypothesis i is rejected if its p-value is lower than $\alpha/(k-i+1)$ where k is the total number of hypotheses. The results of the Holm's procedure are shown in Table 7.20, where the fourth and sixth columns indicate if the hypothesis is rejected (denoted as R in such case) by Mann-Whitney and/or Holm's procedure. As can be seen hypothesis $PCH(2) = FF(n/10) - FPE(1)$ is the only one that cannot be rejected by Holm's procedure, but it has to be noted that the computational effort required by $FF(n/10) - FPE(1)$ is much higher to that by $PCH(2)$. In summary, it can be concluded that $PCH(n/10)$, $PCH(10)$, $PCH(15)$

and $PCH(n)$ are statistically efficient and that $PCH(2)$ is not inefficient. Note that $PCH(2)$ would be statistically efficient when considering the Pareto frontier using the average CPU time instead of the $ARPT1$.

A final series of experiments have been conducted to compare the $PCH(n)$ heuristic with an iterated local search (denoted as $MRSILS$) and an iterated greedy algorithm (denoted as IG_{RIS}). These two are among the best metaheuristics for the problem (see [34] and [138]). In order to analyse the impact of $PCH(n)$, we separately run both metaheuristics until the stopping criterion $60 \cdot n \cdot m/2$ milliseconds. For each instance, five runs are considered and the average flowtime values are recorded. Both metaheuristics have been again implemented under the same conditions and the comparison has been performed for all instances of the benchmark. Results in terms of $ARPD2$ and $ACPU$ are shown in Table 7.21. Note that, last column indicates the ratio between the time needed by metaheuristics and the $PCH(n)$ heuristic for each size of instance. The UB is the best known upper bound for the instance i taken from [136].

As it can be seen, both $ARPD2$ and $ACPU$ values of the metaheuristics are clearly improved by the proposed constructive heuristic. One the one hand, the best $ARPD2$ value of the metaheuristics is 0.76 while the $ARPD2$ value of the $PCH(n)$ heuristic is 0.40 (there are statistical differences between the algorithms when a non-parametric Mann-Whitney test is used as p-value equals to 0.004). Additionally, 35 new best upper bounds have been found in the instances (see Table 7.22). This fact clearly highlights the excellent performance of the proposed heuristic since e.g. only 12 upper bounds were updated when [136] ran the several metaheuristics until a stopping criterion of $400 \cdot m \cdot n$ milliseconds (i.e. an average CPU time of 731.7 seconds). On the other hand, big differences are found when analysing the average CPU time between the algorithms, which are 19.4 seconds for the $PCH(n)$ heuristic and 54.88 seconds for the metaheuristics. Although the differences in average CPU time are not so relevant, it is due to the use of an instance-size dependent indicator to compare algorithms with different stopping criteria (see Section 7.2 and Chapter 3.3 for a more detailed explanation). In fact, regarding the ratio of the CPU time between the metaheuristics and the proposed heuristic, the computational effort for the metaheuristics is 767.23 times bigger than for the proposed heuristic. This also serves to explain the good performance of the metaheuristics in the 60 smallest instances as compared with the proposed constructive heuristic since a huge computational effort is used for the former (e.g. approximately 3,500 times higher in instances Ta21-Ta-30). In contrast, the CPU time of the proposed heuristic is always less than 1 second, and its average CPU times for the first 90 instances is 0.17 seconds against 19.83 seconds required by the metaheuristics.

Instance	FF(1)	FF(2)	FF($n/10$)	FF(n/m)	FF(2)-FPE($n/10$)	FF(15)-FPE($n/10$)	FF($n/10$)-FPE(1)	FF($n/10$)-FPE($n/10$)	FF($n/10$)-FPE(n)	FF(n/m)-FPE(n)	FF-IC1
20 x 5	3.20	2.71	2.71	2.45	1.90	1.54	1.95	1.90	1.24	1.42	1.30
20 x 10	3.20	3.38	3.38	3.38	2.37	1.94	2.49	2.37	1.76	1.76	1.07
20 x 20	3.06	2.68	2.68	3.26	1.91	2.24	2.13	1.91	1.54	1.79	1.02
50 x 5	2.28	2.10	2.03	2.01	1.51	1.55	1.91	1.62	1.38	1.36	1.05
50 x 10	3.49	3.60	3.12	3.12	2.82	2.64	2.84	2.63	2.26	2.26	1.59
50 x 20	3.19	3.44	3.15	3.44	2.76	2.48	2.83	2.48	1.96	2.16	1.50
100 x 5	1.92	1.61	1.55	1.55	1.41	1.41	1.50	1.41	1.25	1.25	1.25
100 x 10	3.63	3.70	3.34	3.34	2.76	2.55	3.17	2.59	2.41	2.41	2.20
100 x 20	5.55	5.24	4.06	4.42	4.02	3.13	3.76	3.28	2.40	2.53	1.84
200 x 10	3.59	3.24	2.95	2.95	2.41	2.37	2.85	2.35	2.25	2.25	2.14
200 x 20	5.93	5.46	4.13	4.32	4.33	3.42	3.89	3.42	3.03	3.07	2.69
500 x 20	4.12	3.83	3.12	3.15	3.21	2.90	3.05	2.75	2.54	2.55	2.46
Average	3.84	3.42	3.02	3.11	2.62	2.35	2.70	2.39	2.00	2.07	1.68

Table 7.15: RPD1 of heuristics (I)

Instance	FF-IC2	FF-IC3	IC2	IC3	FF-PR1(5)	FF-PR1(10)	FF-PR1(15)	PCH(2)	PCH(5)	PCH(10)	PCH(15)	PCH($n/10$)	PCH(n/m)	PCH(n)
20 x 5	1.20	1.20	0.65	0.57	0.46	0.34	0.26	2.44	1.44	0.82	0.61	2.44	1.23	0.96
20 x 10	1.16	1.16	1.09	0.98	0.67	0.53	0.29	2.28	1.03	0.82	0.43	2.28	2.28	0.46
20 x 20	1.02	1.01	1.13	1.20	0.24	0.12	0.06	1.96	0.81	0.62	0.44	1.96	3.34	0.46
50 x 5	0.91	0.91	1.06	1.05	0.95	0.88	0.88	1.74	0.81	0.54	0.45	0.81	0.54	0.12
50 x 10	1.69	1.66	1.59	1.58	1.33	1.27	1.20	2.50	1.38	0.65	0.44	1.38	1.38	0.06
50 x 20	1.45	1.45	1.12	1.08	0.86	0.81	0.71	2.65	1.15	0.96	0.73	1.15	2.65	0.10
100 x 5	1.12	1.11	1.15	1.15	1.20	1.16	1.14	1.39	0.93	0.54	0.50	0.54	0.39	0.07
100 x 10	1.96	2.00	1.82	1.76	1.86	1.76	1.74	2.92	1.54	0.80	0.53	0.80	0.80	0.04
100 x 20	1.81	1.79	2.18	2.04	1.67	1.48	1.41	3.89	2.20	1.29	0.87	1.29	2.20	0.00
200 x 10	2.01	1.99	2.07	2.04	2.06	1.92	1.92	2.34	1.31	0.87	0.71	0.58	0.58	0.00
200 x 20	2.61	2.56	2.60	2.59	2.50	2.41	2.33	3.36	1.84	1.43	1.09	0.80	1.43	0.00
500 x 20	2.40	2.36	2.49	2.48	2.66	2.66	2.66	2.69	1.70	1.19	0.92	0.49	0.69	0.00
Average	1.61	1.60	1.58	1.54	1.37	1.28	1.22	2.51	1.35	0.88	0.64	1.21	1.46	0.19

Table 7.16: RPD1 of heuristics (II)

Instance	FF(1)	FF(2)	FF($n/10$)	FF(n/m)	FF(2)-FPE($n/10$)	FF(15)-FPE($n/10$)	FF($n/10$)-FPE(1)	FF($n/10$)-FPE($n/10$)	FF($n/10$)-FPE(n)	FF(n/m)-FPE(n)	FF-IC1
20 x 5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20 x 10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20 x 20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50 x 5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01
50 x 10	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.02	0.03
50 x 20	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.01	0.03	0.03	0.04
100 x 5	0.00	0.00	0.01	0.01	0.01	0.02	0.01	0.01	0.06	0.07	0.08
100 x 10	0.00	0.00	0.01	0.01	0.03	0.04	0.01	0.03	0.15	0.14	0.23
100 x 20	0.00	0.00	0.02	0.01	0.05	0.08	0.03	0.06	0.33	0.33	0.60
200 x 10	0.01	0.01	0.09	0.09	0.22	0.27	0.12	0.30	1.30	1.31	2.13
200 x 20	0.01	0.02	0.19	0.09	0.54	0.57	0.23	0.63	3.26	3.17	5.69
500 x 20	0.06	0.12	2.85	1.42	9.14	9.30	3.12	10.43	57.79	55.96	81.24
Average	0.01	0.01	0.26	0.14	0.83	0.86	0.29	0.96	5.25	5.09	7.51
ARPT1	-0.99	-0.99	-0.98	-0.97	-0.94	-0.83	-0.96	-0.92	-0.66	-0.66	-0.42

Table 7.17: Computational times of heuristics I

Instance	FF-IC2	FF-IC3	IC2	IC3	FF-PR1(5)	FF-PR1(10)	FF-PR1(15)	PCH(2)	PCH(5)	PCH(10)	PCH(15)	PCH($n/10$)	PCH(n/m)	PCH(n)
20 x 5	0.00	0.00	0.00	0.00	0.01	0.02	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20 x 10	0.00	0.00	0.00	0.00	0.01	0.03	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20 x 20	0.00	0.00	0.00	0.00	0.02	0.03	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50 x 5	0.02	0.03	0.03	0.04	0.06	0.13	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.03
50 x 10	0.04	0.05	0.06	0.07	0.13	0.27	0.39	0.00	0.00	0.01	0.01	0.00	0.00	0.03
50 x 20	0.07	0.08	0.14	0.15	0.25	0.53	0.82	0.00	0.00	0.01	0.02	0.00	0.00	0.06
100 x 5	0.16	0.26	0.30	0.35	0.52	1.03	1.56	0.00	0.01	0.01	0.02	0.01	0.03	0.31
100 x 10	0.36	0.79	0.45	0.94	1.07	2.20	3.35	0.00	0.01	0.02	0.03	0.02	0.02	0.40
100 x 20	0.81	1.64	0.83	1.91	2.56	5.05	7.58	0.01	0.02	0.04	0.06	0.04	0.02	0.68
200 x 10	3.17	14.54	4.39	15.01	10.21	19.59	28.92	0.02	0.04	0.08	0.13	0.18	0.19	7.25
200 x 20	7.66	32.59	8.43	23.84	22.74	45.59	68.98	0.03	0.07	0.15	0.23	0.33	0.15	8.57
500 x 20	120.86	1079.65	174.03	1090.81	378.95	390.46	391.29	0.19	0.44	0.95	1.62	7.32	2.88	215.44
Average	11.10	94.14	15.72	94.43	34.71	38.74	41.93	0.02	0.05	0.11	0.18	0.66	0.27	19.40
ARPT1	-0.15	1.04	0.14	1.26	1.82	4.57	7.12	-0.98	-0.95	-0.87	-0.80	-0.95	-0.95	0.02

Table 7.18: Computational times of heuristics II

Heuristic	$ARPD1$	$ARPT1$	Avg. Time
$FF(1)$	3.84	-0.99	0.01
$FF(2)$	3.42	-0.99	0.01
$FF(n/10)$	3.02	-0.98	0.26
$FF(n/m)$	3.11	-0.97	0.14
$FF(2)-FPE(n/10)$	2.62	-0.94	0.83
$FF(15)-FPE(n/10)$	2.35	-0.83	0.86
$FF(n/10)-FPE(1)$	2.70	-0.96	0.29
$FF(n/10)-FPE(n/10)$	2.39	-0.92	0.96
$FF(n/10)-FPE(n)$	2.00	-0.66	5.25
$FF(n/m)-FPE(n)$	2.07	-0.66	5.09
$FF-IC1$	1.68	-0.42	7.51
$FF-IC2$	1.61	-0.15	11.10
$FF-IC3$	1.60	1.04	94.14
$IC2$	1.58	0.14	15.72
$IC3$	1.54	1.26	94.43
$FF-PR1(5)$	1.37	1.82	34.71
$FF-PR1(10)$	1.28	4.57	38.74
$FF-PR1(15)$	1.22	7.12	41.93
$PCH(2)$	2.51	-0.98	0.02
$PCH(5)$	1.35	-0.95	0.05
$PCH(10)$	0.88	-0.87	0.11
$PCH(15)$	0.64	-0.80	0.18
$PCH(n/10)$	1.21	-0.95	0.66
$PCH(n/m)$	1.46	-0.95	0.27
$PCH(n)$	0.19	0.02	19.40

Table 7.19: Summary of results of the heuristics.

7.6 Conclusions

In this chapter, we have presented two new constructive heuristics denoted by $FF(x)$ and $PCH(x)$ for the permutation flowshop scheduling problem to minimise flowtime. On the one hand, the first heuristic constructs the final sequence adding jobs, one by one, at the end of the sequence based in the machine idle times and in the makespan of the inserted job. The complexity of the proposed algorithm is $x \cdot n^2 \cdot m$ being lower than the complexity of the heuristics in the actual Pareto set. Since most efficient heuristics use the algorithm LR in some of their phases, the latter can be replaced by the new algorithm FF in each of these heuristics. On the other hand, the population-based constructive heuristic, PCH, constructs sequences and, at the same time, combines them and selects the best x ones. Since the individuals are formed by partial sequences, a forecast index is introduced in order to be able to compare individuals with different un- and scheduled jobs.

The proposed heuristics have been compared on an extensive computational evaluation with the state-of-the-art algorithms. The results obtained by FF and PCH are much better than those obtained by other constructive heuristics in the literature for the problem (e.g. the $ARPD1$ and $ARPT1$ of the $PCH(n)$ heuristic is 0.19 and 0.02 respectively which are much less than those obtained by the most

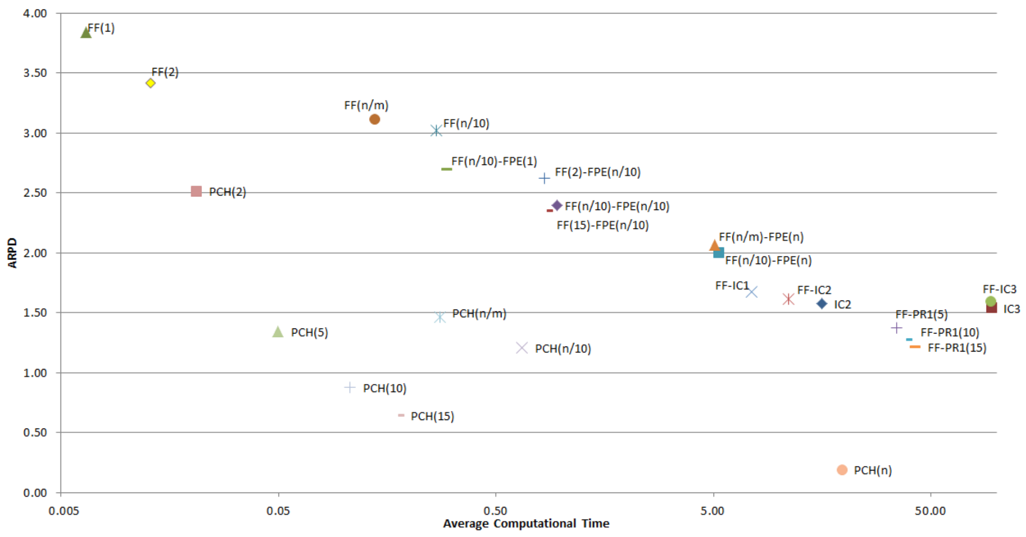

Figure 7.9: $ARPD1$ versus $ACPU$. Average computational time (X-axis) is shown in logarithmic scale.

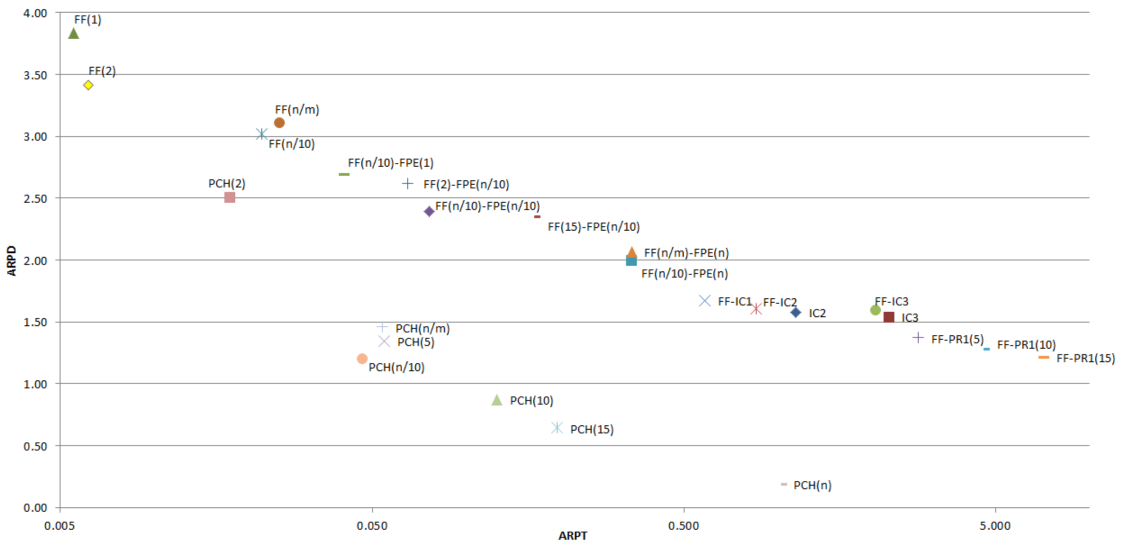

Figure 7.10: $ARPD1$ versus $ARPT1 + 1$. X-axis is shown in logarithmic scale

i	H_i	p-value	Mann-Whitney	$\alpha/(k-i+1)$	Holm's Procedure
1	$PCH(2)$=$FF(n/m)$	0.000	R	0.0031	R
2	$PCH(n/10)$=$FF(2)-FPE(n/10)$	0.000	R	0.0033	R
3	$PCH(n/10)$=$FF(n/10)-FPE(n/10)$	0.000	R	0.0036	R
4	$PCH(10)$=$FF(15)-FPE(n/10)$	0.000	R	0.0038	R
5	$PCH(15)$= $FF(n/10)-FPE(n)$	0.000	R	0.0042	R
6	$PCH(15)$=$FF(n/m)-FPE(n)$	0.000	R	0.0045	R
7	$PCH(15)$=$FF-IC1$	0.000	R	0.0050	R
8	$PCH(15)$=$FF-IC2$	0.000	R	0.0056	R
9	$PCH(n)$=$IC2$	0.000	R	0.0063	R
10	$PCH(n)$=$FF-IC3$	0.000	R	0.0071	R
11	$PCH(n)$=$IC3$	0.000	R	0.0083	R
12	$PCH(n)$=$FF-PR1(5)$	0.000	R	0.0100	R
13	$PCH(n)$=$FF-PR1(10)$	0.000	R	0.0125	R
14	$PCH(n)$=$FF-PR1(15)$	0.000	R	0.0167	R
15	$PCH(2)$=$FF(n/10)$	0.001	R	0.0250	R
16	$PCH(2)$=$FF(n/10)-FPE(1)$	0.163		0.0500	

Table 7.20: Holm's procedure.

	$ARPD2$			Avg. time		
Instance	$MRSILS$	IG_{RIS}	$PCH(n)$	$MRSILS, IG_{RIS}$	$PCH(n)$	$\frac{MRSILS,IG_{RIS}}{PCH(n)}$
20 x 5	0.01	0.05	1.25	3.00	0.00	1704.55
20 x 10	0.00	0.08	0.75	6.00	0.00	2500.00
20 x 20	0.00	0.01	0.75	12.00	0.00	3508.77
50 x 5	0.57	0.69	0.75	7.50	0.03	291.60
50 x 10	0.70	0.90	1.04	15.00	0.03	438.34
50 x 20	0.69	0.99	1.48	30.00	0.06	529.10
100 x 5	1.11	1.17	0.30	15.00	0.31	48.49
100 x 10	1.44	1.60	0.57	30.00	0.40	74.63
100 x 20	1.50	1.89	1.14	60.00	0.68	87.60
200 x 10	1.10	1.35	-0.61	60.00	7.25	8.28
200 x 20	1.24	1.46	-0.76	120.00	8.57	14.01
500 x 20	0.79	0.85	-1.87	300.00	215.44	1.39
Average	0.76	0.92	0.40	54.88	19.40	767.23

Table 7.21: $ARPD2$ and average CPU time, for each instance size, required by the $PCH(n)$ heuristic and the metaheuristics $MRSILS$ and IG_{RIS}.

Instance	Best Bound	Instance	Best Bound	Instance	Best Bound	Instance	Best Bound
TA1	14033	TA31	64802	TA61	**253232**	TA91	**1042494**
TA2	15151	TA32	68051	TA62	**242093**	TA92	**1028957**
TA3	13301	TA33	63162	TA63	237832	TA93	**1043467**
TA4	15447	TA34	68226	TA64	227738	TA94	**1029244**
TA5	13529	TA35	69351	TA65	240301	TA95	**1029384**
TA6	13123	TA36	66841	TA66	232342	TA96	**999241**
TA7	13548	TA37	66253	TA67	240366	TA97	**1042663**
TA8	13948	TA38	64332	TA68	230945	TA98	**1035981**
TA9	14295	TA39	62981	TA69	**247677**	TA99	**1015389**
TA10	12943	TA40	68770	TA70	242933	TA100	**1022277**
TA11	20911	TA41	87114	TA71	298385	TA101	**1223860**
TA12	22440	TA42	82820	TA72	**273826**	TA102	**1234081**
TA13	19833	TA43	79931	TA73	288114	TA103	**1259866**
TA14	18710	TA44	86446	TA74	301044	TA104	**1228060**
TA15	18641	TA45	86377	TA75	**284279**	TA105	**1219886**
TA16	19245	TA46	86587	TA76	269686	TA106	**1219432**
TA17	18363	TA47	88750	TA77	279463	TA107	**1234366**
TA18	20241	TA48	86727	TA78	290908	TA108	**1240627**
TA19	20330	TA49	85441	TA79	301970	TA109	**1220873**
TA20	21320	TA50	87998	TA80	291283	TA110	**1235462**
TA21	33623	TA51	125831	TA81	365463	TA111	**6558547**
TA22	31587	TA52	119247	TA82	372449	TA112	**6679507**
TA23	33920	TA53	116459	TA83	370027	TA113	**6624893**
TA24	31661	TA54	120261	TA84	372393	TA114	**6649855**
TA25	34557	TA55	118184	TA85	368915	TA115	**6590021**
TA26	32564	TA56	120586	TA86	370908	TA116	**6603691**
TA27	32922	TA57	122880	TA87	373408	TA117	**6576201**
TA28	32412	TA58	122489	TA88	384525	TA118	**6629393**
TA29	33600	TA59	121872	TA89	374423	TA119	**6589205**
TA30	32262	TA60	123954	TA90	379296	TA120	**6626342**

Table 7.22: New best bounds (in bold) found by the proposed algorithm.

efficient heuristic so far, $FF - PR1(15)$ with 1.22 and 7.13). When comparing $PCH(x)$ with the so-far most efficient heuristics in the literature, there are statistical differences for each new efficient heuristic with the only exception of $PCH(2)$. Thereby, the set of efficient heuristics for the problem has been reduced from 14 heuristics to seven heuristics of only two types of heuristics, the FF for parameters 1 and 2 which is efficient for the smallest CPU times, and the PCH with $x \in \{2, n/10, 10, 15, n\}$. The excellent performance of the proposed heuristic PCH is also shown by means of its comparison against two of the best metaheuristics for the problem. Our proposal statistically outperforms both metaheuristics ($ARPD2$ of $PCH(n)$ is 0.40 against 0.76 of the best metaheuristic) using much less computational effort for each instance of the benchmark. Additionally, the proposed heuristic found new best upper bounds for 35 of the 120 instances in Taillard's benchmark.

Additionally, certain issues have been identified in the evaluation of efficient heuristics in the literature. When analysing the trade-off between the quality of the solutions and the time required by the heuristics to obtain them, an dimensionless (and relative) variable ($ARPD1$) was used to represent the former while a dimensional and absolute variable (average computational time of the heuristic) was used to represent the latter. As discussed earlier in this chapter, some heuristics are deemed as efficient whereas they are not efficient for many problem sizes.

The intended contribution of the chapter can be summarised as follows. FF and PCH, two efficient heuristics, has been presented. These heuristics achieve better results in terms of both CPU time and $ARPD1$ than those obtained by the fastest efficient heuristic (with complexity $O(n^3 \cdot m)$) so far. By means of these heuristics, a new set of efficient heuristics for the problem has been identified, all of them formed by the proposed heuristics.

Chapter 8

PFSP to minimise total tardiness

8.1 Introduction

In this chapter, we will show that an analysis reveals the importance of adequately addressing the high number of ties appearing in the constructive phase of the NEHedd. In order to handle these ties in an efficient way, we propose several tie-breaking mechanisms for the problem and conduct an extensive computational experiment to test their performance (Objection SO8). The results show that one of these mechanisms (based on machine idle time) improves the original results obtained by NEHedd by roughly 25% while requiring the same CPU time. Another one (based on Taillard's acceleration for makespan) outperforms the NEHedd by 15% while employing less CPU time. Furthermore, when using the idle time-based version of the NEHedd as starting solution for the metaheuristic by [197] (which is the state-of-the-art metaheuristic for the problem), the metaheuristic improves its result for different stopping criteria.

The remainder of the section is organized as follows: in Section 8.2, eight tie-breaking mechanisms are proposed. An extensive comparison among them and with the original NEHedd procedure are performed in Section 8.3. Finally, conclusions are discussed in Section 8.4.

8.2 New tie-breaking mechanisms

The analysis carried out in Section 5.2 also serves to explain the excellent performance of the NEHedd procedure and to identify possible improvements. The NEHedd heuristic performs well in the three regions since it minimises also flowtime in Region 1, and, since it includes the EDD rule as a sorting order, it guarantees good performance in Region 3. However, it can be seen that its performance decreases for medium/high values v as compared to that for low/medium values of v. An explanation of this rather

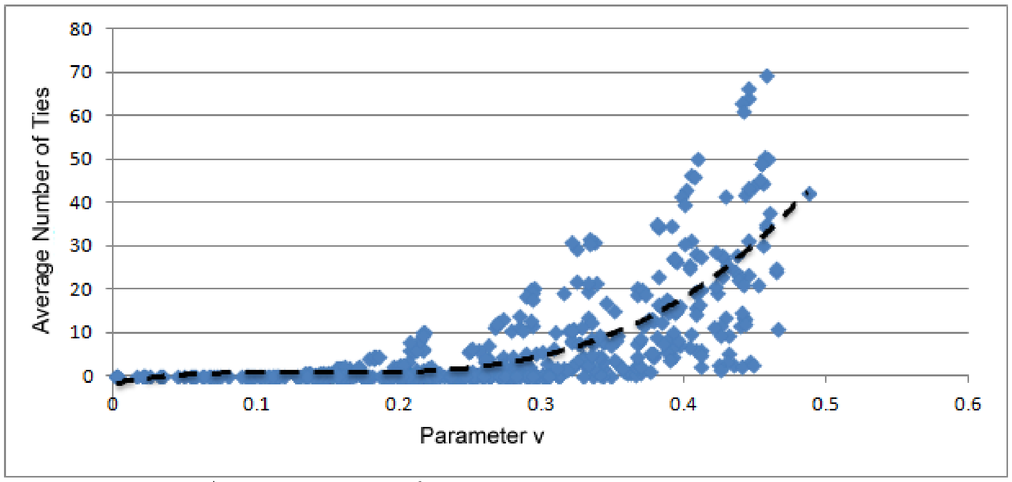

Figure 8.1: Average number of ties in each instance grouped by the parameter v.

surprising fact could lie in the high number of ties that would have to broken when, for each iteration of the NEHedd procedure, several partial sequences may have the same total tardiness. This situation could be rather common, as in the first iterations of the algorithm the total tardiness of the partial sequence is zero, thus leading to a high number of ties. In addition, since these ties appear in the first iterations, the mechanism chosen to solve them can greatly influence the final sequence obtained.

To confirm this fact, the number of ties on the well-known benchmark of instances proposed by [199], $\mathcal{B}_3$, has been studied. Results are shown in Figure 8.1 for different values of v, and yield an average of 10.1 ties per iteration, where 210 is the maximum number of ties found in an iteration. The number of ties increases with v and is close to zero for low values of v, which is consistent with the fact that the problem is similar to that of flowtime minimisation. The analysis also shows that, for some instances with a high value of v, an average of around a 40% of the positions where the job is to be inserted has the same total tardiness in each iteration, which represents a huge amount of ties.

In view of the results of the experiments, it can be concluded that the existence of a mechanism to break ties is extremely important for the NEHedd procedure in the $Fm|prmu|\sum T_j$ problem. However, a tie-breaking mechanism is not considered either in the NEHedd procedure, or in the original NEH algorithm for makespan minimisation. In the next section, we propose different tie-breaking mechanisms so the performance of NEHedd procedure can be improved in the most interesting region of the $Fm|prmu|\sum T_j$.

As mentioned in the previous section, no specific tie-breaking mechanism is mentioned in the original NEH heuristic for makespan minimisation. Indeed, it is cited [127] that "... Next, the job with the third highest total process time is selected and the three partial sequences are tested in which this job is placed at the beginning, middle and end of the partial sequence...", which seems to indicate that the first position where a tie is found is selected. In the following, we will denote this tie-breaking mechanism as FT (First-

```
Procedure NEHedd($TB_X$)
    $\alpha$ := Jobs ordered by non-decreasing due dates where $\alpha = \{\alpha_1, ..., \alpha_i, ..., \alpha_n\}$;
    $\pi := \{\alpha_1\}$;
    for $k = 2$ to $n$ do
        Test job $\alpha_k$ in any possible position of $\pi$.
        $\pi$ := permutation obtained by inserting $\alpha_k$ in the position of $\pi$ with less total tardiness breaking
        ties according to an specific mechanism;
    end
end
```

Figure 8.2: NEHedd with different tie-breaking mechanisms

Tie). Later, in the race for improving the NEH heuristic, [83] established the importance of breaking ties in the NEH heuristic and proposed a tie-breaking mechanism to improve the results obtained by the NEH heuristic. Since then, this aspect has been extensively analysed in the literature and several tie-breaking mechanisms have been proposed for the PFSP to minimise makespan (see [84], [35], [85], [162], [86], or the proposed in Chapter 6).

To the best of our knowledge, there are no tie-breaking mechanisms proposed for the NEHedd procedure, which adopts the first-tie mechanism as in the original NEH heuristic. However, it has to be noted that, since the EDD rule sorts the jobs according to non-decreasing due dates, in case of ties in the first iterations of NEHedd, the jobs would be finally ordered according to non-increasing due dates, which would probably lead to a worse final sequence than using a different mechanism.

In this section, several tie-breaking mechanisms are proposed to improve the traditional tie-breaking mechanism of the NEHedd procedure. The pseudo-code for the NEHedd algorithm including a generic tie-breaking mechanism is shown in Figure 8.2.

The proposed tie-breaking mechanisms involve using a secondary indicator related to the performance of the partial sequence. The goal is to pick, among those slots with the same tardiness, the slot yielding the best value of the secondary indicator for the unscheduled jobs. Thereby, total idle time ($IT1$ or $IT2$, see below), total flowtime (CT)), total earliness (ET) and makespan (MS) are chosen as potential secondary indicators. Note that, since these indicators have to be computed for every slot where the job is to be inserted in each iteration of the algorithm, they have to be carefully chosen so that the additional computational effort pays off.

More specifically, the tie-breaking mechanisms analysed in this chapter are:

- **First tie,** $NEHedd(TB_{FT})$. Original tie-breaking mechanism of the NEHedd algorithm proposed in [127] where, in case of ties, the first tie is chosen.

- **Last tie,** $NEHedd(TB_{LT})$. This tie-breaking mechanism simply consists in selecting the last tie as reference for the next iteration. This tie-breaking mechanism tries to solve the problem of TB_{FT}

where jobs are sorted according to the reverse EDD rule.

- **Total idle time,** $NEHedd(TB_{IT1})$ **and** $NEHedd(TB_{IT2})$. Denoting *front delay* of a machine as the time until it starts processing the first job, and *back delay* of a machine as the time between completing the processing of the last job and the completion of all jobs in any machine, machine idle time can be ambiguously defined by means of at least three different ways ([42]), i.e.: idle time including front delays and excluding back delays (denoted as IT1); idle time excluding front and back delays (denoted as IT2); and idle time considering front delays and back delays.

  If we adopt the first definition of idle time, then the idle time of machine i can be calculated as $IT1_i = C_{in} - \sum_{j=1}^{n} p_{ij}$. Consequently, the total idle time is $IT1 = \sum_{i=1}^{m} IT1_i$. Minimising $IT1$ looks for a more compacted schedule of the inserted jobs and it is equivalent to the minimisation of the sum of the completion times of each job in each machine. On the other hand, the second definition of idle time (excluding both delays) can be calculated as $IT2 = \sum_{j=2}^{n} \sum_{i=2}^{m} max\{C_{i-1,j} - C_{i,j-1}, 0\}$. The heuristics resulting from the use of these tie-breaking mechanisms in NEHedd are denoted as $NEHedd(TB_{IT1})$ and $NEHedd(TB_{IT2})$ respectively. Finally, note that the minimisation of the third definition of idle time is analogous to the minimisation of makespan and, therefore, it is considered below when discussing breaking ties according to the makespan.

- **Total completion time,** $NEHedd(TB_{CT})$. Total completion time can be defined as follows: $ct = \sum_{j=1}^{j} C_{m,j}$. As with idle time, this tie-breaking mechanism tries to balance the use of resources, and the resulting NEHedd heuristic is denoted by $NEHedd(TB_{CT})$.

- **Total earliness,** $NEHedd(TB_{ET})$. If a job finishes before its due date, its earliness indicates the time between the due date and the completion time of the job. Given several sequences with the same total tardiness, sequences with a high value of the total earliness indicate that, on average, the completion times of the jobs are far from their due dates. Thus, breaking ties by maximising earliness looks for sequences with a greater buffer against the due date of each job, which tries to improve the objective function when the following jobs are inserted in any position of the sequence. $NEHedd(TB_{ET})$ is denoted when earliness maximisation is used in the NEHedd algorithm to break ties.

- **Makespan,** $NEHedd(TB_{MS})$. Similarly to the first two tie-breaking mechanisms, the minimisation of the makespan tries to compress the sequence for the subsequent iterations. The NEHedd heuristic using the minimisation of the makespan as tie-breaking mechanism is denoted as $NEHedd(TB_{MS})$.

- **Makespan using Taillard's acceleration,** $NEHedd(TB_{MS-Taillard,IT1})$. As explained above,

```
Procedure $NEHedd(TB_{MS-Taillard,IT})$
    $\alpha :=$ Jobs ordered by non-decreasing due dates where $\alpha = \{\alpha_1, ..., \alpha_i, ..., \alpha_n\}$;
    $\pi := \{\alpha_1\}$;
    $flag :=$ true;
    for $k = 2$ to $n$ do
        if $flag$ then
            $\pi_1 := \pi$;
            Test job $\alpha_k$ in any possible position of $\pi_1$ (using Taillard's acceleration).
            $\pi_1 :=$ permutation obtained by inserting $\alpha_k$ in the position of $\pi_1$ with less makespan;
            $TT :=$ total tardiness of the sequence $\pi_1$;
            if $TT > 0$ then
                $flag :=$ false;
            else
                $\pi := \pi_1$;
            end
        end
        if $flag \neq true$ then
            Insert job $\alpha_k$ in the position of $\pi$ which minimises the total tardiness breaking ties according
            to the total idle time IT1 of the sequence.
        end
    end
end
```

Figure 8.3: $NEHedd(TB_{MS-Taillard,IT})$

Taillard's acceleration represents a huge reduction of the computation time of the NEH algorithm and it is one of the main reasons for its efficiency. However, it cannot be applied to total tardiness minimisation since the completion time of each job in the last machine is needed. To reduce the computation time of the NEHedd algorithm for the tardiness goal, this tie-breaking mechanism applies the NEH algorithm to minimise the makespan, using Taillard's acceleration as long as the tardiness of the (partial) sequence is zero, i.e. in the first iterations of the algorithm when applied. Once the (partial) tardiness is greater than zero, the proposed algorithm minimises the total tardiness (without Taillard's acceleration) breaking ties according to the total idle time, $IT1$. The pseudo code of this method is shown in Figure 8.3.

- **Random,** $NEHedd(TB_{rand})$. A random tie-breaking mechanism is proposed as a baseline for comparisons with the other mechanisms.

8.3 Computational experience

Each proposed tie-breaking mechanism has been compared under the same conditions (see Section 3.4). Algorithms were tested using the set of instances of the benchmark $\mathcal{B}_3$. The different tie-breaking mechanisms were compared by means of the RDI as an indicator of the quality of the solution.

Instance	TB_{FT}	TB_{LT}	TB_{rand}	TB_{IT1}	TB_{IT2}	TB_{CT}	TB_{MK}	TB_{ET}	$TB_{MS-Taillard,IT1}$
50x10	17.46	17.46	17.25	13.72	15.22	15.21	14.47	15.21	14.53
50x30	19.79	20.31	19.55	18.61	18.69	18.80	18.74	18.80	18.68
50x50	18.17	17.94	17.88	17.57	18.12	17.88	17.98	17.88	17.97
150x10	13.80	13.60	14.45	9.91	10.91	11.11	10.61	11.11	10.69
150x30	20.70	20.35	20.68	15.81	16.47	18.32	17.83	18.32	17.02
150x50	22.04	21.26	21.70	18.57	19.64	19.96	20.14	19.96	19.64
250x10	10.06	9.46	10.02	6.70	7.31	7.45	7.47	7.45	7.26
250x30	17.81	17.03	17.93	11.62	12.19	14.58	13.82	14.58	13.29
250x50	20.21	19.52	20.13	13.96	14.73	17.49	16.87	17.49	15.90
350x10	9.01	8.59	8.86	6.14	6.30	6.47	6.63	6.47	6.65
350x30	15.74	15.43	15.95	9.84	10.41	12.21	11.88	12.21	11.40
350x50	17.38	16.87	17.11	11.10	11.63	14.01	13.74	14.01	13.11
Average	16.85	16.48	16.79	12.80	13.47	14.46	14.18	14.46	13.84

Table 8.1: Relative deviation index (RDI) for the NEHedd heuristic using different tie-breaking mechanisms

Instance	TB_{FT}	TB_{LT}	TB_{rand}	TB_{IT1}	TB_{IT2}	TB_{CT}	TB_{MK}	TB_{ET}	$TB_{MS-Taillard,IT1}$
50x10	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
50x30	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
50x50	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08
150x10	0.43	0.42	0.43	0.42	0.42	0.44	0.42	0.43	0.40
150x30	1.29	1.29	1.29	1.30	1.30	1.30	1.29	1.30	1.27
150x50	2.16	2.16	2.16	2.18	2.18	2.17	2.15	2.16	2.15
250x10	1.92	1.90	1.91	1.89	1.89	1.96	1.90	1.95	1.77
250x30	5.86	5.84	5.86	5.85	5.84	5.94	5.85	5.92	5.67
250x50	9.92	9.91	9.95	9.96	9.97	9.99	9.89	9.94	9.78
350x10	5.15	5.14	5.15	5.08	5.07	5.26	5.12	5.24	4.73
350x30	15.86	15.85	15.89	15.80	15.76	16.10	15.85	16.03	15.23
350x50	26.99	26.98	27.06	27.01	27.01	27.26	27.05	27.18	26.35
Average	5.81	5.80	5.82	5.80	5.80	5.88	5.81	5.86	5.63

Table 8.2: $ACPU$ for the NEHedd heuristic with different tie-breaking mechanisms

The results of the heuristics are shown in Table 8.1 in terms of their values of RDI. The best overall results are found using $IT1$ as tie-breaking mechanism with an average RDI (denoted as $ARDI$) of 12.80, roughly about a 25% less than in the original FT. Note that each tie-breaking mechanism (also including the random mechanism) outperforms on average the original mechanism of the NEHedd algorithm, $NEHedd(TB_{FT})$, which has an $ARDI$ of 16.85. Although the difference between this original tie-breaking mechanism and the $NEHedd(TB_{LT})$ or the $NEHedd(TB_{rand})$ is less than 0.37, for the rest of tie-breaking mechanisms the $ARDI$ is at least a 2.39 lower than that obtained by $NEHedd(TB_{FT})$, which represents an increase in the quality of the solution without increasing the complexity of the algorithm. The CPU times of each algorithm for each combination of n and m are shown in Table 8.2. The differences between CPU times are negligible with the exception of the $NEHedd(TB_{MS-Taillard,IT1})$, which requires a bit less computational effort and has an $ARDI$ of 5.63, 3.22 lower than that ofFT.

Given that each tie-breaking mechanism is a version of the original NEHedd algorithm and that the

Comparison	N	Correlation	Sig.
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{IT1})$	540	0.707	0.000
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{IT2})$	540	0.760	0.000
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{MS-Taillard,IT1})$	540	0.816	0.000
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{CT})$	540	0.876	0.000
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{MK})$	540	0.826	0.000
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{ET})$	540	0.876	0.000
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{LT})$	540	0.949	0.000
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{rand})$	540	0.962	0.000

Table 8.3: Analysis of dependence of samples

Comparison	Wilcoxon signed-rank test		Sign test	
	Z	Sig.	Z	Sig.
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{IT1})$	-14.498	0.000	-12.363	0.000
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{IT2})$	-13.665	0.000	-11.446	0.000
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{MS-Taillard,IT1})$	-13.829	0.000	-12.020	0.000
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{CT})$	-13.616	0.000	-13.207	0.000
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{MK})$	-13.246	0.000	-11.810	0.000
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{ET})$	-13.616	0.000	-13.207	0.000
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{LT})$	-3.865	0.000	-2.904	0.004
$NEHedd(TB_{FT})$ vs $NEHedd(TB_{rand})$	-0.262	0.794	-0.349	0.727

Table 8.4: Wilcoxon signed-rank test and sign test

same test bed for all tie-breaking mechanisms is used, it is clear that the random variables (RDI) are related and the hypothesis of independence can be rejected (see Table 8.3 for each comparison). However, the hypothesis of normality is not fulfilled, so a paired samples t-test cannot be used. Two non-parametric statistical hypothesis tests (Wilcoxon signed-rank test and sign test) are carried out then with a confidence level of 99% to compare the statistical significance between the mean and the median of the samples, respectively. Results of the tests are shown in Table 8.4. For both tests, each tie-breaking mechanism statistically outperforms the original one with the exception of the random tie-breaking mechanism, for which no statistical difference was found (p-values of 0.794 and 0.727 for the Wilcoxon signed-rank test and sign test respectively). Regarding the significance of the different tie-breaking mechanisms, the highest p-value found was 0.004 when comparing TB_{LT} and TB_{FT}, which indicates the relatively bad performance of the original tie-breaking mechanism of the NEHedd procedure. The rest of the p-values are 0.000.

$ARDI$ is shown in Table 8.5 grouped by the values of the different parameters in the testbed: The first and second columns correspond to the value of each parameter in each row according to the values of T, R, n and m of the testbed. The third and fourth columns represent the average number of ties per iteration and the maximum number of ties in an iteration, respectively. The rest of the columns show the $ARDI$ values for each tie-breaking mechanism. $ARDI$ values for each tie-breaking mechanism are always lower than the $ARDI$ of TB_{FT} regardless of the value of the parameters, with the exception of TB_{rand} and

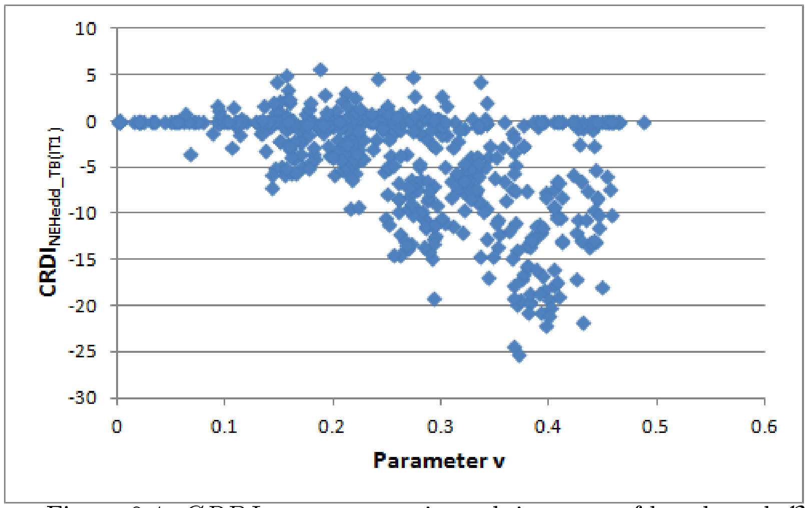

Figure 8.4: $CRDI_{NEH\_TB(IT1)}$ in each instance of benchmark $\mathcal{B}_3$.

TB_{LT}. Although $NEHedd(TB_{LT})$ statistically outperforms $NEHedd(TB_{FT})$ in the whole testbed, this does not happen when grouping by parameters. The minimum difference between the original tie-breaking mechanism and the rest is found for $T = 0.6$ and $R = 1.0$, which corresponds to tighter due dates with high variance. Obviously, the performance of the tie-breaking mechanism is related to the average number of ties. Thereby, note that the average and maximum number of ties decreases as m, T, or R increase, or as n decreases, reaching the maximum value of ties for the following combination of parameters: $T = 0.2$, $R = 0.2$, $n = 350$ and $m = 10$. Regarding the behaviour with respect to indicator v, $CRDI_{NEH\_TB(IT1)}$ the difference between the best tie-breaking mechanism TB_{IT1} as compared to the original one is shown in Figure 8.4. Most points are below zero in the y-axis, which highlights the improvement achieved by the heuristic when using $IT1$ as a tie-breaking mechanism, especially for $v > 0.15$ where the problem is far from being of the type $Fm|prmu|\sum C_j$.

Influence on iterative improvement algorithms

In this section, we evaluate the influence of the proposed NEH-based heuristics when they are incorporated as seed sequences in iterative improvement algorithms. For this comparison, we use the genetic algorithm, GAPR, proposed by [197]. Three types of genetic algorithms were proposed. Each one was shown to be statistically more efficient than other iterative improvement algorithms in the literature for three different stopping criteria. The GAPR algorithm uses a fast selection mechanism denoted as n-tournament as well as the path relinking as crossover mechanism. As initial solution, the algorithm uses 28 random sequences and two individuals provided by the original NEHedd algorithm and by the EDD despatching rule. To analyse the influence of the chosen tie-breaking mechanism, we substitute the NEHedd seed sequence by

Parameter		Ties		Tie-breaking mechanisms								
		Mean	Max.	FT	LT	rand	IT1	IT2	CT	MK	ET	MS-Taillard, IT1
T	0.2	22.5	210	14.57	13.91	14.47	7.18	8.27	10.45	9.71	10.45	8.90
T	0.4	6.5	144	18.14	17.92	18.08	14.15	14.95	15.84	15.67	15.84	15.45
T	0.6	1.1	81	17.83	17.62	17.83	17.06	17.18	17.09	17.17	17.09	17.19
R	0.2	17.1	210	20.86	19.72	20.49	13.14	14.27	15.65	15.02	15.65	14.96
R	0.6	8.3	146	16.83	16.70	16.84	12.88	13.60	15.00	14.84	15.00	14.07
R	1	4.7	112	12.86	12.98	12.95	12.41	12.51	12.69	12.77	12.69	12.57
n	50	0.5	23	18.25	18.28	17.85	16.81	17.32	17.18	17.20	17.18	17.15
n	150	3.8	81	18.85	18.40	18.94	14.76	15.67	16.47	16.19	16.47	15.78
n	250	8.9	153	16.03	15.33	16.02	10.76	11.41	13.17	12.72	13.17	12.15
n	350	14.9	210	14.11	13.70	14.04	9.13	9.55	10.98	10.84	10.98	10.47
m	10	16.0	210	12.37	12.02	12.32	9.13	9.80	9.90	9.82	9.90	9.75
m	30	8.7	182	18.25	18.06	18.38	13.82	14.26	15.79	15.43	15.79	14.97
m	50	5.4	162	19.30	18.78	19.04	15.16	15.86	17.20	17.05	17.20	16.52

Table 8.5: Average number of ties for iteration, maximum number of ties in an iteration and *ARDI* for each tie-breaking mechanism.

Stopping Criterion	*ARDI*-GAPR		Wilcoxon signed-rank test	Sign test
	$NEHedd(TB_{FT})$	$NEHedd(TB_{IT1})$	p-value	p-value
$t = 0.5$	14.66	11.01	0.000	0.000
$t = 1$	12.65	9.72	0.000	0.000
$t = 2$	10.61	8.42	0.000	0.000
$t = 5$	7.57	6.65	0.000	0.000
$t = 10$	6.25	5.63	0.000	0.000
$t = 20$	5.09	4.71	0.000	0.000

Table 8.6: ARDI, Wilcoxon signed-rank test and sign test for the GAPR algorithm when it is initialized with $NEHedd(TB_{IT1})$ and $NEHedd(TB_{FT})$

the best proposed NEHedd-based constructive heuristic, i.e. $NEHedd(TB_{IT1})$, and we compare them using the same benchmark as in the previous Section. Average computational results in terms of *ARDI* are shown in Table 8.6 and in Figure 8.5 for six different stopping criteria to observe the evolution of the performance for different CPU times, $t \cdot n \cdot (m/2)$ with $t \in [0.5, 1, 2, 5, 10, 20]$ expressed in milliseconds.

Obviously, one might expect that the influence of the initial solution on a well-designed metaheuristic such as the GAPR would decrease with the CPU time. Still, for the range of CPU times employed (which represents around 3 minutes of CPU times per instance for the biggest sizes), the results show that our proposal positively impacts on the quality of the solution. In fact, the positive contribution of the tie-breaking mechanism is found to be statistically significant for every stopping criteria, for both non-parametric statistical hypothesis tests (Wilcoxon signed-rank test and sign test). The highest found p-value was 0.000 (see Table 8.6).

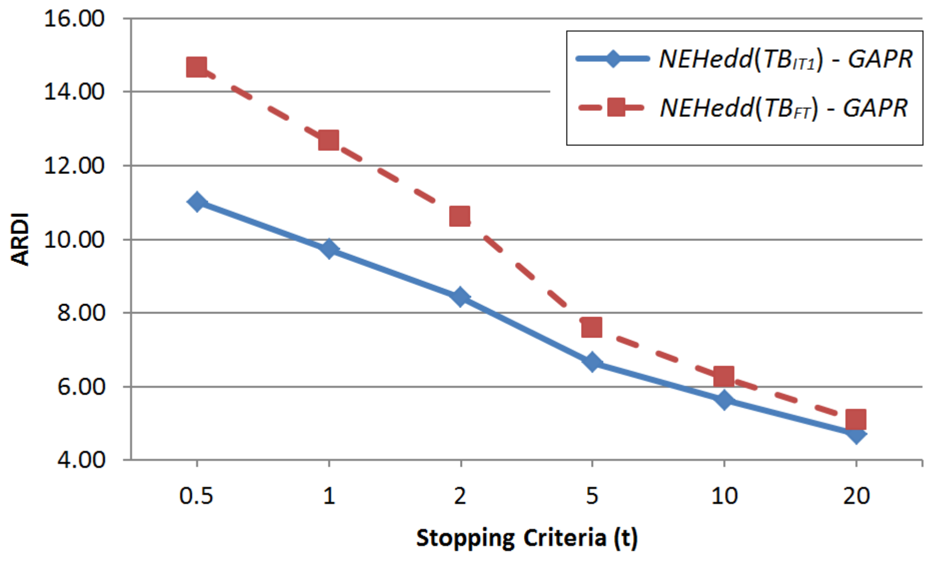

Figure 8.5: Evolution of the GAPR algorithm with different initial solutions for six different stopping criteria.

8.4 Conclusions

In this chapter, several tie-breaking mechanisms for the NEH heuristics have been proposed to solve the $Fm|prmu|\sum T_j$ problem. It is clear that, depending on the due dates, the decision problem to be solved is different. Extremely tight due dates induce to a $Fm|prmu|\sum C_j$ problem, whereas very loose due dates lead to a trivial problem. Thereby, the problem has been first analysed in detail, depicting the limits between the tardiness problem and other problems. As a conclusion, it was obtained that several testbeds generate instances for a problem more similar to $Fm|prmu|\sum C_j$. Additionally, it has been found that the number of ties in each iteration of the NEHedd heuristic is very high outside these limits (i.e. the most interesting setting regarding tardiness minimisation), and that the original tie-breaking mechanism of NEHedd would result in worse sequences as it orders the jobs in non-increasing due dates in the very likely case of ties in the first iterations. To address this problem and to enhance the performance of the NEHedd procedure, a set of eight tie-breaking mechanism have been proposed. These are tested against the original one in an extensive computational evaluation, and the results show that some of these mechanisms improve the performance of the NEHedd procedure by more than 25% while requiring similar computation time. Additionally, when embedding this mechanism as seed sequence in a state-of-the-art iterative improvement algorithm, the performance of the resulting algorithm significantly improves that of the original one.

Chapter 9

PFSP to minimise total earliness and tardiness

9.1 Introduction

In this chapter, four new efficient heuristics (one constructive heuristic and three composite heuristics) are proposed (see Objective SO9). These heuristics incorporate several properties and a speed up procedure in order to reduce and accelerate the search space of the heuristics. The subsequent computational experience shows that the proposed heuristics outperform the best-so-far heuristics for the problem, as well as adaptations of other state-of-the-art heuristics for related problems. Note that, for this problem, insertion of idle time is not allowed, which represents a common assumption in the literature due to its undesirable effects in certain production environments (see e.g. [81] and [179]).

The rest of the chapter is organised as follows. A speed up procedure for the insertion phase of the heuristics as well as a complete comparison among the implemented heuristics is performed in Section 9.3. Additionally, the influence of using the heuristics as seed sequences in the best so-far metaheuristic (ILS) is discussed. Finally, in Section 11.5, conclusions are presented.

9.2 Proposed algorithms

Following the recommendations in Section 5.2 regarding very fast heuristics and complete local search methods, four heuristics are proposed for the $Fm|prmu|\sum E_j + \sum T_j$ problem:

- an adaptive constructive heuristic (see Section 9.2), denoted as ACH1,

- a composite heuristic (see Section 9.2), denoted as ACH2, composed by ACH1 plus a bounded local search procedure labelled BLS,
- a composite heuristic (see Section 9.2), denoted as ACH3, formed by ACH1 plus an iterative bounded relative local search method, iBRLS,
- a composite heuristic (see Section 9.2), denoted as ACH4, formed by ACH1 and an iterative local search method, iLS.

Additionally, a speed-up procedure is described in Section 9.2 to accelerate the insertion phases of all implemented algorithms.

Proposed constructive heuristic

ACH1 tries to find a good solution using very short computational times so it can embedded in more sophisticated constructive and composite heuristics such as the ones proposed in the next subsections. The procedure of this heuristic is relatively simple: Beginning with a partial sequence with a single job, the procedure constructs a final sequence appending one by one jobs at the end of the partial sequence according to an index $\xi_{u_{jk}}(\Pi)$. Let us denote by $\Pi^k := (\pi_1, ..., \pi_k)$ the partial sequence in iteration k and by $\mathcal{U}_k$ the set of unsequenced jobs of that sequence (u_{jk} the jth unsequenced job with $j \in [1, n-k]$). Additionally, let NT_k be the number of tardy jobs in iteration k according to sequence Π^k. The algorithm chooses the job from $\mathcal{U}_k$ with the lowest value of $\xi_{u_{jk}}(\Pi^k)$ and places it at the end of sequence Π^k, i.e. in position $k+1$, forming the sequence Π^{k+1} of the next iteration. This procedure has been shown to be very efficient for other decision problems, being the appropriate choice of the index the critical issue for the efficiency of the algorithm (see e.g. Chapter 7). This difficulty increases in our case due to the strong dependence of the best solutions on the due dates of the jobs. The index must be adapted to solve different problems depending on the due dates (loose due dates, tight ones, or neither of them). In Section 5.2, three different situations have been identified: tight due dates ($Fm|prmu|\sum C_j$ decision problem), loose due dates ($Fm|prmu|-\sum C_j$ decision problem) and normal due dates ($Fm|prmu|\sum E_j + \sum T_j$). Therefore, at each iteration, the algorithm would check whether the sequence is within one of these cases:

- Case 1: Tight due dates (i.e., the problem is similar to the $Fm|prmu|\sum C_j$). There are hundred of heuristics solving the $Fm|prmu|\sum C_j$ in the literature. Particularly, in Section 7.3 we have designed an efficient constructive heuristic following a similar procedure of insertion in last position of the partial sequence. There, jobs are chosen according to the $\xi^1_{u_{jk}}(\Pi^k)$ index, Equation (9.1), which considers the minimization of the completion time and the weighted idle time of the candidates jobs (i.e. u_{ik} with $j \in [1, n-k]$) to be inserted (see Section 7.3 and Equations 7.2 and 7.1 for more detailed explainations):

$$\xi_{u_{jk}}(\Pi^k) = \xi^1_{u_{jk}}(\Pi^k) = \frac{(n-k-2)}{4} \cdot IT_{u_{jk}}(\Pi^k) + C_{m,u_{jk}}(\Pi^k) \tag{9.1}$$

where $IT_j(\Pi^k)$ are:

$$IT_{u_{jk}}(\Pi^k) = \sum_{i=2}^{m} \frac{m \cdot max\{C_{i-1,u_{jk}}(\Pi^k) - C_{i,\pi_k}(\Pi^k), 0\}}{i - 1 + k \cdot (m-i+1)/(n-2)} \tag{9.2}$$

In this chapter, this index is directly incorporated into the ACH1 heuristic when due dates of jobs are tight, assuming that this is the case if, in iteration k, the fraction of tardy jobs in the partial sequence is equal to or greater than a. More specifically, the index is used if $a \cdot 100\%$ $(NT_k/(n-k) \geq a)$ and there are at least four tardy jobs. Note that a is a parameter of the algorithm which is introduced in order to determine when the algorithm is adapted to solve $Fm|prmu|\sum C_j$. The suitable values for a are discussed later.

- Case 2: Loose due dates (the problem is similar to $Fm|prmu| - \sum C_j$). We divided this case in two cases depending on how loose the due dates are. The idea behind these two subcases is to separate the case where all due dates are extremely high (the earliness can be omitted and the problem is similar to the $Fm|prmu| - \sum C_j$) and where only some of the due dates are extremely high (earliness should be also considered). To the best of our knowledge, there are no algorithms for the PFSP to maximise total flowtime as it is not a common objective to be followed by companies. Therefore, we consider the inverse index for the first subcase, $\xi_{u_{jk}}(\Pi^k) = -\xi^1_{u_{jk}}(\Pi^k)$. The conditions to apply this index are fulfilled when there are at least four candidates $(n-k > 3)$, all candidates in iteration k are in earliness (i.e. $C_{m,u_{jk}}(\Pi^k) < d_{u_{jk}}$ $\forall j$), and $NE_k = n-k$ where NE_k is the number of jobs whose earliness is lower than $(n-k) \cdot c$. On the other hand, when due dates are not so loose (this fact is measured by the condition $b \cdot (n-k) \leq NE_k < n-k$) we consider the index ξ^2 which adds the earliness of job $E_{u_{jk}}(\Pi^k)$ to the index $-\xi^1$:

$$\xi_{u_{jk}}(\Pi^k) = \xi^2_{u_{jk}}(\Pi^k) = -\frac{(n-k-2)}{4} \cdot IT_{u_{jk}}(\Pi^k) - C_{m,u_{jk}}(\Pi^k) + E_{u_{jk}}(\Pi^k) \tag{9.3}$$

  Note that b and c are parameters of the proposed algorithm.

- Case 3: Intermediate due dates, i.e. the instance is a pure $Fm|prmu|\sum E_j + \sum T_j$ problem. When a job is inserted at the end of the sequence, the algorithm should focus in the minimization of total

Case	Condition	$\xi_{u_{jk}}(\Pi^k)$ index
Tight due dates	$NT_k/(n-k) \geq a$ $NT_k > 3$	$\xi^1_{u_{jk}}(\Pi^k)$
Loose due dates (Subcase 1)	$C_{m,u_{jk}}(\Pi^k) < d_{u_{jk}}, \forall j$ $n-k > 3$ $NE_k = n-k$	$-\xi^1_{u_{jk}}(\Pi^k)$
Loose due dates (Subcase 2)	$C_{m,u_{jk}}(\Pi^k) < d_{u_{jk}}\ , \forall j$ $n-k > 3$ $b \cdot (n-k) \leq NE_k < n-k$	$\xi^2_{u_{jk}}(\Pi^k)$
Intermediate due dates	Otherwise	$\xi^3_{u_{jk}}(\Pi^k)$

Table 9.1: Summary of cases in the ACH

earliness and tardiness. Then, we try to place each candidate at the end of the partial sequence and, in order to select the job to be inserted, we focus on the minimisation of earliness by using the index in Equation (9.4):

$$\xi_{u_{jk}}(\Pi^k) = \xi^3_{u_{jk}}(\Pi^k) = E_{u_{jk}}(\Pi^k) \tag{9.4}$$

Note that the minimisation of earliness implicitly takes into account also tardy jobs as their earliness is equal to zero and then, they would be the first to be chosen.

The different values adopted by $\xi_{u_{jk}}(\Pi^k)$ are summarized in Table 9.1 together with the corresponding conditions. Once the index has been identified, we choose the job with the lowest value and place it at the end of the current partial sequence. Ties are broken according to the weighted idle time of the candidate jobs ($IT_{u_{jk}}(\Pi^k)$) for all cases.

Finally, note that, when inserting a job at the end of the sequence, the completion time of previous last job remains the same and hence, only the completion times of the inserted job on each machine has to be computed, which can be easily done in $O(m)$. Analogously, earliness time and/or weighted idle time of each candidate job can be computed with complexity m. Thereby, the proposed ACH1 constructive heuristic has the same number of complexity than the NEH. However, the heuristic heavily decreases the CPU time due to the complexity of each evaluation, which is simply m. Therefore, the complexity of the proposed heuristic is $\frac{(n+1)\cdot n}{2} \cdot m \sim O(n^2 \cdot m)$.

The pseudo code of the ACH1 heuristic is shown in Figure 9.1.

Proposed composite heuristics

Once a sequence has been obtained by the fast ACH1 constructive heuristic, it can be improved by three different insertion-based local search methods, leading to composite heuristics. The first two local search methods try to reduce the computational effort required in each iteration by avoiding the insertion of a

```
Procedure ACH1
    Determination of each $IT_{j,0}$, $CT_{j,0}$ and $E_{j,0}$
    $\pi_1$ := Job with least value of $E_{j,0}$ breaking ties in favor of the job with the lowest $IT_{j,0}$;
    $\Pi^1 = (\pi_1)$
    for $k = 1$ to $n - 1$ do
        Determination of each $IT_{u_{jk}}(\Pi^k)$, $C_{m,u_{jk}}(\Pi^k)$, $NT_k$, $NE_k$, and $E_{u_{jk}}(\Pi^k)$, $\forall j \in [1, n-k]$;
        if $NT_k/(n-k) \geq a$ & $NT_k > 3$ then
            for $j = 1$ to $k - n$ do
                $\xi_{u_{jk}}(\Pi^k) = \xi^1_{u_{jk}}(\Pi^k) = \frac{(n-k-2)}{4} \cdot IT_{u_{jk}}(\Pi^k) + C_{m,u_{jk}}(\Pi^k)$
            end
        else if AllEarliness & $n - k > 3$ & $NE_k = n - k$ then
            for $j = 1$ to $k - n$ do
                $\xi_{u_{jk}}(\Pi^k) = -\xi^1_{u_{jk}}(\Pi^k) = -\frac{(n-k-2)}{4} \cdot IT_{u_{jk}}(\Pi^k) - C_{m,u_{jk}}(\Pi^k)$
            end
        end
        else if AllEarliness & $n - k > 3$ & $b \cdot (n-k) \leq NE_k < n - k$ then
            for $j = 1$ to $k - n$ do
                $\xi_{u_{jk}}(\Pi^k) = \xi^2_{u_{jk}}(\Pi^k) = -\frac{(n-k-2)}{4} \cdot IT_{u_{jk}}(\Pi^k) - C_{m,u_{jk}}(\Pi^k) + E_{u_{jk}}(\Pi^k)$
            end
        end
        else
            for $j = 1$ to $k - n$ do
                $\xi_{u_{jk}}(\Pi^k) = \xi^3_{u_{jk}}(\Pi^k) = E_{u_{jk}}(\Pi^k)$
            end
        end
        $\alpha$:= Job with the lowest value of $\xi_{u_{jk}}(\Pi^k)$ in iteration $k$, breaking ties in favor of the job with the lowest $IT_{u_{jk}}(\Pi^k)$.
        $\Pi^{k+1}$:= Permutation obtained by inserting job $\alpha$ at the end of sequence $\Pi^k$.
    end
end
```

Figure 9.1: ACH1

```
Procedure ACH2()
    $(\Pi, \sum E_j + \sum T_j) = ACH1()$;
    $(\Pi, \sum E_j + \sum T_j) = BLS(\Pi, \sum E_j + \sum T_j)$;
end
```

Figure 9.2: ACH2

```
Procedure BLS($\Pi, OF$)
    $OF_b = OF$
    for $j = 1$ to $n$ do
        $\Pi^0$ := remove job $\pi_j$ from $\Pi$;
        Calculate $P1$ and $P2$;
        Test job $\pi_j$ between the positions $P1$ and $P2$ of $\Pi^0$;
        $\Pi$ := permutation obtained by inserting $\pi_j$ in the position $j \in [P1, P2]$ of $\Pi^0$ with less total
        earliness and tardiness, $OF^{'}$;
        if $OF^{'} < OF_b$ then
            $OF_b = OF^{'}$;
            $\Pi^b := \Pi$;
        end
    end
    return $\Pi^b$ and $OF_b$;
end
```

Figure 9.3: Bounded Local Search, BLS

job far from its optimal position in order to accelerate the intensification of the procedures. The idea is to bound the positions where the jobs are inserted between $P1$ and $P2$, which are defined in Equations (9.5) and (9.6).

$$P1 = \min\{P_{AS}, P_{edd}\} \tag{9.5}$$

$$P2 = \max\{P_{AS}, P_{edd}\} \tag{9.6}$$

where P_{edd} is the position in the EDD rule of π_j, and P_{AS} is the position that should have that π_j in the actual sequence in order to get a minimum value of E_{π_j} and T_{π_j}.

Recently, bounded local search methods have been successfully applied to scheduling problems. The key of this success comes from the reduction of the search space in each iteration of the local search methods in order to avoid sequences that are far from the optimum, therefore decreasing the computational effort of the algorithms. More specifically, the proposed composite heuristics are:

- ACH2. It performs a bounded local search (denoted BLS) after the ACH1 heuristic. The BLS tries to insert each π_j in each position between $P1$ and $P2$. Pseudo code of both ACH2 and BLS methods are shown in Figures 9.2 and 9.3 respectively.

```
Procedure ACH3()
    $(\Pi, \sum E_j + \sum T_j) = ACH1();$
    $(\Pi, \sum E_j + \sum T_j) = iBRLS(\Pi, \sum E_j + \sum T_j);$
end
```

Figure 9.4: ACH3

```
Procedure iBRLS($\Pi, OF$)
    $OF_b = OF$
    $h = 1;$
    $i = 1;$
    $\Pi^b := \Pi;$
    while $i <= n$ do
        $j := h \bmod n;$
        $\Pi^0 :=$ remove job $\pi_j$ from $\Pi$;
        Calculate $P1$ and $P2$;
        Test job $\pi_j$ between the positions $P1$ and $P2$ of $\Pi^0$;
        $\Pi :=$ permutation obtained by inserting $\pi_j$ in the position $j \in [P1, P2]$ of $\Pi^0$ with less total
        earliness and tardiness, $OF^{'}$;
        if $OF^{'} < OF_b$ then
            $OF_b = OF^{'};$
            $i = 1;$
            $\Pi^b := \Pi;$
        else
            $i++;$
        end
        $h++;$
    end
    return $\Pi^b$ and $OF_b$;
end
```

Figure 9.5: Iterative Bounded Relative Local Search, iBRLS

- ACH3. It performs an iterative bounded local search method, denoted as iBRLS after the ACH1 heuristic. Similarly to the BLS method, the iBRLS tries to iteratively insert each job π_j between $P1$ and $P2$ until there is no improvement after trying n consecutive jobs. Pseudo code of ACH3 and iBRLS are detailed in Figure 9.4 and 9.5 respectively.

- ACH4. This heuristic carries out a iterative local search method (iLS) after the ACH1 heuristic. This local search method simply tries to place each job π_j in the rest of positions of the current sequence and has been extensively used in the literature (see e.g. [174], [97] and [139]). The procedure is repeated until there are no more improvements. Pseudo codes for ACH4 and iLS methods are shown in Figures 9.6 and 9.7.

```
Procedure ACH4()
    $(\Pi, \sum E_j + \sum T_j) = ACH1();$
    $(\Pi, \sum E_j + \sum T_j) = iLS(\Pi, \sum E_j + \sum T_j);$
end
```

Figure 9.6: ACH4

```
Procedure iLS($\Pi, OF$)
    $OF_b = OF$
    $flag :=$ false;
    while $flag = false$ do
        $flag :=$ false;
        for $j = 1$ to $n$ do
            $\Pi^0 :=$ remove job $\pi_j$ from $\Pi$;
            Test job $\pi_j$ in each position of $\Pi^0$;
            $\Pi :=$ permutation obtained by inserting $\pi_j$ in the position $j$ of $\Pi^0$ with less total earliness and
            tardiness, $OF^{'}$;
            if $OF^{'} < OF_b$ then
                $OF_b = OF^{'}$;
                $\Pi^b := \Pi$;
                $flag :=$ true;
            end
        end
    end
    return $\Pi^b$ and $OF_b$;
end
```

Figure 9.7: Iterative Local Search, iLS

Speed up procedure

In this section, a simple speed-up procedure to accelerate the insertion phases of the algorithms for the $Fm|prmu|\sum E_j + \sum T_j$ problem is described. Let Π^k be a partial sequence with k jobs and l the job which is to be inserted in position $j \in [1, k+1]$. Similarly to the speed up methods proposed by [96] and [197], this method stores the completion time of each job on each machine of the partial sequence Π^k. When the job l is inserted in each position j of the partial sequence, the completion times of the jobs prior to this position j are already known and do not have to be recomputed. According to several studies, this procedure reduces the CPU times between 30% and 50% and is therefore introduced in each insertion phase of all algorithms implemented in this chapter.

9.3 Computational experience

In this section, the proposed algorithms are compared against the most efficient heuristics in the literature. The procedure adopted to evaluate the algorithms is the following: First, we introduce the set of instances used for both the experimental parameter tuning and the comparison among heuristics. In Section 9.3, a full factorial design of experiments is carried out to find the best values of the parameters of the algorithms proposed. The algorithms under comparison are listed in Section 9.3. Constructive and composite heuristics are compared in Section 9.3, leading to the identification of the set of efficient heuristics for the problem. Finally, in Section 9.3, the efficient heuristics are compared as seed sequences of one of the best metaheuristic for the problem.

Experimental parameter tuning

The proposed heuristic ACH1 uses three parameters: a, b and c. In this section, a full factorial design of experiments is carried out to determine their best values on the set of instances $\mathcal{B}_{C3}$. The following values are chosen for the experiments:

- $a = \{0.8, 0.85, 0.9, 0.95, 1\}$,
- $b = \{0.4, 0.45, 0.5, 0.55, 0.6\}$,
- $c = \{25, 30, 35, 40, 45, 50, 55\}$

In each instance, the ACH1 heuristic is evaluated according to Equation (9.7):

$$RPD3 = \frac{OF - Base}{Base} \cdot 100 \tag{9.7}$$

where OF and $Base$ are the solutions obtained by the ACH1 heuristic and a reference algorithm (NEHedd) respectively.

Since normality and homoscedasticity assumptions are not fulfilled, a non-parametric Kruskal-Wallis test is carried out. The p-values are 0.267, 0.865 and 0.000 for parameters a, b and c respectively. Results show that there is statistically significant differences only between the levels of parameter c. Additionally, among the 175 combinations of a, b and c, the best results are found for $a = 0.90$, $b = 0.55$ and $c = 30$. These values are subsequently used in each heuristic which incorporates the ACH1, i.e. ACH2, ACH3 and ACH4.

Implemented algorithms

The performance of the proposed heuristics is tested against the most efficient heuristics for the problem, as well as for some of the most efficient heuristics for similar scheduling problems. More specifically, due to their excellent performance (see computational evaluations by [137] and [150]), the following heuristics are considered:

- NEHedd$^{\mathrm{or}}$: NEHedd$^{\mathrm{or}}$ is the NEHedd heuristic proposed by [88]) for $Fm|prmu|\sum T_j$. The speed up procedure described in Section 9.2 is not applied to maintain its original version.

- NEHedd$^{\mathrm{et}}$: Heuristic NEHedd$^{\mathrm{or}}$ using the speed up procedure in Section 9.2. Additionally, the evaluation of total tardiness in each iteration is replaced by the evaluation of the sum of total earliness and tardiness.

- Raj: Adaptation of the Raj heuristic by [151], originally proposed for the $Fm|prmu|\sum C_j$ problem. To adapt the heuristic to our problem, the speed up procedure in Section 9.2 is applied, and the original evaluation of total flowtime is replaced by the evaluation of total earliness and tardiness. Additionally, the original initial order is replaced by the EDD rule.

- RZ: Adaptation of the RZ heuristic by [152] proposed for the $Fm|prmu|\sum C_j$ problem, with the initial order replaced by the EDD rule. The speed up procedure is applied, and the evaluation of total flowtime is replaced by the evaluation of total earliness and tardiness.

- RZ_LW: Adaptation of the RZ_LW heuristic by [152], originally proposed for the $Fm|prmu|\sum C_j$ problem. The speed up procedure is applied and the evaluation of total flowtime is replaced by the evaluation of total earliness and tardiness. Furthermore, the EDD rule is used as initial order.

- FRB4$_{\mathrm{k}}$: Adaptation of the FRB4$_{\mathrm{k}}$ heuristic by [150], originally proposed for the $Fm|prmu|C_{max}$ problem. The evaluation of the makespan is replaced by the evaluation of the total earliness and tardiness, and the speed up procedure by [189] is replaced by the proposed one. As in the NEHedd$^{\text{et}}$, the original order is replaced by the EDD rule.

All heuristics are fully recoded for the $Fm|prmu|\sum E_j + \sum T_j$ problem under the same conditions (see Section 3.4) using an Intel Core i7-3770 with 3.4 GHz and 16 GB RAM).

Efficient set of heuristics

In this section, all implemented heuristics are compared using benchmark $\mathcal{B}_3$. Average results in terms of $ARPD1$ are shown in Table 9.2 for each combination of n and m, and in Table 9.4 for each value of the parameters. The CPU time required by each heuristic is shown in Table 9.3 for each n and m. The last two rows show the $ACPU$ and the $ARPT2$ of each heuristic. A summary of the results is graphically shown in Figure 9.8 using $ACPU$ to evaluate the computational effort, while $ARPT2$ is used as indicator in Figure 9.9. In view of the results, the NEHedd$^{\text{et}}$ heuristic clearly outperforms the NEHedd$^{\text{or}}$ in terms of quality of the solution and computational effort. The best $ARPD1s$ are clearly found by the proposed heuristic ACH4 (1.19), and by the RZ_LW heuristic (2.40). Regarding heuristics adapted from other problems, the best results are found by Raj, RZ and RZ_LW, which are either very fast heuristics, or local search methods (using dispatching rules as seed sequences). The good performance achieved by the composite heuristics RZ, RZ_LW, ACH2, ACH3, and ACH4 confirms the conclusions obtained after the analysis of the problem in Section 5.2 which advocated for fast heuristics employing as soon as possible local search methods of full sequences. This fact is also confirmed by the performance of the family of heuristics FRB4$_{\text{k}}$. Each of these heuristics is outperformed in terms of quality of the solutions and computational effort by RZ and ACH3. According to Figure 9.9, the efficient heuristics (set $\mathcal{A}$) are: ACH1, Raj, NEHedd$^{\text{et}}$, ACH2, RZ, ACH3 and ACH4. To statistically justify this statement, we perform a Holm's procedure [73] with the following hypotheses:

- H$_1$: ACH2 = NEHedd$^{\text{or}}$.
- H$_2$: RZ = FRB4$_2$.
- H$_3$: RZ = FRB4$_4$.
- H$_4$: RZ = FRB4$_6$.
- H$_5$: ACH3 = FRB4$_8$.
- H$_6$: ACH3 = FRB4$_{10}$.
- H$_7$: ACH3 = FRB4$_{12}$.

- H_8: ACH4 = RZ_LW.

Results are shown in Table 9.5, where the p-values have been calculated using a non-parametric Mann-Whitney test since the normality and homoscedasticity assumptions were not confirmed (see e.g. [138]). Assuming a confidence of 0.95, only two hypotheses (H2 and H3) are not rejected and the proposed heuristics (ACH2, ACH3 and ACH4) can be therefore considered as statistically efficient. The heuristics of the sets $\mathcal{A}$ are shown in Figure 9.9.

Instance	Raj	NEHedd^{et}	NEHedd^{or}	RZ	FRB4$_2$	FRB4$_4$	FRB4$_6$	FRB4$_8$	FRB4$_{10}$	FRB4$_{12}$	RZ_LW	ACH1	ACH2	ACH3	ACH4
50x10	26.52	22.38	44.23	13.53	14.16	11.62	10.00	9.91	8.79	8.07	2.53	30.15	13.65	8.74	1.41
50x30	20.61	14.20	12.27	10.55	7.86	6.22	5.39	4.56	4.41	4.14	2.59	18.33	9.31	6.11	1.81
50x50	15.06	7.94	6.89	8.03	4.54	3.68	2.85	2.84	1.76	1.60	2.11	11.49	5.95	4.00	1.40
150x10	41.36	37.56	66.72	14.39	25.56	24.13	23.19	21.88	21.59	19.33	1.69	46.12	18.06	9.83	1.65
150x30	31.16	25.09	39.50	14.76	15.89	15.07	14.15	13.07	11.49	11.04	2.53	37.47	15.88	7.40	1.19
150x50	30.61	24.40	25.86	14.93	18.92	17.11	16.36	14.73	14.02	13.10	3.18	33.14	14.33	5.23	0.44
250x10	36.04	29.13	76.55	13.94	22.63	20.62	19.42	18.69	17.61	17.25	2.31	50.20	13.88	7.85	1.09
250x30	40.84	35.95	57.49	15.95	24.08	23.16	21.62	21.04	20.13	18.94	2.50	45.28	19.56	8.96	1.14
250x50	32.37	26.65	39.12	15.29	17.54	16.23	15.34	15.13	13.28	11.87	2.69	37.35	16.44	5.99	1.29
350x10	38.57	30.75	77.89	13.37	23.70	23.05	22.46	20.94	20.71	20.09	1.41	56.46	14.83	8.76	0.82
350x30	48.71	41.83	66.24	18.72	32.40	29.66	28.60	27.43	26.10	26.66	2.83	51.33	22.34	9.05	1.02
350x50	32.56	26.18	47.91	13.49	18.94	16.45	16.34	15.17	15.51	13.08	2.44	39.64	17.25	7.44	0.98
Average	32.87	26.84	46.72	13.91	18.85	17.25	16.31	15.45	14.61	13.76	2.40	38.08	15.12	7.45	1.19

Table 9.2: Average Relative Percentage Deviation $ARPD1$ for the implemented heuristics under the set of instances $\mathcal{B}_3$

Instance	Raj	NEHedd^{et}	NEHedd^{or}	RZ	FRB4$_2$	FRB4$_4$	FRB4$_6$	FRB4$_8$	FRB4$_{10}$	FRB4$_{12}$	RZ_LW	ACH1	ACH2	ACH3	ACH4
50x10	0.00	0.00	0.00	0.01	0.02	0.02	0.02	0.03	0.03	0.04	0.03	0.00	0.00	0.02	0.03
50x30	0.00	0.01	0.01	0.02	0.04	0.05	0.07	0.09	0.10	0.11	0.09	0.00	0.01	0.06	0.08
50x50	0.00	0.01	0.02	0.03	0.06	0.09	0.11	0.14	0.16	0.19	0.17	0.00	0.02	0.08	0.13
150x10	0.01	0.06	0.10	0.13	0.26	0.41	0.55	0.68	0.80	0.91	0.98	0.00	0.09	0.56	0.95
150x30	0.04	0.16	0.35	0.41	0.82	1.30	1.75	2.17	2.56	2.92	3.42	0.01	0.28	1.74	3.20
150x50	0.07	0.26	0.61	0.72	1.44	2.31	3.12	3.89	4.58	5.26	5.84	0.02	0.49	2.64	5.27
250x10	0.06	0.21	0.46	0.56	1.12	1.81	2.47	3.09	3.68	4.24	4.93	0.01	0.38	2.84	4.73
250x30	0.18	0.64	1.60	1.82	3.67	5.92	8.00	9.93	11.80	13.58	19.33	0.03	1.25	10.24	18.88
250x50	0.31	1.13	2.80	3.24	6.50	10.50	14.22	17.74	21.06	24.30	33.00	0.06	2.21	18.64	32.80
350x10	0.15	0.53	1.24	1.52	3.02	4.91	6.72	8.46	10.12	11.73	14.74	0.02	1.03	8.88	14.33
350x30	0.46	1.69	4.30	4.88	9.90	16.15	21.96	27.51	32.70	37.91	55.90	0.06	3.33	31.85	61.23
350x50	0.81	2.98	7.54	8.60	17.37	28.02	38.17	47.65	57.07	65.66	95.51	0.09	5.88	55.56	96.84
$ACPU$	0.17	0.64	1.59	1.83	3.68	5.96	8.10	10.12	12.06	13.90	19.49	0.03	1.25	11.09	19.87
$ARPT2$	0.03	0.11	0.24	0.30	0.61	0.96	1.27	1.57	1.85	2.12	2.43	0.01	0.21	1.33	2.32

Table 9.3: Average Computational Effort ($ACPU$) (in seconds) and Average Relative Percentage Time ($ARPT2$) of the implemented heuristics under the set of instances $\mathcal{B}_3$

Parameter		Raj	NEHedd^{et}	NEHedd^{or}	RZ	FRB4$_2$	FRB4$_4$	FRB4$_6$	FRB4$_8$	FRB4$_{10}$	FRB4$_{12}$	RZ_LW	ACH1	ACH2	ACH3	ACH4
T	0.2	49.71	38.62	92.83	16.45	28.60	26.05	24.71	23.59	21.92	19.90	1.60	65.76	25.09	13.01	2.49
T	0.4	31.61	29.44	36.71	15.93	20.61	19.15	18.10	17.18	16.76	16.40	2.72	36.04	14.30	6.15	0.71
T	0.6	17.28	12.46	10.63	9.36	7.34	6.55	6.12	5.58	5.16	5.00	2.88	12.44	5.98	3.19	0.36
R	0.2	18.40	16.25	21.39	9.35	11.14	10.06	9.60	9.31	8.95	8.83	2.31	22.21	8.93	5.05	0.43
R	0.6	24.85	18.31	35.67	12.50	12.01	11.05	10.27	10.02	9.63	9.26	2.59	28.50	11.52	6.62	0.86
R	1	55.35	45.95	83.11	19.89	33.40	30.64	29.07	27.02	25.27	23.20	2.30	63.53	24.92	10.68	2.27
n	50	20.73	14.84	21.13	10.70	8.85	7.17	6.08	5.77	4.98	4.60	2.41	19.99	9.64	6.28	1.54
n	150	34.38	29.02	44.03	14.69	20.12	18.77	17.90	16.56	15.70	14.49	2.47	38.91	16.09	7.49	1.09
n	250	36.42	30.57	57.72	15.06	21.42	20.00	18.79	18.29	17.00	16.02	2.50	44.28	16.63	7.60	1.17
n	350	39.95	32.92	64.01	15.19	25.01	23.05	22.47	21.18	20.77	19.94	2.23	49.14	18.14	8.42	0.94
m	10	35.62	29.95	66.34	13.81	21.51	19.85	18.77	17.85	17.17	16.19	1.99	45.73	15.11	8.80	1.24
m	30	35.33	29.26	43.88	15.00	20.06	18.53	17.44	16.52	15.53	15.20	2.61	38.10	16.77	7.88	1.29
m	50	27.65	21.29	29.95	12.93	14.99	13.37	12.72	11.97	11.14	9.91	2.60	30.40	13.49	5.67	1.03

Table 9.4: Average Relative Percentage Deviation $ARPD1$ for each heuristic grouped by the values of the parameters

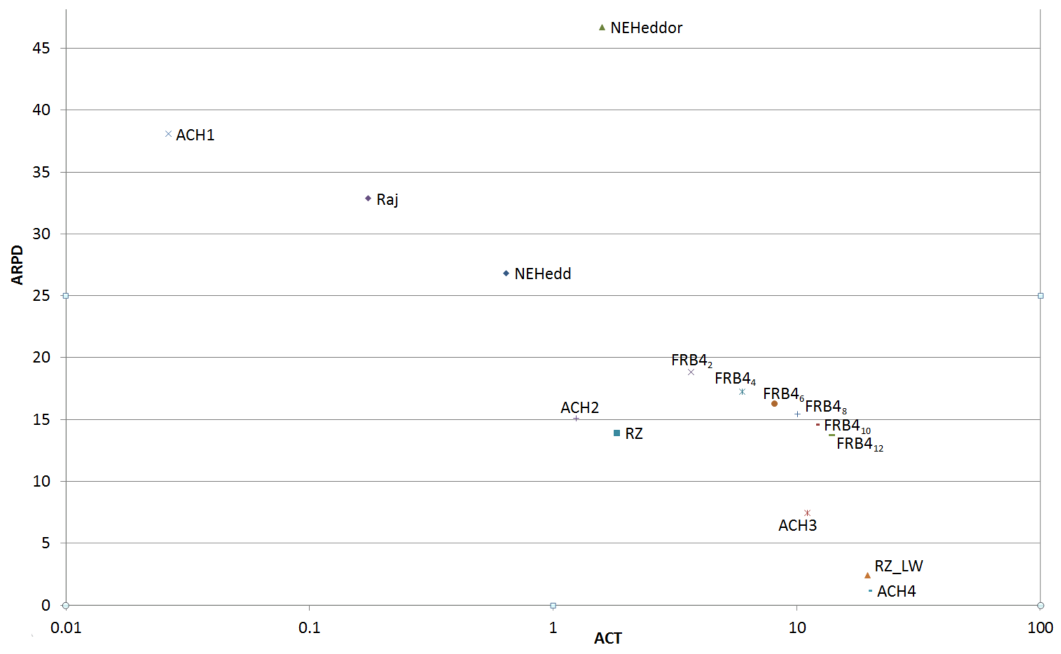

Figure 9.8: $ARPD1$ vs $ACPU$ of implemented heuristics. X-axis ($ACPU$ in seconds) is shown logarithmic scale.

Comparison among efficient heuristics

As there is a trade-off between quality of the solution and computational effort, heuristics in set $\mathcal{A}$ cannot be directly compared in terms of $ARPD1$ due to their different computational efforts. In this section, they are included as initial solution for one of the best metaheuristic for this problem, i.e. the ILS by [117], replacing the original seed sequence of the metaheuristic (EDD rule). Thus, the metaheuristic is run using eight different initial sequences (EDD rule and each heuristic in set $\mathcal{A}$) where the EDD rule is included in the comparison as it is the original seed sequence of the metaheuristic. Each variation of the metaheuristic is run under the same computational conditions described in Section 9.3 using benchmark $\mathcal{B}_3$. In this case, five runs are performed per instance and the average values are recorded. The variations of the ILS are stopped depending on the size of the problem according to expression $n \cdot m \cdot t/2$ (milliseconds)

i	H_i	p-value	Mann-Whitney	$\alpha/(k-i+1)$	Holm's Procedure
1	ACH2 $=$ NEHedd^{or}	0.000	R	0.0063	R
2	ACH3 $=$ FRB4$_8$	0.000	R	0.0071	R
3	ACH3 $=$ FRB4$_{10}$	0.000	R	0.0083	R
4	ACH3 $=$ FRB4$_{12}$	0.000	R	0.0100	R
5	ACH4 $=$ RZ_LW	0.000	R	0.0125	R
6	RZ $=$ FRB4$_2$	0.001	R	0.0167	R
7	RZ $=$ FRB4$_4$	0.069		0.0250	
8	RZ $=$ FRB4$_6$	0.690		0.0500	

Table 9.5: Holm's procedure.

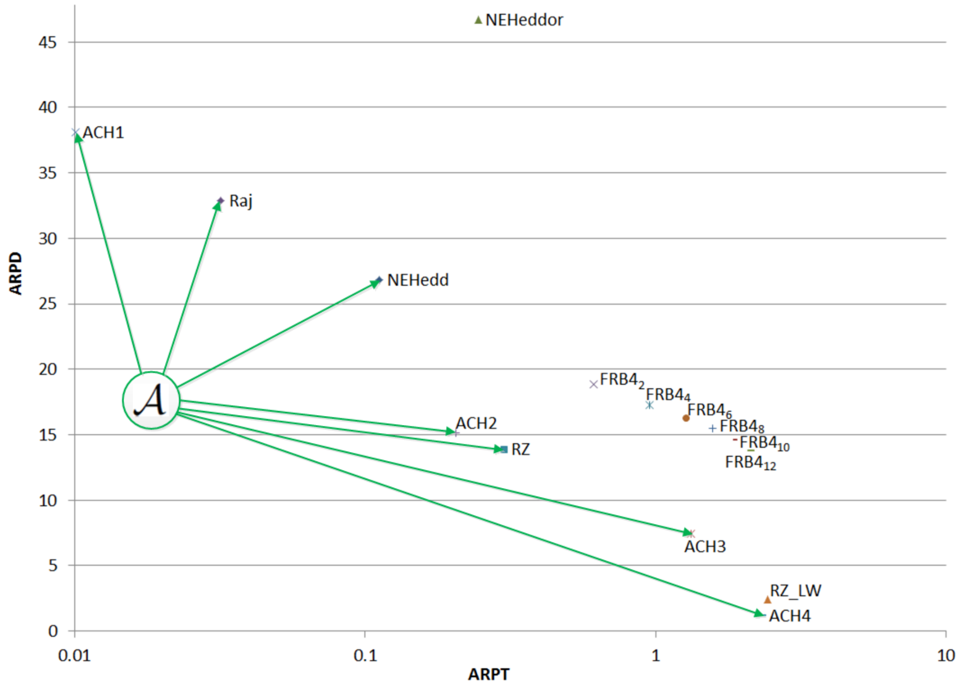

Figure 9.9: $ARPD1$ vs $ARPT2$ of implemented heuristics. X-axis ($ACPU$ in seconds) is shown logarithmic scale.

where $t = 5, 10, 15, 20, 25, 30$ (see e.g. [174] for a similar stopping criterion). Obviously, the CPU time required by each heuristic is included in the CPU time of the metaheuristic, i.e. the clock starts before applying the heuristic. Results of the ILS metaheuristic using different heuristics as initial solution are shown in terms of $ARPD1$ in Table 9.7. Note that using the original seed sequence (EDD rule) in the metaheuristic outperforms several other initial sequences (Raj, NEHedd^{et} and RZ). However, the best $ARPD1s$ are found when embedding the proposed heuristics (ACH1, ACH2, ACH3 and ACH4) in the ILS metaheuristic being e.g. 1.87, 1.94, 1.56 and 1.32 respectively the $ARPD1$ of these heuristics for $t = 10$, as compared to 2.20 obtained by the EDD rule. The best value is found using ACH4 as initial solution regardless the stopping criteria, being 1.08 the lowest $ARPD1$ found for $t = 30$. Additionally, in order to confirm the excellent results found by the ILS using the ACH4 heuristic as seed sequence, a Holm's procedure is carried out comparing the ILS both with the ACH4 heuristic and with the EDD rule. More specifically, the hypotheses tested are:

- H_1: For $t = 5$, ILS(ACH4) = ILS(EDD rule).
- H_2: For $t = 10$, ILS(ACH4) = ILS(EDD rule).
- H_3: For $t = 15$, ILS(ACH4) = ILS(EDD rule).
- H_4: For $t = 20$, ILS(ACH4) = ILS(EDD rule).

i	H_i	p-value	Mann-Whitney	$\alpha/(k-i+1)$	Holm's Procedure
1	ILS(ACH4) = ILS(EDD rule) (for $t = 5$)	0.000	R	0.0083	R
2	ILS(ACH4) = ILS(EDD rule) (for $t = 10$)	0.000	R	0.0100	R
3	ILS(ACH4) = ILS(EDD rule) (for $t = 15$)	0.000	R	0.0125	R
4	ILS(ACH4) = ILS(EDD rule) (for $t = 20$)	0.000	R	0.0167	R
5	ILS(ACH4) = ILS(EDD rule) (for $t = 25$)	0.000	R	0.0250	R
6	ILS(ACH4) = ILS(EDD rule) (for $t = 30$)	0.000	R	0.0500	R

Table 9.6: Holm's procedure for comparisons with metaheuristics.

Parameter t	EDD rule	Raj	NEHedd^{et}	RZ	ACH1	ACH2	ACH3	ACH4
5	3.26	4.58	4.77	3.60	2.97	2.72	1.90	1.39
10	2.20	2.91	3.09	2.43	1.87	1.94	1.56	1.32
15	2.11	2.80	2.96	2.33	1.82	1.85	1.46	1.22
20	2.03	2.69	2.84	2.24	1.76	1.75	1.41	1.16
25	1.95	2.60	2.75	2.17	1.70	1.69	1.34	1.11
30	1.94	2.58	2.71	2.14	1.67	1.67	1.31	1.08

Table 9.7: Average relative deviation index ($ARDI$) of the metaheuristic ILS using different heuristics as initial solution

- H_5: For $t = 25$, ILS(ACH4) = ILS(EDD rule).
- H_6: For $t = 30$, ILS(ACH4) = ILS(EDD rule).

Results of the Holm's procedure are shown in Table 9.6. Each p-value is equal to 0.000 and therefore, each hypothesis is rejected statistically, confirming the previous results.

9.4 Conclusions

In this chapter, we have addressed the PFSP with a just-in-time objective. By incorporating this knowledge, we propose four different heuristics. Firstly, a fast constructive heuristic, ACH1, inserts the jobs one by one at the end of the partial sequence based on an dynamic index. This index is automatically calculated in each iteration depending on the idle times, completion times, earliness and tardiness of the jobs. Then, three composite heuristics ACH2, ACH3 and ACH4 are proposed by incorporating three different local search procedures after ACH1.

The proposed heuristics have been compared under a complete set of instances with the best heuristic for the problem as well as with adaptations of efficient heuristics for similar scheduling problems. The computational results show the excellent performance of the proposed algorithms. In fact, the heuristics ACH1, Raj, NEHedd^{et}, ACH2, RZ, ACH3 and ACH4 are established as efficient, being the ACH4 the heuristic with the lowest $ARPD1$ (1.19).

Finally, the impact of the efficient heuristics is evaluated including them as seed sequences for one of the best metaheuristic for the problem, and for six different stopping criteria. As a result, the four proposed heuristics statistically outperform every other heuristic, thus establishing a new state-of-the-art of approximate solutions for the problem.

Part IV

PROBLEM EXTENSIONS

Chapter 10

PFSP to minimise makespan subject to total tardiness

Among the criteria established to measure the performance of the different schedules shown in Section 2.1, the maximum completion time of a sequence or makespan is related to resource usage, while tardiness refers to the delay of the completion time of a job with respect to its committed due date. Since these are key aspects in manufacturing companies' competitiveness, it seems appropriate to consider both objectives together. Regarding tardiness minimisation, customer due dates may be regarded as "hard" constraints (i.e. deadlines) in some manufacturing scenarios, while in others some flexibility is allowed by the customer as long as the deviation from the completion times of the jobs is limited. In contrast, makespan is an intra-company criteria that is related to maximising machine utilisation, which in turns minimises fixed unit costs. Therefore, one option to balance both objectives is to seek the minimisation of the makespan while allowing only a given deviation from the committed due dates, expressed as the maximum tardiness allowed. Note that this problem includes the special case where no deviation from the jobs' due dates is allowed, thus forcing the fulfilment of the committed due dates.

The problem described in the previous paragraph can be denoted as $Fm|prmu|\epsilon(C_{max}/T_{max})$ (see [193])). This problem belongs to the class of ϵ-constrained multi-criteria scheduling problems, and it has been the subject of several research contributions in the last decades. Since the minimisation of any of the individual criteria (either makespan or maximum tardiness) in a flow shop is NP-hard, the research effort has focused on approximate procedures providing good –but not necessarily optimal– solutions in a relative short period of time. In this regard, the works by [27], [12], [45], and [171] develop different heuristics either for the problem, or for general cases of the problem. In this chapter, we propose a constructive heuristic and a metaheuristic that exploits the specific structure of solutions of the problem to reduce the search space and to accelerate the evaluation of solutions. Both algorithms improve existing ones by a larger degree and constitute therefore the new state-of-art approximate solution procedures for the problem.

The remainder of the chapter is structured as follows: the state of the art for the problem is shown in Section 10.1. In Section 10.2, some definitions and properties of the problem are defined. Sections 10.3 and 10.4 are devoted to propose two algorithms (a constructive heuristic and a metaheuristic) which use the properties discussed previously. The algorithms are compared with the (up to now) state-of-the-art algorithms in Section 10.5 and, finally, conclusions are discussed in Section 10.6.

10.1 Literature review

[27] were the first in proposing a constructive heuristic for the $Fm|prmu|\epsilon(C_{max}/T_{max})$ problem. In their heuristic, assuming a partial sequence Π formed by already scheduled jobs, a (partial) sequence is constructed for each non-scheduled job u_k by placing it as the first job, and then scheduling the jobs in Π after u_k according to the NEH algorithm. Out of these so-obtained sequences, the one with the lowest makespan is chosen for the next iterations (consequently, u_k is removed from the non-scheduled jobs set for the next iteration).

[12] propose a simulated annealing algorithm to solve the $Fm|prmu|\epsilon(Z/T_{max})$ where $Z = \lambda \cdot C_{max} + (1 - \lambda) \cdot T_{max}$, $\lambda \in [0, 1]$. Clearly, our problem is a special case of their problem when $\lambda = 1$. Their algorithm begins with the best sequence among the solutions found by the NEH heuristic, the earliest due date rule and the least slack rule (jobs ordered according to ascending order of $d_j - \sum_{i=1}^{m} t_{ij}$). The procedure iteratively samples neighbour solutions (using an adjacent pairwise interchange neighbourhood) until the stopping criterion is fulfilled.

[45] propose a constructive heuristic, denoted in the following as FL, based on the NEH algorithm to solve the $Fm|prmu|\epsilon(C_{max}/T_{max})$ problem. The heuristic tries to improve the makespan without worsening the tardiness by using a property of the problem. The heuristic is compared with those of [27] and [12] for small and big instances. The results show that the FL outperforms the other ones in terms of both the quality of the solutions and the number of the feasible solutions obtained.

Finally, [171] propose an iterated optimization algorithm to solve the $Fm|prmu|\epsilon(Z/T_{max})$ problem. More specifically, they proposed a high-performance Genetic Algorithm (GA in the following) where the selection procedure is based on n-tournament (see [170]). The fitness values of the individuals are calculated depending on whether all individuals are feasible; feasible and infeasible; or only infeasible.

The algorithm outperforms the FL for the $Fm|prmu|\epsilon(Z/T_{max})$ problem in an extended benchmark. Nevertheless, GA and FL were not compared for the specific $Fm|prmu|\epsilon(C_{max}/T_{max})$ problem.

To summarise the state of the art regarding the $Fm|prmu|\epsilon(C_{max}/T_{max})$ problem, there are some efficient heuristics for the problem, but their performance is not completely clear, as the comparison between the most efficient contributions (i.e. GA and FL) has been only partially conducted. In addition, both mechanisms made extensive use of insertion neighbourhoods, so it could be extremely interesting to devise a mechanism similar to that by Taillard to reduce the computational burden. Finally, it is also to note that all existing procedures make little use (or no use at all) of the knowledge on the problem domain.

10.2 Problem properties

As mentioned in Section 10.1, Taillard's acceleration does not compute the completion times of each job and therefore, it cannot be used to compute the maximum tardiness of the sequence. Indeed, our problem is complicated by the fact that, when inserting a new job σ in position r of an existing partial sequence, an infeasible solution can be obtained due to either the increase in the completion times of the jobs after σ, or due to the completion time of job σ itself. In order to further classify these two possibilities, let us introduce the following definitions:

Definition 10.2.1 (First Feasible Position). *Given a feasible (partial) sequence $\Pi := (\pi_1, \ldots, \pi_k)$, and a non scheduled job σ, let $\Pi'_r := (\pi_1, \ldots, \pi_{r-1}, \sigma, \pi_r, \ldots, \pi_k)$ be the (partial) sequence obtained by the insertion of σ in position r of Π. Then, the First Feasible Position (FFP) is defined as follows:*

$$FFP(\Pi, \sigma) := \arg\min_{1 \leq r \leq k+1} \{T_{\pi_j}(\Pi'_r) \leq \epsilon \quad \forall j = r, \ldots, k\}$$

As it can be seen from the definition, FFP is the lowest position where a new job can be inserted in an existing sequence without causing infeasible due dates in any of the jobs resulting in positions later than the insertion point. It is clear that, in a given instance of the problem and a partial sequence Π, it is not possible to obtain feasible schedules by inserting a non scheduled job σ into a position $j < FFP(\Pi, \sigma)$.

Note also that obtaining FFP for a tuple Π and σ does not guarantee that Π'_j is feasible for $j \geq FFP$, since the computation of FFP does not take into account the potential infeasibility caused by job σ.

Definition 10.2.2 (Last Feasible Position). *Given a feasible (partial) sequence $\Pi := (\pi_1, \ldots, \pi_k)$, and a non scheduled job σ, let $\Pi'_r := (\pi_1, \ldots, \pi_{r-1}, \sigma, \pi_r, \ldots, \pi_k)$ be the (partial) sequence obtained by the insertion of σ in position r of Π. Then, the Last Feasible Position (LFP) is defined as follows:*

$$LFP(\Pi, \sigma) := \arg\max_{1 \leq r \leq k+1} \{T_\sigma(\Pi_r^{'}) \leq \epsilon\}$$

In this manner LFP is the highest position r where job π_r can be inserted without making its completion time infeasible. Note that the feasibility of the jobs in positions $r+1, r+2, \ldots$ is not considered when computing LFP.

The calculation of both limits is of interest due to some straightforward observations which follow from both definitions:

1. If $FFP(\Pi, \sigma) > LFP(\Pi, \sigma)$ for a given tuple Π and σ, then no feasible sequence can be obtained by inserting σ into Π.
2. If $FFP(\Pi, \sigma) \leq LFP(\Pi, \sigma)$ for a given tuple Π and σ, then the sequence obtained by inserting σ in position FFP of Π is feasible.
3. If $FFP(\Pi, \sigma) \leq LFP(\Pi, \sigma)$ for a given tuple Π and σ, then at least one feasible sequence can be obtained by inserting σ in positions between FFP and LFP, inclusive.
4. For a given tuple Π and σ with $FFP(\Pi, \sigma) \leq LFP(\Pi, \sigma)$, the set of feasible sequences obtained by inserting σ in positions between FFP and LFP represent *all* feasible sequences that can be obtained by inserting σ into Π.

Once FFP and LFP are obtained, the sequence with the lowest makespan can be computed by using Taillard's acceleration between both bounds, i.e.:

$$C_{max} = \min_j (C^j_{max}) \quad j = FFP, \ldots, LFP \tag{10.1}$$

where C^j_{max} is obtained using the Taillard's accelerations. Note that the so-found sequence is not necessarily feasible. The advantage of this mechanism lies in speeding up the computations.

In view of the above expressions, the challenge now is to compute both FFP and LFP in an efficient manner. To do so, we introduce the following properties:

Property 10.2.1. *Given an instance of the $Fm|prmu|\epsilon(C_{max}/T_{max})$ problem, and given a tuple Π and σ of a partial sequence and a non-scheduled job respectively, then*

$$\underline{FFP}(\Pi, \sigma) := 1 + \arg\max_j \{C_{m\pi_j} + \min_{1 \leq i \leq m} t_{i\sigma} - d_{\pi_j} > \epsilon\}$$

is a lower bound for $FFP(\Pi, \sigma)$. Furthermore, $\underline{FFP}$ can be computed in $O(n \cdot m)$

Proof. Recall that the completion times of the jobs in Π placed after the insertion of a job σ must increase at least $\min_i(t_{i,\sigma}), i \in [1, \cdots, m]$ (see the Property 12.2.1 in the Chapter 12). Therefore, $C_{m\pi_j} + \min_{1\leq i\leq m} t_{i\sigma}$ is a lower bound of completion time of job in position j after the insertion of σ in position $r < j$. As a consequence, if $C_{m\pi_j} + \min_{1\leq i\leq m} t_{i\sigma} - d_{\pi_j} > \epsilon$, then the due date of job in position j is always infeasible, so $\underline{FFP}$ is a lower bound of FFP.

$\underline{FFP}$ can be computed in two steps: First $M = \min_{1\leq i\leq m} t_{i\sigma}$ is computed in $O(m)$. Next, the expression $E_j = C_{m,\pi_j} + M - d_{\pi_j}$ is computed in $O(n \cdot m)$ for $j = 1, 2, \ldots$ until it verifies that $E_j > \epsilon$. It is thus clear that the computation of $\underline{FFP}$ is $O(n \cdot m)$. □

Property 10.2.2. *Given an instance of the $Fm|prmu|\epsilon(C_{max}/T_{max})$ problem, and given a tuple Π and σ of a partial sequence and a non-scheduled job respectively, then LFP can be computed in $O(n \cdot m)$.*

Proof. First e_{i,π_j} the earliest completion times of the jobs before σ are computed in $O(n \cdot m)$ (see Section 2.2). Then, for a position k where σ can be inserted, the completion time of σ is computed in $O(m)$ by adding the processing times of σ to e_{i,π_j}, and the result is compared to ϵ. Since this comparison is performed for all positions prior to the candidate position where the new jobs is to be inserted, it is clear that LFP can be computed in $O(n \cdot m)$. The detailed pseudo code is presented in Figure 10.2. □

Equipped with these problem properties, we propose efficient approximate procedures based on the insertion of jobs into existing partial schedules. More specifically, in Section 10.3 we present a constructive heuristic for the problem whereas in Section 10.4 we present a non-population based metaheuristic.

10.3 Bounded-insertion-based constructive heuristic, $BICH$

In this section, we present a constructive heuristic based on a so-called *bounded insertion* (BICH) using some properties of the problem, that also repairs infeasibility by means of a tabu local search in each iteration (see pseudo code in Figure 10.1).

More specifically, the algorithm obtains a sequence $\Pi := (\pi_1, \ldots, \pi_n)$ in the following manner: Initially, jobs are sorted in non ascending order of the sum of their processing times, so a sorted sequence $\alpha := (\alpha_1, \ldots, \alpha_n)$ is obtained. The first job in the sorted sequence is also the first job in Π, i.e. $\pi_1 = \alpha_1$. Then, the remaining jobs in α are inserted in Π one by one in the following manner: in iteration k ($k \in [2, n]$), job α_k is removed from α and the following steps are carried out:

- **Compute $e_{i\pi_j}, q_{i\pi_j}$ and $f_{i\pi_j}$.** In this step, the variables required to apply Taillard's acceleration are calculated here according to the expressions described in Section 2.2. These computations can be implemented in $O(n \cdot m)$.

- **Compute** $\underline{FFP}$. In this step, $\underline{FFP}(\Pi, \alpha_k)$ is obtained according to Property 10.2.1.
- **Compute** LFP. In this step, $LFP(\Pi, \alpha_k)$ is obtained according to Property 10.2.2.
- **Obtaining the best makespan between** $\underline{FFP}$ **and** LFP **inclusive**. If $\underline{FFP} \leq LFP$, a set of schedules can be obtained when inserting α_k between these two indices. To select the position of insertion in Π, Taillard's acceleration is employed as described in Section 10.2. If $\underline{FFP} > LFP$, α_k is inserted in position LFP. Note that this does not necessarily mean that the so-obtained partial sequence is infeasible, as $\underline{FFP}$ is a lower bound for FFP.
- **Repairing infeasible solutions**. If the resulting partial sequence Π is infeasible, a Feasible Tabu Search (FTS) procedure is performed to try to get to a feasible solution. The FTS is an iterative procedure which maintains the idea of insertion between $\underline{FFP}$ and LFP. First, for each iteration, infeasible jobs are removed from the partial sequence Π and are randomly ordered. Then, they are successively inserted one by one between the $\underline{FFP}$ and LFP indices (inclusive), but in this case the feasibility of each so-obtained sequence is checked, so Taillard's acceleration cannot be used. Furthermore, a simple tabu procedure is introduced to avoid cycles: Each job has a tabu list of positions previously chosen. Once an infeasible job is inserted in a position, this position is added to the tabu list of such job. Thereby, when a job is to be inserted in the actual sequence, only positions that are not in its tabu list can be chosen. The tabu lists of all jobs are set to zero if the infeasible jobs of the current iteration are different from the previous infeasible jobs. Only in this case jobs from the tabu list can be removed from the lists. The procedure finishes when either there are no more infeasible jobs, or when more than x iterations have been run and the number of infeasible jobs has not decreased during the last iteration. The length of the tabu list is sufficiently long to allow storing each possible position (here the maximum number of iterations of the search method i.e. x). Furthermore, when the output sequence of this procedure is infeasible, the FTS is not further implemented in the rest of iterations of the $BICH$ heuristic.

  The pseudo code of the FTS procedure is shown in Figure 10.3.

10.4 Advanced non-population-based algorithm, $ANPA$

In this section, an Advanced Non-Population-based Algorithm ($ANPA$) is proposed as an extension of the ideas presented in the constructive heuristic $BICH$. The algorithm tries to improve the solution by iteratively performing greedy methods and local search methods. The global procedure of the algorithm is shown in Figure 10.4.

```
Procedure BICH(x)
    $\alpha$ := Jobs ordered by descending sums of processing times where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$;
    $\Pi := \{\alpha_1\}$;
    $flag$ := true;
    for $k = 2$ to $n$ do
        Calculate $e_{i\pi_{j_1}}, q_{i\pi_{j_1}}$ and $f_{i\pi_{j_2}}$ for $i = 1 \ldots m$, $j_1 = 1 \ldots k$ and $j_2 = 1 \ldots k+1$;
        for $j = 1$ to $k$ do
            if $e_{m,\pi_j} + \min_{i=1\ldots m}(t_{i\alpha_k}) - d_{\pi_j} > \varepsilon$ then
                $\underline{FFP} = j + 1$;
            end
        end
        $LFP := CalculateExactlyLFP(\Pi, \alpha_k, k, \{e_{i\pi_{j_1}}\})$;
        Test job $\alpha_k$ between the positions $\underline{FFP}$ and $LFP$ of $\Pi$;
        $\Pi$ := permutation obtained by inserting $\alpha_k$ in the position $j \in [\underline{FFP}, LFP]$ of $\Pi$ with lowest
        makespan using Taillard's Acceleration (note that infeasible permutations are allowed here as
        feasibility is not checked);
        if $flag = true$ then
            $\Pi := FeasibleTabuSearch(\Pi, x)$;
            if $\Pi$ is infeasible then
                $flag$ := false;
            end
        end
    end
end
```

Figure 10.1: *BICH*

```
Function CalculateExactlyLFP($\pi, NewJob, k, \{e_{i\pi_j}\}$)
    $flag$ := false;
    $LFP := 0$;
    $C_0 := 0$;
    for $i = 1$ to $m$ do
        $C_0 = C_0 + t_{i,NewJob}$;
    end
    if $C_0 - d_{NewJob} > \varepsilon$ then
        $flag$ := true;
    end
    $j := 1$
    while $j < k$ and $flag = false$ do
        $C_j := 0$;
        for $i = 1$ to $m$ do
            $C_j = \max\left(C_j, e_{i,\pi_j}\right) + t_{i,NewJob}$
        end
        if $C_j - d_{NewJob} > \varepsilon$ then
            $flag$ := true;
        else
            $LFP{+}{+}$
        end
        $j{+}{+}$;
    end
    return $LFP$;
end
```

Figure 10.2: Procedure CalculateExactlyLFP

```
Procedure FeasibleTabuSearch($\pi, x$)
    $\gamma$ := Infeasible jobs of $\Pi$;
    $n_\gamma$ := Number of infeasible jobs in $\Pi$, $|\gamma|$;
    $n_\gamma^{old} := n_\gamma + 1$
    $\#Iterations = 1$;
    while $n_\gamma$ !=0 and ($n_\gamma < n_\gamma^{old}$ or $\#Iterations <= x$) do
        $\Pi$ := Extract jobs $\gamma$ of $\Pi$;
        $\gamma$ := Randomly order infeasible jobs $\gamma$;
        if Infeasible jobs are different from last iteration then
            Empty tabu list;
        end
        for $k = n_\gamma$ to 1 do
            for $j = 1$ to $k$ do
                if $e_{M,\pi_j} + \min_i(t_{i\gamma_k}) - d_{\pi_j} > \varepsilon$ then
                    $\underline{FFP} = j + 1$;
                end
            end
            $LFP := CalculateExactlyLFP(\Pi, \gamma_k, k, e_{i\Pi_j})$;
            Test job $\gamma_k$ in the feasible and non-tabu positions between $\underline{FFP}$ and $LFP$ of $\Pi$ and denote $bj$
            the position with the lowest makespan;
            $\Pi$ := permutation obtained by inserting $\gamma_k$ in $bj$;
            Add position $bj$ to the tabu list of job $\gamma_k$;
        end
        $n_\gamma^{old} := n_\gamma$
        $\gamma$ := Infeasible jobs of $\Pi$;
        $n_\gamma = |\gamma|$;
        $\#Iterations + +$;
    end
end
```

Figure 10.3: Procedure FeasibleTabuSearch

```
Procedure ANPA(d, x, T)
    $(\Pi, C_{max}) = BICH(x)$;
    $\Pi = BRLS(\Pi, C_{max})$;
    while stopping criterion is not reach do
        $\Pi^1 = \Pi$;
        $\Pi^1$ := randomly remove $d$ jobs from $\Pi^1$ and insert it in $\Pi^D$;
        $(\Pi^2, C_{max}) := ConstructionPhase(\Pi^D, \Pi^1)$;
        $\Pi^2 := BRLS(\Pi^2, C_{max})$;
        $\Pi^2 := FeasibleTabuSearch(\Pi^2, x)$;
        $\Pi := SimulatedAnnealingCriterion(\Pi^2, T)$;
    end
end
```

Figure 10.4: $ANPA$

```
Procedure ConstructionPhase($\Pi^D, \Pi$)
    for $k = n - d + 1$ to $n$ do
        Calculate $e_{i\pi_{j_1}}, q_{i\pi_{j_1}}$ and $f_{i\pi_{j_2}}$ for $i = 1 \dots m$, $j_1 = 1 \dots k$ and $j_2 = 1 \dots k + 1$;
        for $j = 1$ to $k$ do
            if $e_{m,\pi_j} + \min_i(t_{i\pi^D[k-(n-d)]}) - d_{\pi_j} > \varepsilon$ then
                $\underline{FFP} = j + 1$;
            end
        end
        $LFP := CalculateExactlyLFP(\pi, \pi^D_{k-(n-d)}, k, \{e_{i\pi_{j_1}}\})$;
        Test job $\pi^D_{k-(n-d)}$ between the positions $\underline{FFP}$ and $LFP$ of $\Pi$;
        $\Pi$ := permutation obtained by inserting $\pi^D_{k-(n-d)}$ in the position $j \in [\underline{FFP}, LFP]$ of $\Pi$ with the
        lowest makespan using Taillard's Acceleration (note that infeasible permutations are allowed here
        as feasibility is not checked);
    end
end
```

Figure 10.5: ConstructionPhase

$ANPA$ starts with the sequence obtained by the heuristic $BICH$ and tries to improve it by means of a bounded relative local search (denoted as $BRLS$) explained below in more detail. Then, the following phases are repeated until the stopping criterion is reached:

- **Jobs Determination Phase**. In this phase, d jobs are randomly chosen to be removed from the sequence. The set of removed jobs is denoted Π_D.

- **Construction Phase**. Jobs in Π_D are re-inserted one by one in the sequence following a similar procedure as in the $BICH$ heuristic, but without applying the FTS procedure after each insertion. This phase is explained in detail in Figure 10.5.

- **Bounded RLS (BRLS)**. The solution of the previous phase is improved by a relative local search method. One by one, jobs are removed from the sequence, tried to be inserted in each position $j \in [1, n]$ and finally, placed in the position with the lowest makespan using Taillard's acceleration

```
Procedure $BRLS(\Pi, C_{max})$
    $h = 1$;
    $i = 1$;
    $\Pi^b := \Pi$;
    while $i <= n$ do
        $k := h \bmod n$;
        $\Pi^0$ := remove job $\pi_k$ from $\Pi$;
        Calculate $e_{i\pi_{j_1}}, q_{i\pi_{j_1}}$ and $f_{i\pi_{j_2}}$ for $i = 1 \ldots m$, $j_1 = 1 \ldots k$ and $j_2 = 1 \ldots k+1$;
        for $j = 1$ to $k$ do
            if $e_{m,\pi_j} + \min_i(t_{i\pi_k}) - d_{\pi_j} > \varepsilon$ then
                $\underline{FFP} = j + 1$;
            end
        end
        $LFP := CalculateExactlyLFP(\Pi^0, \pi_k, n-1, \{e_{i\pi_{j_1}}\})$;
        Test job $\pi_k$ between the positions $\underline{FFP}$ and $LFP$ of $\Pi^0$;
        $\Pi$ := permutation obtained by inserting $\pi_k$ in the position $j \in [\underline{FFP}, LFP]$ of $\Pi$ with lowest
        makespan, $C^{'}_{max}$, using Taillard's Acceleration (note that infeasible permutations are allowed here
        as feasibility is not checked);
        if $C^{'}_{max} < C_{max}$ then
            $C_{max} = C^{'}_{max}$;
            $i = 1$;
            $\Pi^b := \Pi$;
        else
            $i++$;
        end
        $h++$;
    end
    return $\Pi^b$;
end
```

Figure 10.6: Bounded Relative Local Search, *BRLS*

between $\underline{FFP}$ and LFP inclusive. This procedure finishes when n jobs are tried without improving the current best makespan. The pseudo code of this local search method is shown in Figure 10.6.

- **Feasible Tabu Search**. After the ConstructionPhase and the $BRLS$ procedures, FTS is implemented in order to try to reach feasibility when the solution is infeasible.

- **Simulated Annealing-like Acceptance Criterion**. To add diversification to the algorithm, solutions are kept according to a simple simulated annealing procedure. When a solution, Π_2, is worse than the local search optimum, Π, it is maintained only if:

$$random \leq exp\left\{\frac{-(C_{max}(\Pi^2) - C_{max}(\Pi))}{Temperature}\right\}$$

where $random$ is a random number between 0 and 1 and the $Temperature$ is a function that depends on parameter T:

$$Temperature = T \cdot \frac{\sum_{\forall i}\sum_{\forall j} t_{i,j}}{n \cdot m \cdot 10}$$

The temperature parameter has been generated following the suggestions by [135] (see e.g. [138] and [174] for similar approaches).

10.5 Computational results

In this section, the performance of the proposed algorithms $BICH$ and $ANPA$ is compared with the best algorithms so far for the problem, i.e. the GA by [171] and the constructive heuristic FL by [45]. Additionally, two efficient heuristics for makespan minimisation (i.e. the NEH heuristic, and the iterated greedy algorithm, IG_{RS_LS}) are included in the comparison as they are two of the most efficient constructive heuristic and iterative improvement algorithm, respectively. The adaptations of these heuristic to our problem are denoted A_NEH and A_IGA, respectively. When adapting both methods to the proposed problem, the following assumptions are adopted:

- The objective function remains the original of these algorithms, i.e. the minimisation of makespan.
- In the insertion phases of the algorithms, only feasible sequences are considered, i.e. the jobs to be inserted are placed in the feasible position with the lowest makespan.
- Taillard's acceleration is removed from both algorithms since the calculation of the tardiness for each job does not allow its use.

To be able to determine the most efficient algorithms for the problem, all methods have been coded under the same conditions (see Section 3.4), and using an Intel Core i7-3770 with 3.4 GHz and 16GB RAM. The algorithms are tested using the set of instances $\mathcal{B}_3$. Furthermore, in order to increase the accuracy of the iterated improvement algorithms, five runs have been performed per instance and the average values are recorded for the makespan and for the CPU times.

The same stopping criteria as in [171] are applied for the iterative improvement algorithms. These stopping criteria depend on the size of the instance (i.e. the number of jobs and machines) following the expression $t \cdot n \cdot m \cdot /2$ milliseconds where the values 2, 5, 20 and 60 are tested for the parameter t. The FL and BICH constructive heuristics stop naturally when their final sequences are constructed.

Due to the fact that the problem under consideration is subject to maximum tardiness, the evaluation of the quality of both constructive and iterative algorithms is not trivial. Usually, the decision maker would first look for the feasibility of the solutions (i.e. tardiness of each job lower than the maximum tardiness) and, once it is achieved, he/she would look for a low value in the makespan. Finally, the quality of the sequences obtained by each algorithm has to be balanced against the time interval required to obtain the sequences, as the high CPU time requirements posed by some of the algorithms may not be acceptable for some scenarios. Therefore, there is a trade-off among these goals that increases the difficulty of a direct comparison of the algorithms. To make an exhaustive analysis of all these aspects, three indicators have been chosen to determine the quality of the solutions obtained by the algorithms, as well as the CPU time required to obtain the solutions. The indicators are:

- Number of feasible solutions obtained by each procedure.
- Makespan value of the solution (in terms of Average Relative Percentage Deviation) obtained by each procedure.
- Number of instances with the best solution obtained by each procedure.

Regarding the computational time requirements, note we use the average CPU time ($ACPU$), which is both instance- and instance-size- dependent indicator (see Section 3.3). However, similar results are also found when using a dimensionless time indicator. Particularly, the average relative percentage computation time described in 3.3 has been tested with similar results. In order not to excessively increase the extension of the chapter, these findings are not detailed.

Experimental parameter tuning

The proposed algorithms use three parameters: T, d, and x. Therefore, it is interesting to investigate the values of these parameters for which the algorithms reach the best performance. Parameter x is used in both $BICH$ and $ANPA$, while the other two parameters are included only in $ANPA$. In order to simplify the experimentation, the three parameters have been tested only for $ANPA$, and the value obtained for parameter x was also chosen for the constructive heuristic $BICH$. The level of parameters tested are:

- $T \in [0.1, 0.2, 0.3, 0.4]$
- $d \in [4, 5, 6, 7]$
- $x \in [10, 20, 30]$

$ANPA$ is tested following the same calibration test as in [197] by means of benchmark $\mathcal{B}_{C3}$. The stopping criterion adopted is to halt the procedure when the CPU time in milliseconds reaches the value $n \cdot (m/2) \cdot 20$. To establish statistically significant differences between parameters T, d and x, a non-parametric Kruskal-Wallis test is performed, since the normality and homoscedasticity assumptions required for an analysis of variance were not satisfied. As a result of the test, statistically significant differences between

Table 10.1: Values of the three quality indicators and the average CPU time of each algorithm

Algorithm	#Feasible Solutions	# Best Instances	ARPD	Average CPU Time (s.)
A_NEH	277	9	2.83	1.27
$BICH$	490	14	3.24	0.43
FL	448	1	6.96	15.22
$ANPA(t = 2)$	491	69	0.76	6
$A\_IGA(t = 2)$	303	26	1.32	6
$GA(t = 2)$	435	18	2.73	6
$ANPA(t = 5)$	491	80	0.52	15
$A\_IGA(t = 5)$	314	28	1.13	15
$GA(t = 5)$	439	26	2.23	15
$ANPA(t = 20)$	491	112	0.20	60
$A\_IGA(t = 20)$	333	43	0.80	60
$GA(t = 20)$	446	39	1.63	60
$ANPA(t = 60)$	492	491	0.00	180
$A\_IGA(t = 60)$	342	68	0.56	180
$GA(t = 60)$	457	47	1.37	180

the levels of the parameters x and T were found, but not for d since the significance values were 0.011, 0.000 and 0.870 respectively. The best combination of parameters was found for $d = 5$, $T = 0.4$ and $x = 30$, so these were used in the computational experience carried out in the next section.

Number of feasible solutions

The average number of feasible solutions obtained by the implemented algorithms are shown in Table 10.1. For 494 instances out of the 540 instances in the benchmark it was possible to find a feasible solution by one/several algorithms. Among them, 492 feasible solutions were found by $ANPA$ for the stopping criterion $t = 60$, being the best algorithm in terms of the number of feasible solutions obtained. Next is the $ANPA$ heuristic with 491 feasible for the stopping criteria $t = 2$, $t = 5$ and $t = 20$, 490 for the $BICH$ heuristic. 457 feasible solutions were found by GA with $t = 60$ followed by the FL heuristic with 448 feasible solutions. The worst results were obtained for A_NEH and A_IGA algorithms.

It is worth to note that both $BICH$ and $ANPA$ found more feasible solutions within lesser CPU time than the rest of the procedures, a remarkable result specially as $BICH$ had very small CPU requirements. As it can be seen in the Table 10.1 and in Figure 10.7, the efficient algorithms following this criterion would be: $BICH$, $ANPA(t = 2)$ and $ANPA(t = 60)$.

Average relative percentage deviation

The makespan of the solutions obtained by each algorithm can be evaluated by means of the $ARPD1$, see Expression (3.1). It has to be noted that $RPD1$ is computed only if a feasible solution is found by

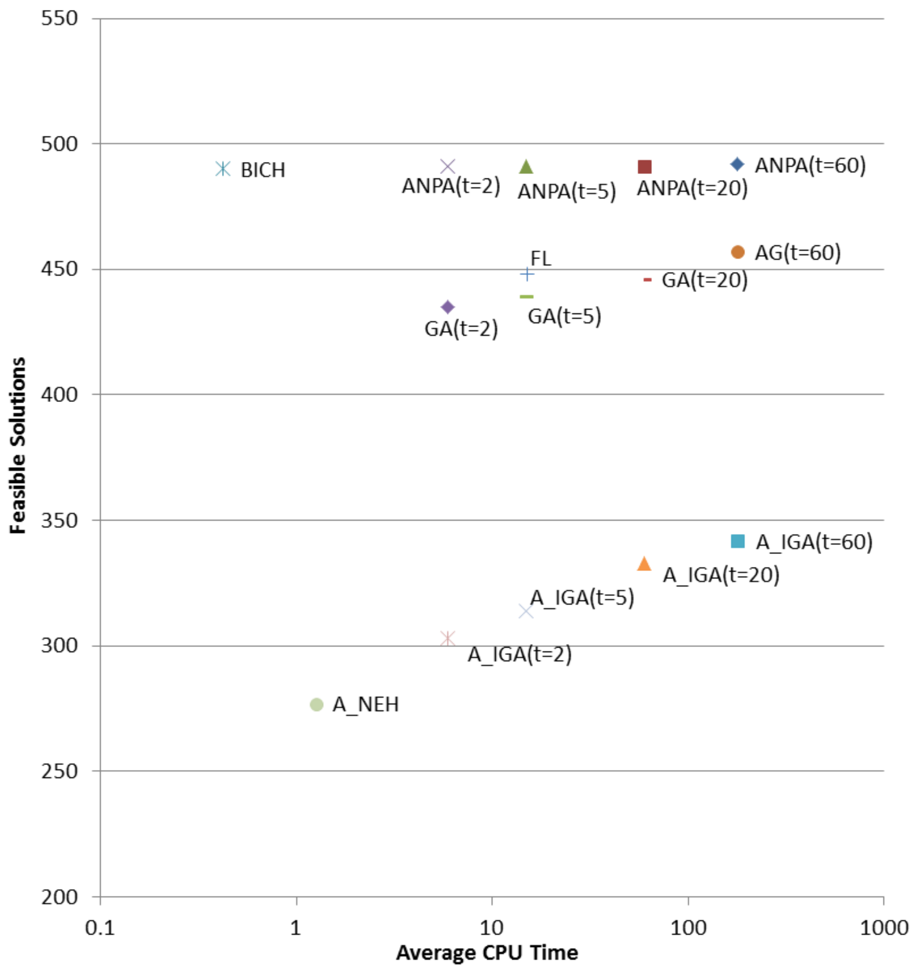

Figure 10.7: Number of feasible solutions vs $ACPU$ for each algorithm. X-Axis is shown in logarithmic scale.

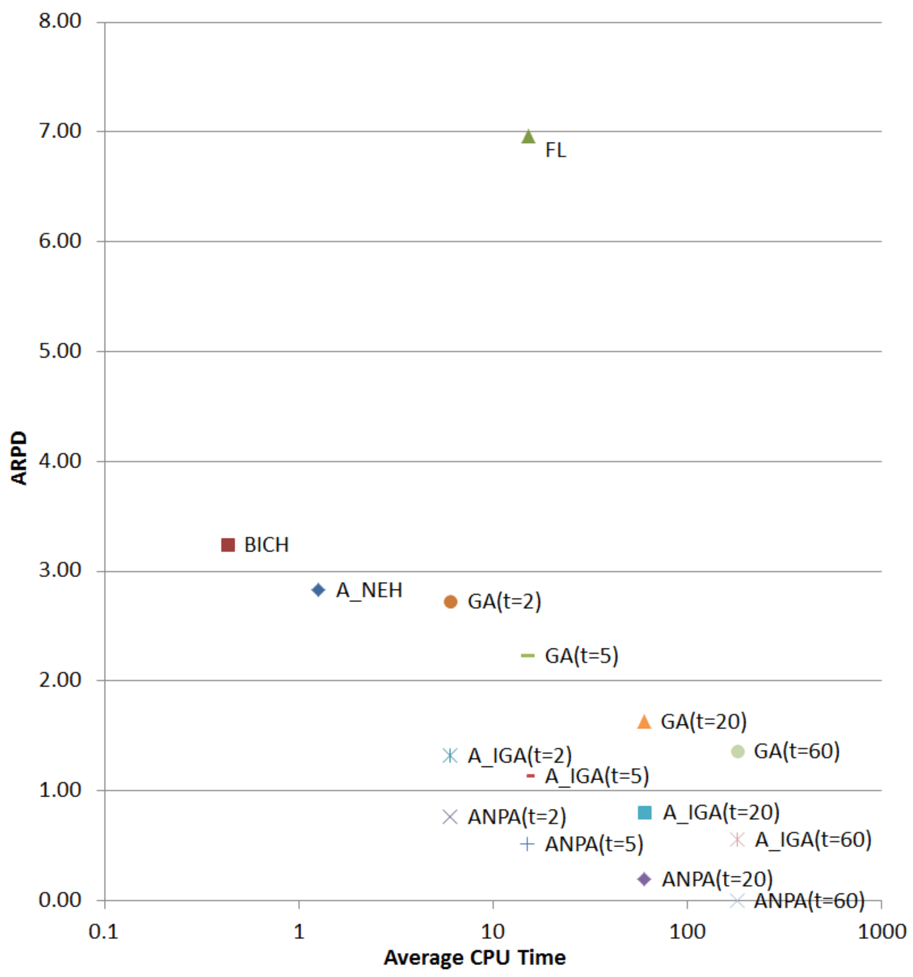

Figure 10.8: $ARPD1$ vs $ACPU$ for each algorithm. X-Axis is shown in logarithmic scale.

the algorithm, otherwise the results would be greatly biased.

The $ARPD1$ is shown in Table 10.1. Since certain algorithms do not find feasible solutions for some instances for which others do, the $ARPD1$ is calculated with different sample sizes depending on the algorithm, e.g. 333 instances for $A\_IGA(t = 20)$ and instances 491 by $ANPA(t = 20)$. This fact might cause that algorithms with lesser feasible solutions than other ones could have less $ARPD1$, as it is the case with $A\_IGA(t = 20)$ and $A\_IGA(t = 60)$ as compared to $GA(t = 60)$. Nevertheless, we also include it in the analysis since it is the usual way in which this analysis is carried out (see e.g. [45, 171]). As it can be seen in Figure 10.8, the efficient heuristics with $ARPD1$ as indicator would be A_NEH, $BICH$, $ANPA(t = 2)$, $ANPA(t = 5)$, $ANPA(t = 20)$ and $ANPA(t = 60)$. In order to statistically justify this statement, we use Holm's procedure [73] with the following hypotheses:

- H_1: $ANPA(t = 2) = GA(t = 2)$
- H_2: $ANPA(t = 2) = A\_IGA(t = 2)$

Table 10.2: Holm's procedure for multiple hypotheses. R indicate that hypothesis is reject by Mann-Whitney and/or Holm's procedure

i	H_i	p-value	Mann-Whitney	$\alpha/(k-i+1)$	Holm's Procedure
1	$ANPA(t=2) = GA(t=2)$	0.000	R	0.0056	R
2	$ANPA(t=2) = A\_IGA(t=2)$	0.000	R	0.0063	R
3	$ANPA(t=5) = GA(t=5)$	0.000	R	0.0071	R
4	$ANPA(t=5) = A\_IGA(t=5)$	0.000	R	0.0083	R
5	$ANPA(t=5) = FL$	0.000	R	0.0100	R
6	$ANPA(t=20) = GA(t=20)$	0.000	R	0.0125	R
7	$ANPA(t=20) = A\_IGA(t=20)$	0.000	R	0.0167	R
8	$ANPA(t=60) = GA(t=60)$	0.000	R	0.0250	R
9	$ANPA(t=60) = A\_IGA(t=60)$	0.000	R	0.0500	R

- H_3: $ANPA(t=5) = GA(t=5)$
- H_4: $ANPA(t=5) = A\_IGA(t=5)$
- H_5: $ANPA(t=5) = FL$
- H_6: $ANPA(t=20) = GA(t=20)$
- H_7: $ANPA(t=20) = A\_IGA(t=20)$
- H_8: $ANPA(t=60) = GA(t=60)$
- H_9: $ANPA(t=60) = A\_IGA(t=60)$

The p-value of each hypothesis is calculated using a non-parametric Mann-Whitney test (see [138]). Then, Holm's procedure orders the hypotheses according to these p-values in non-decreasing order. The procedure rejects hypothesis i if its p-value is lower than $\alpha/(k-i+1)$ where k is the number of hypotheses. The results of this statistical analysis are shown in Table 10.2. As the p-values are always lower than $\alpha/(k-i+1)$, each hypothesis is rejected justifying the statement regarding the efficient algorithms in terms of their $ARPD1$. In fact, each p-value of the non-parametric Mann-Whitney analysis is 0.000.

Number of instances with the best makespan

The third indicator used in this section is the number of instances where each algorithm finds the best solution. This indicator is related to both the feasibility and the makespan in each instance of the algorithm. Results are shown in Table 10.1 for each algorithm. On the one hand, regarding the constructive heuristics, the $BICH$ algorithm finds the best solution in 14 instances as compared to the 9 and 1 of the A_NEH and FL heuristics respectively. On the other hand, the $ANPA$ algorithm is clearly the best with 69, 80, 112 and 491 instances for the stopping criteria $t = 2, 5, 20$ and 60 respectively. Regarding

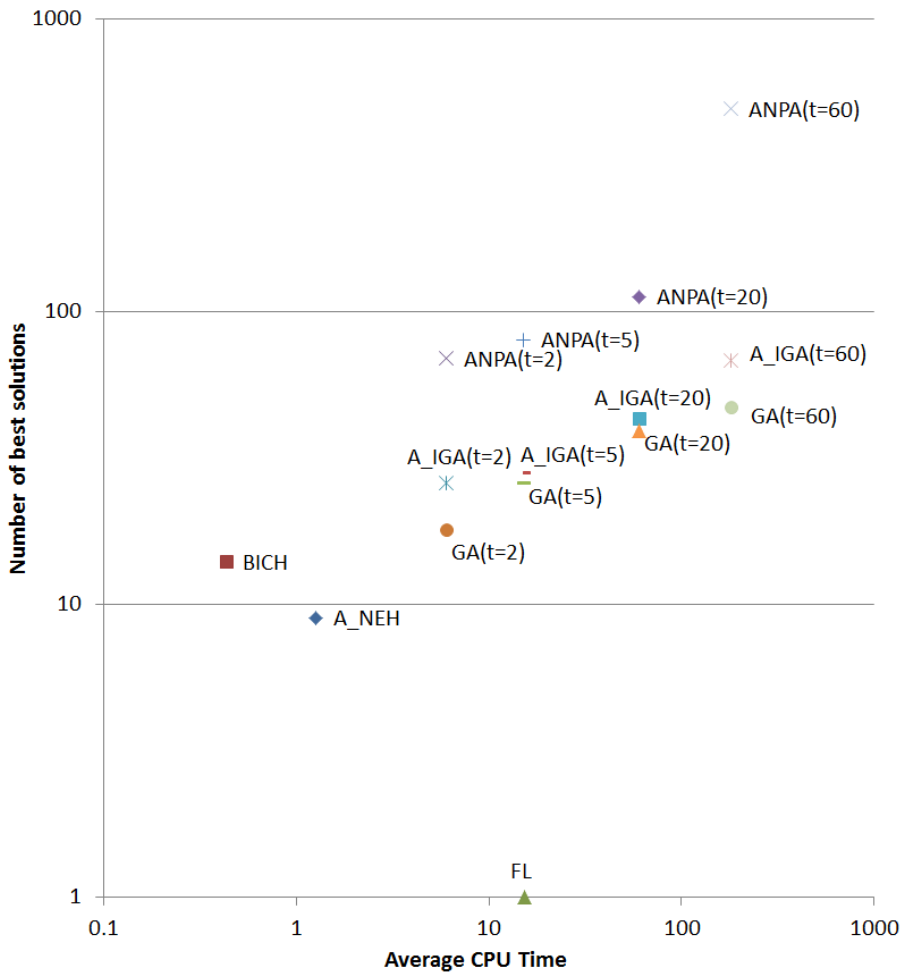

Figure 10.9: Amount of best solutions vs $ACPU$ for each algorithm. X-Axis and Y-Axis are shown in logarithmic scale.

the other two iterative improvement algorithms (A_IGA and GA), A_IGA slightly improves GA for each stopping criterion. Taking into account this indicator, the efficient algorithms would be $BICH$, $ANPA(t = 2)$, $ANPA(t = 5)$, $ANPA(t = 20)$ and $ANPA(t = 60)$, as shown in Figure 10.9.

Different distributions for the processing times

The above analyses have been performed with processing times following a uniform distribution, as it is usual in the literature for the PFSP (see e.g. the benchmarks of [190] and [30]). In this section, three additional benchmarks have been generated using different distributions for the processing times in order to evaluate the robustness of the results. The procedure to generate the three benchmarks is the same as in $\mathcal{B}_3$, with the exception of the distribution of the processing times, which follow Exponential (positive and negative) and Normal distributions, respectively. Hence a total of 540 instances are generated

Table 10.3: Values of the three quality indicators and the average CPU time of each algorithm considering different distributions of the processing times. The exponential distribution (positive and negative) are denoted by EP and EN respectively, as well as the normal distribution is denoted by N.

Algorithm	#Feasible Solutions			# Best Instances			$ARPD1$			Average CPU Time (s.)		
	EP	EN	N	EP	EN	N	EP	EN	N	EP	EN	N
A_NEH	354	253	316	3	17	8	2.63	2.90	2.82	1.34	1.34	1.23
$BICH$	537	458	495	3	15	9	2.73	3.57	3.10	0.58	0.75	0.36
FL	490	420	443	0	3	2	6.19	9.47	6.96	14.18	17.53	13.90
$ANPA(t = 2)$	537	458	495	34	84	55	0.63	0.88	0.73	6	6	6
$A\_IGA(t = 2)$	392	270	343	8	27	25	1.24	1.48	1.38	6	6	6
$GA(t = 2)$	511	416	433	5	19	18	2.10	3.54	2.57	6	6	6
$ANPA(t = 5)$	538	459	495	47	96	69	0.43	0.59	0.49	15	15	15
$A\_IGA(t = 5)$	403	281	352	9	38	28	1.07	1.26	1.17	15	15	15
$GA(t = 5)$	515	426	445	8	36	21	1.74	2.88	2.13	15	15	15
$ANPA(t = 20)$	538	459	495	77	122	98	0.17	0.23	0.19	60	60	60
$A\_IGA(t = 20)$	421	292	364	19	55	40	0.77	0.90	0.84	60	60	60
$GA(t = 20)$	522	442	454	20	51	34	1.26	2.08	1.53	60	60	60
$ANPA(t = 60)$	538	459	495	535	455	490	0.00	0.01	0.00	180	180	180
$A\_IGA(t = 60)$	432	304	370	35	84	57	0.55	0.63	0.59	180	180	180
$GA(t = 60)$	526	444	457	29	65	43	1.04	1.69	1.26	180	180	180

per benchmark representing a total of 1620 instances. In order to have homogeneous results, the same mean (i.e. 50.5) is chosen for those distribution. In the case of the normal distribution, the standard deviation is chosen to achieve a moderate variation of the processing times which means, according to [74], a coefficient of variation between 0.75 and 1.33. Therefore, a value of 1 is used for the coefficient of variation. Additionally, the distributions are truncated and the lower bound and upper bounds are set to 1 and 100, the same as in the uniform distribution. A summary of the results is shown in Table 10.3 for the aforementioned three indicators. The results are very similar to that found using the uniform distribution (see Table 10.1). Additionally, the excellent behaviour and the efficiency of the two proposed algorithms (for the three indicators) are also confirmed in these benchmarks being e.g. the $ARPD1$ of the $ANPA(t = 60)$ algorithms less than 0.01.

10.6 Conclusions

This chapter addresses the permutation flow-shop scheduling problem to minimise the makespan subject to that the tardiness of jobs does not exceed a given maximum tardiness. After analysing the problem and deriving some properties, a constructive heuristic $BICH$ and a non-population based algorithm $ANPA$ are proposed. The performance of both algorithms has been evaluated against the FL and GA algorithms which are the (up to now) state-of-the-art algorithms for the problem under study on an extensive benchmark of 540 instances. Additionally, two of the most efficient algorithms for the $Fm|prmu|C_{\max}$ problem are also included in the comparison.

The efficiency of the two algorithms proposed has been shown according to three different measures of the quality of the solutions: number of feasible solutions, average relative percentage deviation, and number of instances with the best solution. Although the determination of the best algorithms for the problem under study is not trivial due to the existence of infeasible solutions, the proposed algorithms $BICH$ and $ANPA$ have been found to be the most efficient algorithms for each one of the three indicators analysed. Among the 494 feasible instances found in the benchmark, $ANPA(t = 60)$ finds the best solution for 491 instances with an $ARPD1$ equal to 0.00. The performance of $BICH$ is also noteworthy, as it improves several iterative improvement algorithms using much lesser CPU time. These results are also confirmed in other three different benchmarks (of 540 instances each one) generated using three different distributions for the processing times of the jobs. With respect to the rest of the algorithms, it is not clear whether A_IGA outperforms GA or vice versa, since the latter A_IGA is better for the last two indicators, but finds less feasible solutions. The same happens when comparing FL and A_NEH.

Chapter 11

Blocking flowshop scheduling problem

In the classical permutation flowshop scheduling problem studied above, unlimited buffers capacity between two consecutive machines are considered. However, zero-buffer flowshops are very common in several industrial sectors, such as iron and steel industry, chemical and pharmaceutical industries, just-in-time production lines and in-line robotic cells (see e.g. [159], [181], [59] and [57]). This problem is usually denoted as blocking flowshop scheduling problem (BFSP) since a job blocks a machine until the next machine is available. Therefore, interest in this problem is increasing over the past years ([161]), although there are not many algorithms as compared to the number of heuristics and metaheuristics for the traditional permutation flowshop scheduling problem –denoted as PFSP– (see e.g. [172] and [137]), which is one of the most studied problems in Operations Research.

The problem to minimise total flowtime (makespan) is denoted as $Fm|block|\sum C_j$ ($Fm|block|C_{max}$) according to the notation by [58]. Note that as there are zero-capacity buffers between two consecutive machines, several jobs cannot wait at the same time before the machine and the job sequence must therefore be the same on every machine. As a conclusion, $n!$ schedules have to be considered, i.e. the number of solutions is the permutation of n jobs.

In this chapter, we propose an efficient constructive heuristic for the BFSP with flowtime objective based on beam search which can easily be adapted to makespan minimisation. The proposed algorithm outperforms existing heuristics for the $Fm|block|\sum C_j$ and $Fm|block|C_{max}$. Additionally, we test adaptations of the most efficient algorithms for the PFSP to minimise makespan and total flowtime (respectively denoted as $Fm|prmu|C_{max}$ and $Fm|prmu|\sum C_j$ according to [58]). We include them in the comparison since algorithms originally implemented for the PFSP have turned to be efficient algorithms for several decision problems (see e.g. the iterated greedy proposed by [174] or the NEH heuristic by [127]). The resulting computational evaluation is composed of a total of 36 heuristics which are fully recoded and exhaustively compared under the same conditions. Additionally, we introduce a speed-up method to accelerate the insertion phases of all algorithms.

The rest of the chapter is organised as follows. The state of the art is analysed in Section 11.1. In Section 11.2, the problem and the notation is described. The beam-search-based constructive heuristic is proposed in Section 11.3. In Section 11.4, a complete comparison of heuristics is performed. Finally, the conclusions are discussed in Section 11.5.

11.1 Literature review

In this section, a review of the literature on the problem under consideration is presented. Since there are heuristics for related scheduling problems that can be adapted to our problem, we also review these contributions. More specifically, we review:

- Heuristics for the (classical) permutation flowshop scheduling problem to minimise makespan.
- Heuristics for the permutation flowshop scheduling problem to minimise total flowtime.
- Heuristics for the blocking flowshop scheduling problem, both with makespan and flowtime objectives.
- Speed-up procedures developed for related flowshop scheduling problems.

Regarding heuristics for the $Fm|prmu|C_{max}$ problem, we focus on the most promising ones and refer the reader to [43], [160] and [172] for more extensive reviews. Among the available heuristics, the NEH heuristic [127] is, without doubt, the most efficient heuristic for the problem. Its excellent performance –established by [172]– probably lies in the low computational cost of carrying out the insertion phases due to the speed-up by [189] (see Section 2.2). Therefore, several papers have focused on improving some of the phases of the NEH, or on proposing NEH-based heuristics. More specifically, improvements in the initial order of the NEH are proposed by e.g. [42] and [201]. Regarding improvements in the insertion phase of the NEH, [150] propose several heuristics (denoted as FRB1, FRB2, FRB3, FRB4_k and FRB5) where a partial insertion local search method after the insertion of a job is employed. In a similar way, [211] employ another partial local search based on the interchange of jobs. Several works address the problem of breaking the ties of the partial makespans when inserting a job, wuch as e.g. [162], [83], [84] and [85], [35].

Regarding the $Fm|prmu|\sum C_j$ problem, the evaluation carried out in Chapter 7 shows that, in terms of average relative percentage deviation and average relative percentage computation time, the set of efficient heuristics is formed by the Raj heuristic by [151]; the LR heuristic proposed by [108]; the RZ heuristic by [152]; the RZ-LW proposed by [97]; the LR-NEH heuristic proposed by [137]; the IC1, IC2 and IC3 improvement heuristics by [96]; and finally, the PR1 heuristic proposed by [137]. As most of these heuristics include the LR heuristic as initial or main procedure, [41] propose an improvement of this method, denoted as FF, which heavily decreases the required CPU time. This procedure has been incorporated in each heuristic that the LR procedure obtaining excellent results for the heuristics: FF, FF-FPE (replacing LR by FF in the LR-FPE heuristic by [108]), FF-ICi (ICi heuristics by [96] with FF instead of LR) and FF-PR1 (PR1 heuristic by [137] using the FF procedure).

Regarding BFSP, several algorithms have been proposed for makespan minimisation. [116] implement a constructive heuristic, denoted as PF, to minimise cycle time, which constructs a sequence inserting progressively an unsequenced job with minimal sum of idle and blocking time. [169] propose three constructive heuristics (denoted as MM, MME and PFE) to solve the $Fm|block|C_{max}$. MME and PFE are variations of the original NEH heuristic where the initial order is replaced by the MM and PF heuristics respectively. In [163], several NEH-based heuristics are proposed using different mechanisms to break ties in the first and second phase of the NEH heuristic. The heuristics are compared with the MME and PFE heuristics. [139] propose eight heuristics (the wPF and PW constructive heuristics and the PF-NEH, wPF-NEH, PW-NEH, PF-NEH$_{\mathrm{LS}}$, wPF-NEH$_{\mathrm{LS}}$ and PW-NEH$_{\mathrm{LS}}$ improvement heuristics) based on NEH and LR. The heuristics clearly outperform MME and PFE in terms of quality of the solution and computational effort. In [164], these improvement heuristics have been improved by evaluating the sequences before and after the insertion phase as well as using the reversibility property.

Regarding the minimisation of total flowtime in the BFSP, [204] introduce an adaptation of the NEH algorithm using the non-decreasing sum of processing times as initial order. Note that this order outperforms the original one for the $Fm|prmu|\sum C_j$ problem. This heuristic is used as initial sequence for the metaheuristics proposed by [5] and [31]. [63] and [62] adapt the MME heuristic to minimise total flowtime as well as proposed two new NEH-based heuristics modifying the initial order (denoted as MME-A and MME-B). Finally, [161] propose 6 new heuristics for the problem. Firstly, they adapt the PF heuristic ([116]) to the problem and propose two new constructive heuristics denoted as HPF1 and HPF2 modifying the index to choose a job. Then, they propose three NEH-based heuristics (NPF, NHPF1 and NHPF2) using the previous heuristics as initial sequences of the NEH.

Finally, regarding speed up methods to accelerate algorithms, they have been successfully applied for several problems related to flowshop scheduling: the PFSP to minimise total flowtime (see e.g. [96]); the PFSP to minimise total tardiness (see e.g. [197]); the PFSP to minimise makespan subject to maximum tardiness (see e.g. [45]); and the distributed PFSP to minimise makespan (see e.g. [124]). However, to the best of our knowledge, they have not been applied to the BFSP so far.

11.2 Problem statement

The problem under study can be stated as follows: a set $\mathcal{N}$ of n jobs have to be scheduled in a flow shop consisting on a set $\mathcal{M}$ of m machines without intermediate buffers. Each machine is always available and can process at most one job at the same time. Following the notation established in Section 11.2. Each job $j \in \mathcal{N}$ has a non preemptive processing time t_{ij} on each machine $i \in \mathcal{M}$. Set up times are sequence-independent and non-anticipatory (see [47]) and thus can be included in the processing times of each job. Let c_{ij} (e_{ij}) represent the departure (start) time of job j from (on) machine i. Note that the departure time of a job must not necessarily be equal to its completion time, as the next machine can block this job after its completion. Similarly, $c_{i[k]}$ represents the departure time of job in position k from machine i. Thereby, when a new job j is placed in the last position of a partial sequence $\Pi_k := (\pi_1, \ldots, \pi_k)$, the departure and the start times of job j can be computed according to Expressions (11.1) and (11.2), respectively. Additionally, let it_j and b_j be the total idle and the total blocking time induced by job j, respectively (see Expressions 11.3 and 11.4).

$$c_{ij} = \begin{cases} c_{i[k]} + t_{ij}, & i = 1 \\ \max\{c_{i-1,j} + t_{ij}, c_{i+1,[k]}\}, & \forall\ i = \{2, \ldots, m-1\} \\ c_{i-1,j} + t_{ij}, & i = m \end{cases} \tag{11.1}$$

$$e_{ij} = \begin{cases} c_{i[k]}, & i = 1 \\ c_{i-1,j}, & \forall\ i = \{2, \ldots, m\} \end{cases} \tag{11.2}$$

$$it_j = \sum_{i=2}^{m} \max\{e_{i-1,j} + t_{i-1,j} - c_{i[k]}, 0\} \tag{11.3}$$

$$b_j = \sum_{i=2}^{m} \max\{c_{i[k]} - (e_{i-1,j} + t_{i-1,j}), 0\} \tag{11.4}$$

As the algorithm proposed in Section is composed of a set of (partial) sequences in each iteration, let us extend the notation and denote by c^k_{ijl} and e^k_{ijl} the departure and start times of job j on machine i of the lth sequence in iteration k, respectively. Analogously, it^k_{jl} and b^k_{jl} represent the total idle and total blocking times of job j on machine i of the lth sequence in iteration k, respectively. Finally, let c^k_{jl} represent the departure time of job j from the last machine m, i.e. $c^k_{jl} := c^k_{mjl}$.

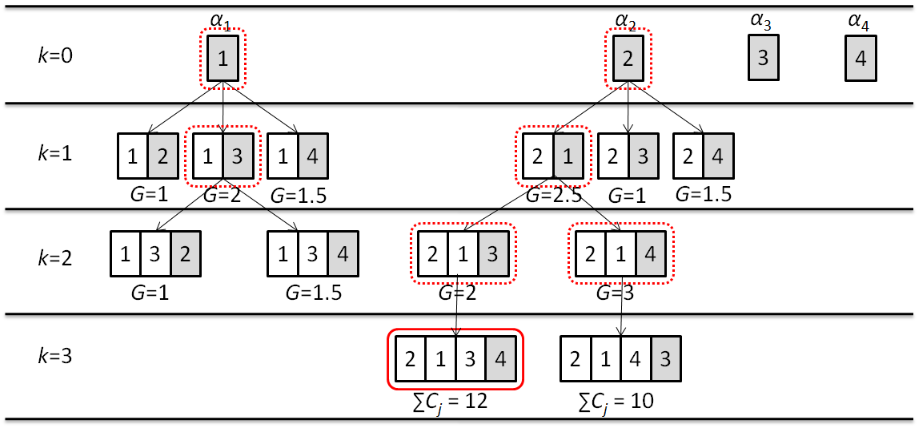

Figure 11.1: Example of the proposed algorithm.

11.3 Proposed heuristic

In this section, a beam-search-based constructive heuristic, BS, is proposed to solve the $Fm|block|\sum C_{ij}$ which successfully combines the diversification of population-based metaheuristics with the speed of constructive heuristics. The algorithm simultaneously constructs several partial sequences in each iteration (denoted as candidate nodes) by appending jobs one by one and keeping the best ones (denoted as selected nodes) over all candidates. A simple example of the algorithm with four jobs is shown in Figure 11.1. More specifically, the algorithm is composed of the following phases:

- Obtain the initial selected nodes
- For n iterations:
  - Construct candidate nodes
  - Evaluate candidates nodes
  - Select the best candidates nodes (selected nodes)

Let us denote by x (beam width) the number of selected nodes in each iteration. At iteration k ($k = 1, \dots, n$), selected node l ($l = 1, \dots, x$) is composed of k sequenced jobs (partial sequence) denoted as $S_l^k := (s_{1l}^k, \dots, s_{kl}^k)$, and a set of $n - k$ unsequenced jobs denoted as $\mathcal{U}_l^k := \{u_{1l}^k, \dots, u_{n-k,l}^k\}$.

First, the algorithm sorts all jobs according to non-decreasing order of indicator ξ_j (see Expression 11.5). Let $\alpha := (\alpha_1, \dots, \alpha_l, \dots, \alpha_n)$ denote this order.

$$\xi_j := \frac{(n-2)}{4} \cdot w_j + \sum_{i=1}^{m} t_{ij}, \; \forall \, j \in [1, n] \tag{11.5}$$

where w_j is the weighted idle time defined by Expression (11.6) (see [108]):

$$w_j := \sum_{i=2}^{m} \frac{m \cdot \sum_{i'=1}^{i-1} t_{i'j}}{i-1}, \ \forall\, j \in [1, n] \tag{11.6}$$

The nodes selected in the first iteration are constructed according to the indicator as follows: the partial sequence $S_l^1 = (s_{1l}^1)$ with $l \in \{1, \ldots, x\}$, is formed by the job in position l of the initial order, i.e. $s_{1l}^1 = \alpha_l$; the $\mathcal{U}_l^1$ set of unsequenced jobs contains all jobs with the exception of the job in S_l^1.

Once the initial selected nodes are obtained, in each iteration k, each selected node l forms $n-k$ candidate nodes for the next generation. Each candidate node $v \in \{1, \ldots, n-x\}$ is constructed from selected node l, appending each job in set $\mathcal{U}_l^k$ at the end of S_l^k. Let $\hat{S}_{vl}^k := (\hat{s}_{1vl}^k, \ldots, \hat{s}_{k+1,v,l}^k)$ and $\hat{\mathcal{U}}_{vl}^k$ be the corresponding partial sequence and set of unsequenced jobs, respectively. Then, the partial sequence of this candidate node and its set of unsequenced jobs are defined by Expression (11.7).

$$\begin{gathered}\hat{S}_{vl}^k = (\hat{s}_{1vl}^k, \ldots, \hat{s}_{k+1,v,l}^k) = (S_l^k, u_{vl}^k) = (s_{1l}^k, \ldots, s_{kl}^k, u_{vl}^k) \\ \hat{\mathcal{U}}_{vl}^k = \{\hat{u}_{1vl}^k, \ldots, \hat{u}_{k+1,v,l}^k\} = \mathcal{U}_l^k - u_{vl}^k\end{gathered} \tag{11.7}$$

Consequently, in iteration k, a total of $x \cdot (n-k)$ candidate nodes are formed. Among these candidate nodes, the best x are selected for the next iteration. Note that, as each new selected node l' in iteration $k+1$ (composed of partial sequence $S_{l'}^{k+1}$, $\forall l' \in \{1, \ldots, x\}$) is formed by adding job u_{vl}^k to selected node l (composed of partial sequence S_l^k), node l' selected in iteration $k+1$ does not have necessarily to come from the partial sequence $S_{l'}^k$ (i.e. l' may be different from l). Therefore, it may happen that one node l is selected in iteration k, but its partial sequence is not selected for the next iteration $(k+1)$. Let $branch[l']$ and $job[l']$ denote the values of l and v respectively for the selected node l and job u_{vl}^k which form selected node l' in iteration $k+1$. In order to select the candidate nodes for the next iteration $(k+1)$, three issues have to be considered to evaluate the candidate node which are typically different for each one:

- Influence of the chosen job, u_{vl}^k, i.e. the last job in the partial sequence $\hat{S}_{vl}^k$ $(\hat{s}_{k+1,v,l}^k)$. Obviously, the departure time of this job on the last machine, $c_{u_{vl}^k l}^k$, has a direct influence on the final objective function. Additionally, the job may incur idle and blocking times which may largely influence the completion times of the subsequent jobs to be inserted. This influence is higher at the beginning of the algorithm when the partial sequence is relative empty and lower in the last iterations where the sequence is almost complete as it affects to a smaller number of jobs (in fact, it does not affect to any job in the last iteration). The index L_{vl}^k (see Expression 11.8), which balances these three objectives, is used to measure the influence of inserting job u_{vl}^k.

$$L_{vl}^{k} = c_{u_{vl}^{k}l}^{k} + a \cdot \frac{n-k-2}{n} \cdot (it_{u_{vl}^{k}l}^{k} + b_{u_{vl}^{k}l}^{k}),$$
$$\forall k = \{1, \dots, n-1\}, v = \{1, \dots, n-k\}, l = \{1, \dots, x\} \tag{11.8}$$

where a is a parameter to balance the influence of the completion time against that of blocking and idle time. $it_{u_{vl}^{k}l}^{k}$ and $b_{u_{vl}^{k}l}^{k}$ are the sum of idle and blocking times between position k (job $\hat{s}_{kvl}^{k}$) and $k+1$ (job $\hat{s}_{k+1,v,l}^{k} = u_{vl}^{k}$) over all machines, respectively. Note that $v_{u_{vl}^{k}l}^{k}$, $it_{u_{vl}^{k}l}^{k}$ and $b_{u_{vl}^{k}l}^{k}$ can be calculated by means of the start time, $e_{iu_{vl}^{k}l}^{k}$, of job u_{vl}^{k} (placed in the last position of the S_{l}^{k} sequence) on machine i and the departure time, $c_{i[k]l}^{k}$, of the previous job (i.e. the job in position k, $\hat{s}_{kvl}^{k}$ or equivalently s_{kl}^{k}) which was already computed in the previous iteration of the algorithm (this fact leads to a high reduction of computational effort since the calculation of the departure times of the complete sequence is avoided).

- Influence of sequenced (previous) jobs, i.e. S_{l}^{k} (or equivalently $\hat{s}_{jvl}^{k}$, $\forall j \leq k$). Due to the process employed to construct the candidate nodes, the first k sequenced jobs of candidate node v may be different to the first jobs of other candidate nodes (e.g. first candidate node is formed by jobs 1 and 2, and second candidate node is formed by jobs 3 and 4). The comparison of these partial sequences is not trivial. Obviously, when the sequences are complete, the algorithm has to look for the minimisation of the total flowtime. However, in case of partial sequence composed of different jobs, several other aspects may have a higher influence. On the one hand, although the goal is the minimisation of total flowtime, a comparison of the partial sequences based only on this measure would obviously be influenced by the characteristics of the jobs of each partial sequence. It would prioritise jobs with low processing times regardless their idle or blocking times. On the other hand, the exclusive consideration of idle and/or blocking times would miss the relation with the objective of the scheduling problem: the minimisation of total flowtime. To cover both aspects, the proposed algorithm uses index F_{l}^{k} (see Expression 11.9) to measure the influence of the sequenced jobs of candidate node v in iteration k. Note that this index is identical for all candidate nodes coming from selected node l since it does not consider the contribution of the last job of the sequence $(\hat{s}_{k+1,v,l}^{k})$. Furthermore, the contribution of idle and blocking times decrease with the number of iterations, thus avoiding their high influence in the last iterations.

$$F_{l'}^{k} = \Delta c_{l'}^{k} + a \cdot (\Delta it_{l'}^{k} + \cdot \Delta b_{l'}^{k}), \; \forall\, k = \{2, \dots, n-1\}, \;\; l^{'} = \{1, \dots, x\} \tag{11.9}$$

where Δt, Δb and Δc are the accumulated idle, blocking and departure time, respectively, defined by the following expressions:

$$\Delta it_{l'}^{k+1} = \Delta it_{branch[l']}^{k} + it_{branch[l'],job[l']}^{k} \cdot \frac{n-k-2}{n},\ \forall\, k = \{1,\dots,n-2\},\ l^{'} = \{1,\dots,x\} \quad (11.10)$$

$$\Delta b_{l'}^{k+1} = \Delta b_{branch[l']}^{k} + b_{branch[l'],job[l']}^{k} \cdot \frac{n-k-2}{n},\ \forall\, k = \{1,\dots,n-2\},\ l^{'} = \{1,\dots,x\} \quad (11.11)$$

$$\Delta c_{l'}^{k+1} = \Delta c_{branch[l']}^{k} + c_{branch[l'],job[l']}^{k} + c_{\lambda,branch[l']}^{k},\ \forall\, k = \{1,\dots,n-2\},\ l^{'} = \{1,\dots,x\} \quad (11.12)$$

where $\Delta it_{l'}^{1} = \Delta c_{l'}^{1} = F_{l'}^{1} = 0,\ \forall\, l^{'} = \{1,\dots,x\}$. $c_{\lambda l}^{k}$ is the departure time of an artificial job placed at the end of the sequence as an estimation of the unscheduled jobs (see the following item).

- Influence of the unsequenced jobs. These are the next jobs to be sequenced in the selected nodes and hence, they also influence the evaluation of the candidate node. However, their impact on the final total flowtime is diffused since they are not scheduled yet. As a measure of its influence, we use an artificial departure time denoted as $c_{\lambda l}^{k}$, which is the departure time of an artificial job λ tested in the last position (position $k+2$) of the sequence (after the last job, u_{vl}^{k} or $\hat{s}_{k+1,v,l}^{k}$). The processing times of this job are equal to the average processing times of all unscheduled jobs of selected node l (i.e. $\mathcal{U}_{l}^{k}$). Note that the chosen job u_{vl}^{k} is also considered to have an artificial departure time. The main reason is that the calculation of this term can be then globally done for all candidate nodes of selected node l, thus decreasing the complexity of the procedure, which is one of the main advantages of the proposed algorithm (see 7 for a more detailed explanation).

Thus, each candidate node v is evaluated using index G_{vl}^{k}, (see Expression 11.13) where the best x values are the nodes selected for the next iteration. The pseudo code of the algorithm is shown in Figure 11.2. The complexity of the algorithm is bounded by the creation and selection of the candidate nodes, which have a complexity of $x \cdot n^2 \cdot m$ and $x^2 \cdot n^2$ respectively. Then the global complexity of the algorithm is $\max\{x \cdot n^2 \cdot m, x^2 \cdot n^2\}$.

$$G_{vl}^{k} = F_{l}^{k} + L_{vl}^{k}, \forall k = \{1,\dots,n-2\}, c = \{1,\dots,n-k\}, l = \{1,\dots,x\} \quad (11.13)$$

Speed up procedure

In this section, we introduce a simple speed up procedure to accelerate the insertion phases of the algorithms. This procedure is based on the speed up methods proposed by [96] and [197]. Basically, the proposed procedure stores the completion times, C_{ij}, of each job j on each machine i before testing a job

```
Procedure BS(x)
    //Initial Order
    Determination of $w_j$ and $\xi_j$, $\forall j \in [1,n]$;
    $\alpha :=$ Jobs ordered according to non-decreasing $\xi_j$ breaking ties in favor of jobs with lower $w_j$;
    Update $S^1_l$ ($s^1_{1l} = \alpha_l$) $\forall l$ and $\mathcal{U}^1_l$ with the remaining jobs.
    $\Delta it^1_l, \Delta c^1_l, F^1_l = 0$, $\forall l \in [1,x]$;
    for $k = 1$ to $n-2$ do
        //Candidate Nodes Creation
        Determination of $it^k_{u^k_{vl}l}$, $b^k_{u^k_{vl}l}$, $c^k_{iu^k_{vl}l}$ ($c^k_{u^k_{vl}l} = c^k_{mu^k_{vl}l}$), $\forall v \in [1,n-k], l \in [1,x]$
        //Candidate Nodes Evaluation
        $G^k_{vl} := F^k_l + c^k_{u^k_{vl}l} + a \cdot (it^k_{u^k_{vl}l} + b^k_{u^k_{vl}l})$, $\forall v \in [1,n-k], l \in [1,x]$
        //Candidate Nodes Selection
        Determination of the $l^{'}$-th best candidate node according to non-decreasing $G^k_{vl}$ in iteration $k$.
        Denote by $branch[l^{'}]$ the value of the index $l$ of that candidate node and by $job[l^{'}]$ the value of $v$,
        $\forall l^{'} \in [1,x]$;
        $c^{k+1}_{i,[k+1],l'} \longleftarrow c_{k,branch[l'],i,job[l']}$
        //Forecasting Phase. Update of the Forecast Index
        for $l^{'} = 1$ to $x$ do
            Update $S^{k+1}_{l'}$ and $\mathcal{U}^{k+1}_{l'}$ by removing job $u^k_{job[l'],branch[l']}$ from $\mathcal{U}^{k+1}_{l'}$ and including in $S^k_{l'}$.
            Determination of $c^{k+1}_{\lambda,branch[l']}$ for new selected node $l^{'}$ formed by the old selected node $branch[l^{'}]$
            with job $job[l^{'}]$. Note that the processing times of the artificial job are equal to the average
            processing times of all unscheduled jobs ($\mathcal{U}^{k+1}_{l'}$);
            $\Delta it^{k+1}_{l'} = \Delta it^k_{branch[l']} + it^k_{job[l'],branch[l']} \cdot \frac{n-k-2}{n}$;
            $\Delta b^{k+1}_{l'} = \Delta b^k_{branch[l']} + b^k_{job[l'],branch[l']} \cdot \frac{n-k-2}{n}$;
            $\Delta c^{k+1}_{l'} = \Delta c^k_{branch[l']} + c^k_{job[l'],branch[l']} + c^k_{\lambda,branch[l']}$;
            $F^{k+1}_{l'} = \Delta c^{k+1}_{l'} + a \cdot (\Delta it^{k+1}_{l'} + \Delta b^{k+1}_{l'})$;
        end
    end
    //Final evaluation
    Evaluate the flowtime of the $x$ selected nodes and return the best one.
end
```

Figure 11.2: BS

in each position. Then, when the job is tested in each position j_1, all completion times C_{ij} with $j < j_1$ stay the same and are not calculated again. Although the complexity of the insertion phase remains the same using this procedure, a strong CPU reduction of about 30%-50% has been achieved for similar procedures in the literature (see e.g. [96]). Note that the procedures proposed by e.g. [189] and [124] cannot be adapted since they are based only on the calculation of the makespan and cannot be applied for the calculation of each completion time on last machine. The proposed speed up procedure has been incorporated in each insertion phase of all implemented heuristics.

11.4 Computational experiments

In this section, a computational evaluation of heuristics is carried out. To perform the comparison we follow the following procedure: a design of experiments is carried out in Section 11.4. In Section 11.4, the implemented heuristics are enumerated. Finally, the computational results of heuristics are shown in Section 11.4.

Experimental parameter tuning

In this section, we perform an experimental tuning of parameter a in the proposed heuristic on set $\mathcal{B}_{C2}$. Regarding the values for the parameter x, we consider $x \in \{2, 5, 15, n/10, n\}$ (see e.g. [108] for similar values of the parameters in other constructive heuristics working with a pool of partial sequences), since this parameter is directly proportional to the CPU time and complexity of the algorithm. The computational experiments for the parameter a are carried out for the proposed BS($x = 5$) and the same value is used for each other value of x. We use the following values for parameter $a \in \{1, 2, 3, ..., 23, 24, 25\}$.

The relationship between the levels of the parameters is evaluated by means of a non-parametric Kruskal-Wallis test since normality and homoscedasticity assumptions are not fulfilled. Note that, the Relative Percentage Deviation $RPD3$ –Expression (9.7)– is used to measure the quality of the solution of the heuristic for each instance.

As a result of the experiments, it turns out that there are statistically significant differences between the levels of the three parameters, since the p-values obtained for the parameters n, m and a are 0.000. The best value found for parameter a is 14, which is used in Section 11.4 in BS(x) $\forall x \in \{2, 5, 15, n/10, n\}$.

Implemented heuristics

In this section, the heuristics included in the computational evaluation are listed. According to the literature review in Section 11.1, 11 heuristics have been published so far for this problem. Additionally, we adapt 8 and 18 heuristics for the $Fm|block|C_{max}$ and for the classical PFSP problem, respectively, given their excellent performance. Finally, the proposed beam-search-based constructive heuristic is added to the comparison. In summary, the heuristics implemented are:

- Heuristics of the $Fm|block|\sum C_i$:
  - Heuristic NEH_WPT: [204].
  - Heuristic MME: [61] (adapted from [169] for $Fm|block|C_{max}$).
  - Heuristic MME-A: [62].
  - Heuristic MME-B: [62] (adapted from [63] for $Fm|block|C_{max}$).
  - Heuristic NEH-MK: [122].
  - Heuristic PF: [161] (adapted from [169] for $Fm|block|C_{max}$).
  - Heuristics HPF1 and HPF2: [161].
  - Heuristics NPF, NHPF1 and NHPF2: [161].
  - Heuristics BS(x), $\forall x \in \{2, 5, 15, n/10, n\}$: Proposed heuristic.
- Heuristics adapted from the $Fm|block|C_{max}$:
  - Heuristics wPF and PW: [139]. These heuristics are implemented as the original ones. For the final sequence, the total flowtime is calculated.
  - Heuristics PF-NEH(x), wPF-NEH(x) and PW-NEH(x), $\forall x \in \{1, 2, 5\}$: [139]. In the NEH-based phase of the algorithms, each evaluation of makespan is replaced by the evaluation of total flowtime. Note that these heuristics include the evaluation of the objective function before applying the NEH-based phase (proposed by [164]). The other improvement proposed by [164] (reversibility property) cannot be applied for total flowtime minimisation.
  - Heuristics PF-NEH$_{\text{LS}}$(x), wPF-NEH$_{\text{LS}}$(x) and PW-NEH$_{\text{LS}}$(x), $\forall x \in \{1, 2, 5\}$: [139]. In both the NEH-based and the local search phases of the algorithms, each evaluation of makespan is replaced by the evaluation of total flowtime.
- Heuristics adapted from the traditional PFSP to minimize total flowtime ($Fm|prmu|\sum C_i$). To adapt the heuristics, each evaluation of the total flowtime of a partial sequence is replaced by the evaluation of the total flowtime with blocking. Note that the indexes of initial sequences and FF and LR-based heuristics are not changed since the objective is the same.
  - Heuristic LR(1): [108].

- Heuristic FF(x), $\forall x \in \{1, 2, n/10, n/m\}$: Section 7.5.
- Heuristic FF-FPE(x, y), $\forall (x, y) \in \{(2, n/10), (15, n/10), (n/10, 1), (n/10, 1), (n/10, n/10), (n/10, n), (n/m, n), (n, n)\}$: [108] with FF($x$) instead of LR($x$) heuristic.
- Heuristics FF-ICH1, FF-ICH2 and FF-ICH3: [96] with FF(x) instead of LR(x) heuristic.
- Heuristic FF-NEH(x) for $x = 5, 10$: [137] with FF(x) instead of LR(x) heuristic.
- Heuristic Raj: [151].
- Heuristic RZ: [152].
- Heuristic RZ_LW: [97].
- Heuristic FF-PR1(x) for $x = [5, 10, 15]$: [137] with FF(x) instead of LR(x) heuristic.

• Heuristics adapted from the traditional PFSP to minimize makespan ($Fm|prmu|C_{max}$). Given a partial sequence, each evaluation of the makespan of this sequence is replaced by the evaluation of total flowtime with blocking:

  - Heuristic NEH proposed by [127].
  - Heuristics FRB2, FRB3, FRB4$_{\mathrm{k}}$ (with $k = [2, 4, 6, 8, 10, 12]$) and FRB5: [150]. Due to the good results found by the NEH_WPT as compared to the original NEH, these heuristics are initialized in a non-decreasing sum of processing times.

Hence, a total of 36 heuristics are compared in this section. Some of them have been executed for different values of the parameters yielding a total of 70 heuristics which are tested. All heuristics are tested under an Intel Core i7-3770 with 3.4 GHz and 16 GB RAM.

Heuristics are evaluated and compared according to the quality of their solutions and their computational efforts. Traditionally, the former is measured by the Average Relative Percentage Deviation, $ARPD1_h$ for heuristic h, while the Average CPU time, $ACPU_h$ for heuristic h, is the indicator used to measure the latter.

Computational evaluation of heuristics

Each implemented heuristic is tested on benchmark $\mathcal{B}_1$. This benchmark is the most common benchmark for the studied problems (see e.g. [204], [61], [63], [62], [165]. Computational results are shown in Table 11.1 in terms of $ARPD1$ (second and fifth columns) and ACPU (third and sixth columns). The best $ARPD1s$ are found by the proposed heuristic BS(x) ($\forall x \in \{5, 15, n/10, n\}$) being 1.239, 0.687, 1.029

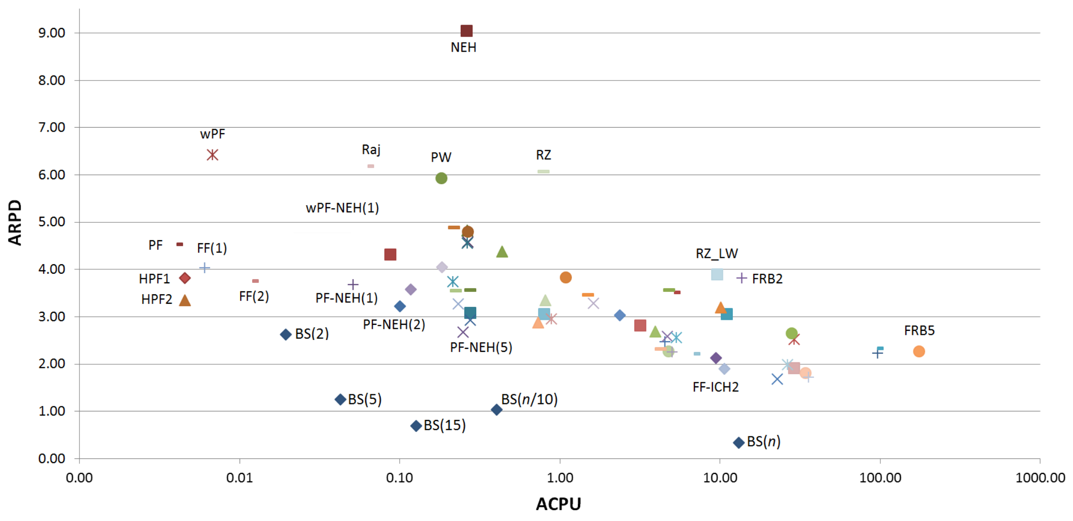

Figure 11.3: $ARPD1$ against ACPU. X-axis (ACPU) is shown in logarithmic scale

and 0.333 respectively. Note the huge distance among the best heuristic (BS(n)) and the best non-proposed heuristics which is 1.682 found by the PF-NEH$_{\text{LS}}$(5) heuristic. Furthermore, the BS(n) needs in average 42.73% lesser CPU time than the PF-NEH$_{\text{LS}}$(5), i.e. the ACPU$_{\text{BS}(n)}$ is 13.148 seconds while the ACPU$_{\text{PF-NEH}_{\text{LS}}(5)}$ is 22.959 seconds. Graphically, the heuristics are shown in Figure 11.3. More detailed results of $ARPD1$ and ACPU for each size for the problem are shown in Table 11.2 and 11.3 respectively. The proposed heuristic BS(x) ($\forall x \in \{2, 5, 15, n\}$) IS efficient as there is no other heuristic with lower ACPU and $ARPD1$. The excellent performance of the proposed heuristic is also highlighted by the 33 the new upper bounds found for the problem.

Regarding heuristics adapted from related decision problems, some of them yield an excellent performance as compared to heuristics specifically implemented for the problem under study. Thereby, e.g. the heuristics PF-NEH(2), FF-FPE(n/10,1) and PF-NEH(5) (with an $ARPD1$ of 3.22, 3.27 and 2.67 respectively) clearly outperform NEH_WPT, MME_A, MME, MME_B and NPF ($ARPD1s$ of 4.82, 4.55, 4.58, 4.80 and 3.56 respectively) using less ACPU. The PF-NEH(5) heuristic even slightly outperforms NHPF1 and NHPF2 with 3.08, and 2.92 of $ARPD1$ respectively. In fact, the best $ARPD1$ among the non-proposed heuristics is found by PF-NEH$_{\text{LS}}$(5), which was originally proposed for the $Fm|block|C_{max}$ problem.

In order to statistically justify the efficiency of the proposed heuristic, we compare it with the best heuristics requiring higher ACPU. We use a Holm's procedure ([73]) to contrast the following hypotheses:

- H_1: BS(5) = PF-NEH(2)
- H_2: BS(15) = FF-ICH2

Table 11.1: *ARPD*1*s* and ACPUs of the implemented heuristics (ordered by increasing ACPU). In bold it is indicated the proposed set of heuristics.

Heuristic	*ARPD*1	ACPU	Heuristic	*ARPD*1	ACPU
PF	4.529	0.004	FF-NEH(5)	3.340	0.813
HPF2	3.349	0.005	FF-FPE(n/10,n/10)	2.945	0.890
HPF1	3.813	0.005	PW-NEH(5)	3.828	1.091
FF(1)	4.028	0.006	FRB4$_2$	3.452	1.509
wPF	6.423	0.007	FF-NEH(10)	3.286	1.623
FF(2)	3.750	0.012	FRB4$_4$	3.025	2.386
BS(2)	**2.614**	**0.019**	FRB4$_8$	2.807	3.205
BS(5)	**1.239**	**0.043**	FRB4$_8$	2.684	3.946
wPF-NEH(1)	4.915	0.044	FF-ICH1	2.313	4.271
PF-NEH(1)	3.670	0.051	PF-NEH$_{\text{LS}}$(1)	2.462	4.548
Raj	6.184	0.063	FRB4$_{10}$	2.584	4.723
wPF-NEH(2)	4.304	0.087	FF-FPE(n/10,n)	2.258	4.776
PF-NEH(2)	3.221	0.101	PW-NEH$_{\text{LS}}$(1)	3.560	4.847
FF(n/m)	3.573	0.117	FF-FPE(n/m,n)	2.250	5.058
BS(15)	**0.687**	**0.127**	wPF-NEH$_{\text{LS}}$(1)	3.508	5.225
PW	5.926	0.182	FRB4$_{12}$	2.558	5.372
LR(1)	4.039	0.184	FF-FPE(n,n)	2.209	6.943
wPF-NEH(5)	3.732	0.216	PF-NEH$_{\text{LS}}$(2)	2.120	9.504
PW-NEH(1)	4.885	0.219	RZ_LW	3.891	9.651
FF(n/10)	3.548	0.224	PW-NEH$_{\text{LS}}$(2)	3.197	10.164
FF-FPE(n/10,1)	3.266	0.234	FF-ICH2	1.896	10.745
PF-NEH(5)	2.669	0.250	wPF-NEH$_{\text{LS}}$(2)	3.056	11.037
NEH	9.043	0.262	**BS(n)**	**0.333**	**13.148**
NEH_WPT	4.816	0.264	FRB2	3.814	13.749
MME_A	4.553	0.266	PF-NEH$_{\text{LS}}$(5)	1.682	22.959
MME	4.576	0.267	FF-PR1(5)	1.978	26.477
MME_B	4.797	0.268	PW-NEH$_{\text{LS}}$(5)	2.647	28.197
NHPF1	3.080	0.277	wPF-NEH$_{\text{LS}}$(5)	2.516	29.102
NHPF2	2.921	0.277	FF-ICH3	1.902	29.157
NPF	3.563	0.277	FF-PR1(10)	1.801	34.279
BS(n/10)	**1.029**	**0.403**	FF-PR1(15)	1.720	35.965
PW-NEH(2)	4.375	0.439	FRB3	2.321	96.546
FF-FPE(15,n/10)	2.879	0.732	NEH-MK	2.229	96.771
RZ	6.066	0.792	FRB5	2.269	176.403
FF-FPE(2,n/10)	3.057	0.801			

Table 11.2: Detailed values of $ARPD1$ for each size of the problem. The proposed set of heuristics is indicated in bold.

Heuristic	Size of the problem (n x m)												$ARPD1$
	20x5	20x10	20x20	50x5	50x10	50x20	100x5	100x10	100x20	200x10	200x20	500x20	
NEH_WPT	2.92	2.84	3.39	5.03	4.31	3.56	6.70	5.31	4.39	6.97	5.36	7.00	4.816
MME	2.88	2.74	2.17	5.16	3.81	3.31	5.94	5.27	3.81	7.08	5.67	7.06	4.576
MME_A	2.76	2.69	2.20	5.02	3.88	3.21	5.90	5.26	4.30	7.03	5.35	7.05	4.553
MME_B	3.39	3.26	3.34	4.89	4.15	3.35	6.40	5.41	4.84	6.69	5.17	6.69	4.797
NEH-MK	1.13	0.92	0.69	2.42	1.51	0.93	3.50	2.42	1.84	4.22	2.82	4.35	2.229
PF	4.78	4.63	4.23	5.26	3.90	4.81	7.80	4.47	3.96	4.16	3.39	2.94	4.529
HPF1	4.04	4.74	3.87	3.12	4.12	4.64	2.92	3.59	4.48	3.26	3.86	3.11	3.813
HPF2	3.71	3.02	3.80	2.99	2.50	3.96	2.73	2.94	4.72	2.94	3.64	3.25	3.349
NPF	2.62	2.49	2.65	4.15	2.94	3.34	6.41	4.18	3.49	4.16	3.39	2.94	3.563
NHPF1	2.60	2.58	2.67	2.89	2.99	3.26	2.86	3.42	3.69	3.15	3.74	3.11	3.080
NHPF2	2.73	2.21	2.58	2.71	2.50	3.14	2.74	2.94	3.80	2.83	3.63	3.25	2.921
BS(2)	1.64	1.90	2.34	2.65	2.25	2.99	2.65	2.51	3.04	2.73	3.50	3.18	2.614
BS(5)	0.50	1.04	1.72	1.17	0.71	1.44	1.25	1.09	1.60	1.33	1.46	1.56	1.239
BS(15)	0.17	1.13	1.64	0.33	0.41	0.94	0.68	0.49	0.41	0.67	0.54	0.84	0.687
BS(n/10)	1.64	1.90	2.34	1.17	0.71	1.44	0.71	0.48	0.74	0.44	0.39	0.39	1.029
BS(n)	0.21	1.09	1.56	0.11	0.12	0.58	0.12	0.03	0.07	0.00	0.09	0.00	0.333
wPF	6.11	4.77	3.58	8.64	7.08	4.06	10.47	7.10	4.18	9.29	5.73	6.07	6.423
PW	7.22	2.73	2.62	9.58	6.11	2.46	10.57	6.60	3.63	8.67	5.14	5.77	5.926
PF-NEH(1)	2.87	2.49	2.73	4.35	2.90	3.87	7.10	4.02	3.75	4.17	2.98	2.81	3.670
PF-NEH(2)	2.45	2.08	2.13	4.29	2.15	2.88	6.55	3.45	3.03	4.02	2.89	2.72	3.221
PF-NEH(5)	1.96	1.45	1.63	3.52	1.97	2.26	5.57	2.43	2.67	3.51	2.62	2.43	2.669
wPF-NEH(1)	2.87	2.22	2.40	6.16	4.98	3.04	8.75	5.40	3.60	8.73	5.18	5.64	4.915
wPF-NEH(2)	2.61	1.92	1.88	5.50	3.58	2.44	8.00	5.26	3.11	7.47	4.53	5.35	4.304
wPF-NEH(5)	1.81	1.74	1.47	4.68	3.06	1.73	7.25	4.86	2.51	6.69	3.86	5.14	3.732
PW-NEH(1)	2.84	2.58	2.86	6.78	4.44	1.89	9.52	5.85	3.29	8.15	4.82	5.61	4.885
PW-NEH(2)	2.62	2.39	1.63	5.61	3.88	1.77	9.25	5.32	2.80	7.68	4.23	5.34	4.375
PW-NEH(5)	1.96	1.73	1.44	5.26	3.09	1.50	7.54	4.97	2.60	6.74	3.90	5.21	3.828
PF-NEH$_{LS}$(1)	1.52	1.08	1.04	3.30	2.04	1.21	5.41	3.17	2.30	3.70	2.37	2.39	2.462
PF-NEH$_{LS}$(2)	1.03	0.89	0.71	3.10	1.30	1.01	4.92	2.58	1.85	3.54	2.22	2.27	2.120
PF-NEH$_{LS}$(5)	0.65	0.47	0.33	2.48	0.97	0.71	4.03	1.88	1.48	3.12	1.95	2.10	1.682
wPF-NEH$_{LS}$(1)	1.78	1.21	0.98	4.59	3.13	1.47	5.52	4.32	2.87	6.53	4.45	5.25	3.508
wPF-NEH$_{LS}$(2)	1.09	1.01	0.66	3.88	2.38	1.22	4.87	4.13	2.38	6.15	4.00	4.93	3.056
wPF-NEH$_{LS}$(5)	0.62	0.49	0.30	3.16	1.97	0.65	4.22	3.64	1.92	5.23	3.31	4.69	2.516
PW-NEH$_{LS}$(1)	1.91	0.81	0.77	5.12	3.29	1.31	6.20	4.34	2.74	6.67	4.24	5.33	3.560
PW-NEH$_{LS}$(2)	1.37	0.68	0.63	4.40	2.63	1.20	6.16	3.98	2.33	6.06	3.82	5.10	3.197
PW-NEH$_{LS}$(5)	0.94	0.48	0.37	3.60	1.93	0.78	4.72	3.40	1.97	5.25	3.49	4.82	2.647
LR(1)	4.30	2.99	2.25	6.00	3.50	2.25	7.13	5.11	2.60	5.59	2.94	3.81	4.039
FF(1)	3.99	2.74	2.34	6.21	3.36	2.38	6.74	5.68	2.63	5.28	3.21	3.77	4.028
FF(2)	3.64	2.63	1.96	5.65	3.30	2.28	6.33	5.23	2.35	4.92	3.10	3.61	3.750
FF(n/10)	3.64	2.63	1.96	5.21	3.14	2.21	6.17	4.43	2.10	4.74	2.95	3.38	3.548
FF(n/m)	3.47	2.63	2.34	5.15	3.14	2.28	6.10	4.43	2.18	4.74	2.97	3.43	3.573
FF-FPE(2,n/10)	2.54	2.04	1.53	4.82	2.91	1.71	5.19	3.83	2.03	4.13	2.78	3.18	3.057
FF-FPE(15,n/10)	2.46	2.04	1.51	4.30	2.72	1.55	4.91	3.46	1.83	3.99	2.70	3.07	2.879
FF-FPE(n/10,1)	3.07	2.19	1.54	4.92	2.98	1.95	5.88	4.05	1.97	4.52	2.86	3.26	3.266
FF-FPE(n/10,n/10)	2.54	2.04	1.53	4.57	2.95	1.56	5.10	3.60	1.83	3.99	2.71	2.93	2.945
FF-FPE(n/10,n)	1.53	1.43	1.10	3.22	1.99	1.28	3.30	2.93	1.65	3.41	2.53	2.73	2.258
FF-FPE(n/m,n)	1.57	1.43	1.12	3.07	1.99	1.27	3.17	2.93	1.70	3.41	2.50	2.84	2.250
FF-FPE(n,n)	1.57	1.43	1.11	3.07	1.91	1.14	3.17	2.88	1.65	3.36	2.51	2.72	2.209
FF-ICH1	1.74	0.96	0.78	3.19	2.03	1.12	4.28	2.85	1.64	3.72	2.51	2.93	2.313
FF-ICH2	1.46	0.76	0.72	2.38	1.64	0.92	2.94	2.39	1.51	2.99	2.27	2.78	1.896
FF-ICH3	1.55	0.76	0.77	2.41	1.74	0.84	2.93	2.21	1.48	3.08	2.29	2.76	1.902
FF-NEH(5)	2.60	2.01	1.60	5.09	2.89	1.93	5.98	4.53	2.16	4.75	2.96	3.57	3.340
FF-NEH(10)	2.60	2.01	1.59	5.02	2.89	1.92	5.93	4.23	2.06	4.69	2.96	3.54	3.286
Raj	4.42	3.20	3.54	6.22	4.91	4.96	9.43	6.62	6.21	9.28	6.52	8.89	6.184
RZ	3.68	2.29	2.01	6.52	5.11	3.46	9.07	7.48	5.40	10.33	7.62	9.83	6.066
RZ_LW	1.48	1.81	0.92	4.12	3.39	2.51	5.67	4.77	3.65	6.35	5.06	6.96	3.891
FF-PR1(5)	0.59	0.65	0.29	2.37	1.35	0.60	4.02	2.98	1.55	3.78	2.36	3.19	1.978
FF-PR1(10)	0.43	0.39	0.22	2.05	1.31	0.42	3.85	2.60	1.38	3.53	2.26	3.17	1.801
FF-PR1(15)	0.37	0.36	0.21	2.05	1.09	0.41	3.57	2.38	1.31	3.51	2.22	3.17	1.720
NEH	6.26	5.27	3.69	11.27	8.53	6.51	12.93	10.74	8.09	13.83	9.87	11.55	9.043
FRB2	1.48	1.75	0.62	4.07	3.07	1.52	6.24	5.06	3.05	7.19	4.92	6.81	3.814
FRB3	1.30	0.88	0.71	2.73	1.58	1.15	3.82	2.57	1.87	4.05	2.88	4.31	2.321
FRB4$_2$	1.95	1.67	1.65	3.40	2.92	1.71	5.27	3.58	3.02	6.13	4.17	5.95	3.452
FRB4$_4$	1.62	1.16	1.13	3.10	2.43	1.73	4.65	3.62	2.47	5.24	3.69	5.46	3.025
FRB4$_8$	1.49	1.23	0.75	2.96	2.37	1.52	4.23	3.07	2.52	4.87	3.46	5.21	2.807
FRB4$_8$	1.31	1.02	0.86	2.90	1.84	1.41	4.32	3.01	2.28	4.84	3.49	4.93	2.684
FRB4$_{10}$	1.30	0.96	0.77	2.93	1.75	1.40	4.22	2.81	2.08	4.49	3.28	5.01	2.584
FRB4$_{12}$	1.30	0.96	0.81	2.74	1.69	1.40	4.13	2.72	2.36	4.44	3.21	4.93	2.558
FRB5	1.25	0.94	0.69	2.58	1.63	0.79	3.87	2.56	1.76	4.13	2.86	4.16	2.269

Table 11.3: Detailed values of CPU times for each size of the problem. The proposed set of heuristics is indicated in bold.

Heuristic	Size of the problem (n x m)												ACPU
	20x5	20x10	20x20	50x5	50x10	50x20	100x5	100x10	100x20	200x10	200x20	500x20	
NEH_WPT	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.07	0.16	2.89	0.264
MME	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.07	0.16	2.91	0.267
MME_A	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.07	0.16	2.90	0.266
MME_B	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.07	0.16	2.92	0.268
NEH-MK	0.00	0.00	0.01	0.03	0.06	0.11	0.39	0.72	1.53	10.39	24.24	1123.77	96.771
PF	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.03	0.004
HPF1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.04	0.005
HPF2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.04	0.005
NPF	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.08	0.17	3.02	0.277
NHPF1	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.08	0.17	3.02	0.277
NHPF2	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.08	0.17	3.02	0.277
BS(2)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.04	0.14	0.019
BS(5)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.05	0.07	0.33	0.043
BS(15)	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.04	0.06	0.12	0.18	1.04	0.127
BS(n/10)	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.03	0.04	0.16	0.25	4.33	0.403
BS(n)	0.00	0.00	0.00	0.04	0.05	0.06	0.28	0.33	0.45	4.10	5.03	147.44	13.148
wPF	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.05	0.007
PW	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.07	0.14	1.92	0.182
PF-NEH(1)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.03	0.07	0.48	0.051
PF-NEH(2)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.05	0.13	0.96	0.101
PF-NEH(5)	0.00	0.00	0.00	0.01	0.01	0.02	0.02	0.03	0.07	0.13	0.32	2.39	0.250
wPF-NEH(1)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.03	0.06	0.41	0.044
wPF-NEH(2)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.05	0.11	0.82	0.087
wPF-NEH(5)	0.00	0.00	0.00	0.01	0.01	0.02	0.02	0.03	0.06	0.12	0.28	2.05	0.216
PW-NEH(1)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.09	0.19	2.27	0.219
PW-NEH(2)	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.06	0.18	0.37	4.58	0.439
PW-NEH(5)	0.00	0.00	0.00	0.01	0.02	0.03	0.04	0.07	0.15	0.44	0.91	11.41	1.091
PF-NEH$_{LS}$(1)	0.00	0.00	0.00	0.01	0.02	0.04	0.06	0.11	0.23	1.00	2.37	50.74	4.548
PF-NEH$_{LS}$(2)	0.00	0.00	0.01	0.02	0.03	0.07	0.11	0.23	0.48	1.84	5.07	106.19	9.504
PF-NEH$_{LS}$(5)	0.00	0.01	0.01	0.03	0.07	0.17	0.28	0.50	1.30	4.50	12.51	256.12	22.959
wPF-NEH$_{LS}$(1)	0.00	0.00	0.00	0.01	0.02	0.04	0.07	0.11	0.25	1.25	2.25	58.69	5.225
wPF-NEH$_{LS}$(2)	0.00	0.00	0.01	0.02	0.03	0.07	0.13	0.21	0.47	2.27	5.03	124.21	11.037
wPF-NEH$_{LS}$(5)	0.00	0.01	0.01	0.04	0.08	0.16	0.34	0.55	1.28	5.75	13.88	327.12	29.102
PW-NEH$_{LS}$(1)	0.00	0.00	0.00	0.01	0.02	0.03	0.08	0.13	0.29	1.25	3.12	53.22	4.847
PW-NEH$_{LS}$(2)	0.00	0.00	0.01	0.02	0.04	0.07	0.15	0.24	0.58	2.72	6.28	111.86	10.164
PW-NEH$_{LS}$(5)	0.00	0.01	0.01	0.04	0.08	0.16	0.37	0.59	1.43	6.35	14.57	314.74	28.197
LR(1)	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.07	0.14	1.95	0.184
FF(1)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.05	0.006
FF(2)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.10	0.012
FF(n/10)	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.08	0.16	2.40	0.224
FF(n/m)	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.08	0.08	1.20	0.117
FF-FPE(2,n/10)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.04	0.14	0.38	9.01	0.801
FF-FPE(15,n/10)	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.06	0.20	0.45	7.99	0.732
FF-FPE(n/10,1)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.09	0.17	2.49	0.234
FF-FPE(n/10,n/10)	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.05	0.22	0.49	9.86	0.890
FF-FPE(n/10,n)	0.00	0.00	0.00	0.01	0.01	0.03	0.07	0.11	0.23	1.04	2.41	53.40	4.776
FF-FPE(n/m,n)	0.00	0.00	0.00	0.01	0.01	0.03	0.08	0.11	0.22	1.04	2.33	56.86	5.058
FF-FPE(n,n)	0.00	0.00	0.00	0.02	0.03	0.05	0.11	0.20	0.40	1.73	3.76	77.00	6.943
FF-ICH1	0.00	0.00	0.00	0.01	0.02	0.03	0.07	0.11	0.25	0.90	1.97	47.88	4.271
FF-ICH2	0.00	0.00	0.01	0.02	0.03	0.06	0.14	0.23	0.46	2.42	5.90	119.68	10.745
FF-ICH3	0.00	0.00	0.01	0.03	0.04	0.08	0.34	0.50	0.82	6.42	10.97	330.67	29.157
FF-NEH(5)	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.03	0.07	0.22	0.50	8.88	0.813
FF-NEH(10)	0.00	0.00	0.00	0.01	0.01	0.02	0.04	0.06	0.13	0.43	1.00	17.76	1.623
Raj	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.04	0.67	0.063
RZ	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.06	0.20	0.47	8.71	0.792
RZ_LW	0.00	0.00	0.00	0.01	0.02	0.04	0.08	0.14	0.41	1.60	4.42	109.08	9.651
FF-PR1(5)	0.00	0.01	0.01	0.04	0.07	0.15	0.26	0.55	1.20	4.97	12.14	298.32	26.477
FF-PR1(10)	0.01	0.01	0.02	0.08	0.14	0.31	0.54	1.13	2.44	10.20	23.93	372.53	34.279
FF-PR1(15)	0.01	0.02	0.03	0.11	0.21	0.47	0.83	1.68	3.81	15.22	36.66	372.54	35.965
NEH	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.07	0.16	2.87	0.262
FRB2	0.00	0.00	0.01	0.01	0.03	0.14	0.06	0.23	0.97	1.67	8.19	153.68	13.749
FRB3	0.00	0.00	0.01	0.03	0.06	0.11	0.38	0.70	1.49	10.25	23.88	1121.64	96.546
FRB4$_2$	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.05	0.11	0.38	0.89	16.60	1.509
FRB4$_4$	0.00	0.00	0.00	0.01	0.01	0.03	0.04	0.08	0.17	0.59	1.41	26.29	2.386
FRB4$_8$	0.00	0.00	0.00	0.01	0.02	0.03	0.06	0.11	0.23	0.77	1.86	35.38	3.205
FRB4$_8$	0.00	0.00	0.00	0.01	0.02	0.04	0.07	0.13	0.28	0.95	2.29	43.56	3.946
FRB4$_{10}$	0.00	0.00	0.01	0.01	0.02	0.04	0.08	0.15	0.32	1.12	2.69	52.23	4.723
FRB4$_{12}$	0.00	0.00	0.01	0.02	0.03	0.05	0.09	0.17	0.37	1.28	3.06	59.39	5.372
FRB5	0.00	0.00	0.01	0.04	0.08	0.17	0.62	1.11	2.61	17.50	41.14	2053.54	176.403

i	H_i	p-value	Mann-Whitney	$\alpha/(k-i+1)$	Holm's Procedure
1	BS(5) = PF-NEH(2)	0.000	R	0.0167	R
2	BS(15) = FF-ICH2	0.000	R	0.0250	R
3	BS(n) = PF-NEH$_{\mathrm{LS}}$(5)	0.000	R	0.0500	R

Table 11.4: Holm's procedure.

Hypothesis	p-value	Mann-Whitney
HPF2 = HFP1	0.043	R
HPF2 = FF(2)	0.129	

Table 11.5: Comparison of HPF2 against HPF1 and FF(2) using a Mann-Whitney non-parametric test.

- H_3: BS(n) = PF-NEH$_{\mathrm{LS}}$(5)

We use a non-parametric Mann-Whitney test assuming a 0.95 confidence level (i.e. $\alpha = 0.05$) to establish the p-value of each hypothesis (see e.g. [138] for similar statistical approach). In Holm's procedure, a hypothesis i among a total of k (ordered in ascending order of p-values) is rejected if its p-value is lower than $\alpha/(k - i + 1)$. The results of the procedure are shown in Table 11.4. Each p-value is 0.000 and therefore, each hypothesis can be rejected.

Regarding the fastest heuristics, i.e. HPF1, HPF2, PF, wPF, FF(1) and FF(2), the best $ARPD1$ is found by HPF2. We perform again a Mann-Whitney test to establish the efficiency of HPF1 using the same confidence. We compare it with both HPF2 and FF(2). Results are shown in Table 11.5. There is no statistical significant difference between HPF2 and FF(2).

11.5 Conclusions

In this chapter, an efficient beam-search-based constructive heuristic is proposed. The heuristic constructs a pool of partial sequences in each iteration appending jobs at the end of the most promising sequences. An index based on the idle, blocking and completion times of the jobs is introduced to determine the selected jobs in each iteration. Thereby, the heuristic adopts a beam-search-based strategy which successfully combines the diversification of population-based algorithms and the speed of constructive heuristics.

The proposed heuristic is compared with the best known constructive and improvement heuristics both for the problem under consideration and for related scheduling problems. A total of 36 heuristics are tested in an exhaustive computational evaluation using the set of instances $\mathcal{B}_1$, where each heuristic has been reimplemented in C# to perform a fair comparison. Additionally, a speed up procedure has been introduced to accelerate the insertion phases of each heuristic. This procedure has been included in each insertion phase if applicable.

Among the implemented heuristics, the best $ARPD1$ are found by the proposed heuristic BS(x)

($\forall x \in \{5, 15, n/10, n\}$). 33 new upper bounds for the well-known Taillard benchmark are found by these heuristics (which means that new best-so-far solutions have been found for more than 27% of these instances). The computational experience also highlights the good performance of several heuristics adapted from related scheduling problems, particularly from the $Fm|block|C_{max}$ problem. This fact may speak for certain correlation between both problems and opens some avenues for further research.

Chapter 12

Parallel PFSP

Although the majority of scheduling problems in the literature assumes that the jobs have to be scheduled in a single factory, the number of companies using this environment is decreasing ([121, 124, 205]). Instead, a multi-factory environment is becoming more and more important, since it reduces production costs and risks while increases the product quality (see e.g. [82]). As a consequence, distributed production scheduling problems dealing with both the allocation of jobs to different factories and their subsequent scheduling has been receiving an increasing attention in the literature (e.g. [79, 78, 13, 14, 26]). Among this type of decision problems, [124] recently presented the so-called distributed permutation flowshop scheduling problem, or DPFSP in the following. In this problem, a set of n jobs has to be processed in one of F identical factories, each one consisting of m machines that all jobs must visit in the same order. The decisions involved in this problem are to simultaneously decide in which factory the jobs have to be processed and which is the sequence of the jobs for each factory. If the objective sought for this problem is the minimisation of the global makespan (i.e. the maximum makespan across the F factories), then the problem can be denoted as $DF|prmu|C_{max}$ (see [124]) following the well-known notation of [58].

The problem under consideration can be seen as a generalization of the well-known Permutation Flowshop Scheduling Problem since, once a set of jobs has been assigned to a factory, the remaining decision problem is a PFSP. Since the latter problem is known to be NP-hard for more than two machines [55], $DF|prmu|C_{max}$ is also NP-hard for $m > 2$. Therefore, researchers have focused on finding methods able to find good –but not necessarily optimal– solutions within a reasonable computational effort. Among them, the works by [124, 53, 54, 205, 105] have provided increasingly better heuristics for the problem.

In this chapter, a new efficient approximate algorithm is proposed for the DPFSP. This algorithm is based on one of the most efficient methods for the PFSP (i.e. the iterated greedy algorithm) and exploits the specific structure of solutions of the problem, thus allowing an up-to 30% reduction of the search space. Furthermore, two new efficient local search methods are embedded in our proposal to improve the so-found solutions. The results prove that the proposed algorithm is very effective, being the best one for each size of the problems in the testbed of [124]. Indeed, new upper bounds are found for 27.6% of the instances in this testbed.

The remainder of the chapter is organized as follows: in Section 12.1 the problem under consideration is described along with its state-of-the-art; in Section 12.2 the proposed algorithm is detailed; in Section 12.3 the algorithm is compared with the rest of existing heuristics in the literature; and, finally, in Section 12.4 the main conclusions are presented.

12.1 Problem statement and state-of-the-art

The problem under consideration can be stated as follows: n jobs have to be scheduled in one of the F flowshop-factories consisting of m machines. Each factory is identical with the same set of m machines and is able to process all jobs. Once a job is assigned to a factory, it has to be processed there without being transferred to another factory. On each machine i, each job j has a processing time denoted as t_{ij} regardless the factory f where the job is processed. The problem determines the sequence π^f, formed by n_f jobs, to be scheduled in each factory f. Therefore, a solution π is formed by the sequence in each factory ($\pi = \left[\pi^1, \ldots, \pi^f, \ldots, \pi^F\right]$). Let us $C^f_{i,j}$ be the completion time of job j in machine i when assigned to factory f, and $C^f_{max} = C_{max}(\pi^f)$ the makespan of factory f. Then $C_{max} = C_{max}(\pi)$ denotes the global makespan. i.e. the completion time of the last job to be processed in any factory. Additionally, $\pi^f[i]$ is employed to denote the element of factory f in position i. By using f_{max} to denote the factory with maximum makespan, the global makespan can be also written as $C^{f_{max}}_{max}$.

On one hand, the DPFSP can be seen as a special case of the distributed assembly permutation flowshop scheduling problem (DAPFSP), see [69]. When each factory is formed by exactly two machines, the problem has also been studied in the literature under the name of *Parallel Flowline* [196] or *Parallel Flowshops* [9]. In this special case, the problem turns to be a pure assignment problem, since Johnson's rule [80] can be applied to find the optimal sequence for each shop (examples of heuristics can be found in [219, 3]). On the other hand, it has been already mentioned that it is a generalization of the PFSP, which is one of the main combinatorial optimization problems [220] and in consequence one of the most studied scheduling problems [138]. As mentioned before, the PFSP was proved to be NP-complete for more than three machines. In order to provide efficient approximate methods for this problem, numerous contributions have been presented in the literature (see Chapter 4.2).

The first heuristics to solve the DPFSP were proposed by [124]. They suggested four constructive heuristics, denoted NEH1, NEH2, VND(a) and VND(b), following the ideas taken from the PFSP and employing Taillard's acceleration. More specifically, NEH1 and NEH2 are adaptations to the problem of the original NEH by means of using two assignment rules to choose the factory where a job has to be introduced were tried, i.e.:

1. To allocate the job to the factory with the lowest current makespan (included in the NEH1 heuristic).
2. To allocate the job to the factory that can process it at the earliest time (used in NEH2 algorithm).

On the other hand, VND(a) and VND(b) are composed of a first step in which the NEH2 heuristic is performed, and then its solution is improved by means of a simple variable neighborhood descent consisting of two different neighborhoods and two different acceptance criteria, one for VND(a) and another for VND(b).

Using NEH2 and VND(a) as initial solutions, [53] were the first authors who proposed an iterated optimization algorithm for the problem. More specifically, they presented a genetic algorithm with a local search phase including mechanisms of exchange and insertion of jobs. Their proposal was later outperformed by the tabu search algorithm (TS) by [54], in which partial sequences of two different factories were iteratively exchanged and improved by means of several enhanced local search based also on methods of exchange and insertion of jobs. Next, [205] implemented an Estimation of Distribution Algorithm (EDA), although their proposal was not compared with the tabu search algorithm from [54], arguing that no results were listed for direct comparisons. Finally, [105] proposed for the problem a variation of the iterated greedy, denoted MIG, in which the size of the destruction was randomly chosen between two bounds and the temperature of a simulated annealing-like acceptance criterion decreased with the iterations. However, they do not include any local search phase in their algorithm to improve the solution. Their algorithm was compared with algorithms implemented in [53, 54] concluding that the MIG is the most efficient. However, the CPU times for each instance of the testbed were different for each heuristic under comparison so there is up-to-now no comparison of the state-of-the-art algorithms under the same exact conditions.

To summarise the state-of-the-art in the DPFSP problem, there are two types of algorithms available:

1. Very fast heuristics, i.e. NEH1, NEH2, VND(a) and VND(b). They are simple or composite heuristics (see notation of [49]) where the computational time is non controllable by the decision-maker and the solutions can be quickly found even for large size of the problem.
2. Iterative improvement algorithms, i.e. TS, EDA, and MIG. The improvement phase of these algorithms is performed iteratively improving the solutions –usually based on any of the aforementioned

very fast heuristics– at the expense of substantially increasing the computation times. To the best of our knowledge, there are no direct comparison among these algorithms under the same conditions in the literature.

In this chapter, we focus in the second type of algorithms. Thereby, we first implement these algorithms (MIG, TS and EDA) using the same computer and programming language and reporting the results in Section 12.3. By doing so, a direct comparison among these algorithms is provided. Additionally, in Section 12.2, we propose a new heuristic for the DPFSP that uses the specific structure of solutions of the problem to reduce the search space and that improves the results obtained by existing algorithms, as we show in a subsequent computational experience in Section 12.3.

12.2 The proposed bounded-search iterated greedy algorithm

In this chapter we propose an algorithm labeled Bounded-Search Iterated Greedy (BSIG) which can be seen as a special case of the Iterated Greedy (IG) algorithm [174]. The IG starts with an initial solution and then iteratively applies four steps. First, a number of jobs are taken out of the sequence. Then these jobs are again inserted in the sequence, one by one, following a greedy procedure until no more job has to be inserted. After that, a local search mechanism is performed in order to improve the solution. Finally, a simulated annealing procedure decides if the actual sequence is kept as iteration sequence. IG is one of the best algorithms for the PFSP (see e.g. [138]) and it has been successfully applied to a variety of scheduling problem, such as the sequence dependent setup times problem [175], flowshop scheduling problem with blocking [163], unrelated parallel machine scheduling [39], or the no-wait flowshop scheduling problem [140].

BSIG also iteratively constructs and destructs a solution trying to improve it in each iteration by means of three local search phases. In order to reduce the search space, two properties are derived in Subsection 12.2. In Subsection 12.2, the main procedure of BSIG is described. Its construction phase is detailed in Subsection 12.2, while in Subsections 12.2 and 12.2, two new local search phases are described. Finally, in Section 12.2 a full factorial design of experiment is carried out in order to obtain the best values of the parameters of the algorithm.

Problem properties

It is clear that, when a job is assigned to a specific factory in the DPFSP, the makespan of this factory is increased by a certain value. The following properties serve us to define lower and upper bounds for this makespan:

Property 12.2.1. ***(Lower bound on makespan):*** *The makespan of a factory with m machines must increase at least $min_i\,(t_{i,l})\,\forall t_{i,l} > 0$ with $i \in [1, \cdots, m]$ when a new job l is assigned to this factory, regardless its specific position in the schedule of this factory.*

Proof. The proof of this property is obvious: Before assigning job l in position k of the sequence, there was a set H of machines (with $|H| \geq 1$ being at least one in the first machine) where there was no idle time between positions k-1 and k (worst case). Hence, the new completion times of the jobs placed after position k are increased at least the processing time $t_{h,l}$ in each machine $h \in H$ and at least $min_{h \in H}\,(t_{h,l})$ in the rest of machines, with $min_{h \in H}\,(t_{h,l}) \geq min_{i \in [1,\cdots,m]}\,(t_{i,l})$. This fact proves that the makespan of the factory after introducing the job increases at least the minimum processing time of the job l. □

Property 12.2.2. ***(Upper bound of makespan):*** *The makespan of a factory with $n-1$ jobs and with m machines must increase at most $\sum_{i=1}^{m} t_{i,l}$ with $i \in [1, \cdots, m]$ when a new job l is assigned to this factory, regardless its specific position in the schedule of the factory.*

Proof. The proof of the property is obvious using the same reasoning as in Property 12.2.1. □

Both properties can be useful for the DPFSP and have been taken into account in the design of BSIG. Since BSIG includes iterations where a job has to be assigned to one of the f factories, there are f possible options. The idea is to discard the options where its lower bound (according to Property 12.2.1) is higher than the current best value of the objective function. The effect in the reduction of the search space of Property 12.2.1 will be explained in detail in Section 12.3. However, Property 12.2.2 was not found to be significant for the algorithm and it has not been incorporated in the BSIG. Note that more sophisticated (and tighter) lower bounds for the makespan of each factory have been tested, however they have not resulted in a significant improvement of the objective function due to the increase in the CPU time when the lower bound is calculated. In contrast, the simple lower bound of Property 12.2.1 can be calculated without increasing the complexity of the algorithm, since the minimum processing time of each job is obtained at the beginning of the algorithm.

Main procedure

The main procedure of the proposed algorithm is summarised in Figure 12.1. It starts with the implementation of the fast constructive heuristic NEH2 of [124]. Once an initial solution is so-obtained, it is improved by using three different local search methods (LS1, RLS1 and RLS2) before entering in the iterated procedure. The local search method LS1 was presented in [124] and is a simple iterative improvement algorithm for each factory using a first-improvement type pivoting rule. This local search method has been successfully applied in numerous algorithms for PFSP to minimize makespan and total flowtime (see e.g. [174, 96]). In Figure 12.2 the pseudo code of this method is shown.

When the solution cannot be further improved, the algorithm begins an iterative procedure composed of the following four phases, which are repeated until the stopping criteria is reached:

- Destruction phase: This phase tries to perturb the solution and –together with the simulated annealing phase– its objective is to provide diversification of solutions. In this phase, d jobs are randomly chosen to be removed of the sequence without repetition forming a new sequence denoted π_1.

- Construction phase: Each one of the aforementioned d jobs is inserted, one by one, in the sequence following a greedy procedure (`ConstructionFunction`, see subsection 12.2) using Taillard's acceleration and Property 12.2.1 , i.e. when a job is to be inserted, there are f factories where the job can be assigned. Property 12.2.1 is used to discard factories in which the insertion of the job does not improve the current makespan.

- Improvement phase: It serves to implement the intensification of the algorithm. The sequence constructed in the last phase is improved using three different local search methods: LS1, RLS1 and RLS2. The complexity of these local search methods are $n^2 \cdot m/F$, $n^2 \cdot m/F$ and $n^3 \cdot m/F^2$ respectively being the RLS2 the procedure with highest complexity. This fact may cause a non-efficient behaviour of the heuristic when n is very high in comparison with F since it could lead to a low diversification. Therefore, RLS2 is used only when n/F is lower or equal than a parameter L. By doing so, we get a total complexity of $n^2 \cdot m/F$ for the local search phase of the algorithm.

- Simulated annealing phase: A simple simulated annealing criterion is introduced in the algorithm with a constant *Temperature* which is a function of a parameter T of the algorithm:

$$Temperature = T \cdot \frac{\sum_{\forall i} \sum_{\forall j} t_{i,j}}{n \cdot m \cdot 10}$$

ConstructionFunction

As Taillard's acceleration allows performing the insertion phase with low complexity, π_d (the destructed jobs) are inserted in the sequence using a similar procedure as in NEH2 (see `ConstructionFunction` in Figure 12.3). The d destructed jobs are introduced, one by one, in the position of sequence π_1 which minimises the makespan. If we denote by C_{max}^{ref} the reference makespan or best-known makespan, then it is clear that it makes sense to assign the job to a factory only if its lower bound (calculated according

```
$\pi$ := NEH2(decreasing sum of processing times) ;
for $f = 1$ to $F$ do
    $\pi^f$ := LS1($\pi^f$) ;
end
$flag$ := true;
while $flag$ do
    $\pi$ := RLS1($\pi$) ;
    if solution improved then
        $flag$ := false;
    end
end
if $n/F \leq L$ then
    $flag$ := true;
    while $flag$ do
        $\pi$ := RLS2($\pi$) ;
        if solution improved then
            $flag$ := false;
        end
    end
end
$\pi_b := \pi$;
while stopping criterion is not reach do
    $\pi_1 := \pi$;
    for $i = 1$ to $d$ do
        $\pi_1$ := randomly remove a job from $\pi_1$ and insert it in $\pi_D$;
    end
    $\pi_2 := ConstructionFunction(\pi_1, \pi_D)$
    for $f = 1$ to $F$ do
        $\pi_2^f$ := LS1($\pi_2^f$) ;
    end
    $\pi_3$ := RLS1($\pi_2$) ;
    if $n/F \leq L$ then
        $\pi_3$ := RLS2($\pi_3$) ;
    end
    if $C_{max}(\pi_3) < C_{max}(\pi)$ then
        $\pi := \pi_3$
        if $C_{max}(\pi_3) < C_{max}(\pi_b)$ then
            $\pi_b := \pi_3$
        end
    else if $random \leq exp\{-(C_{max}(\pi_3) - C_{max}(\pi))/Temperature\}$ then
        $\pi := \pi_3$
    end
end
return $\pi_b$
```

Figure 12.1: Main Procedure of the BSIG

```
$flag$ := true;
while $flag$ do
    $flag$ := false;
    for $f = 1$ to $F$ do
        $C_{max}^{ref} = C_{max}(\pi^f)$;
        Remove job $\pi^f[i]$ placed in position $i$ of the factory $f$.
        Test job $\pi^f[i]$ in any possible position of $\pi^f$ (using Taillard's accelerations) and place it in the
        position with the lowest makespan
        if $C_{max}(\pi^f) < C_{max}^{ref}$ then
            $flag$ := true;
            break;
        end
    end
end
```

Figure 12.2: Local Search LS1

to Property 12.2.1) is lower than the reference makespan. This mechanism serves to decrease the number of factories where the jobs are tried (bounded search mechanism). The procedure of this bounded search in each iteration is relative simple: First the job π_d is tried to be placed in the first factory and the best makespan is kept as C_{max}^{ref}. Secondly, the job is tried to be assigned to factories f which satisfy $C_{max}^{f} + min_i\,(t_{i,\pi_d}) < C_{max}^{ref}$. C_{max}^{ref} is then updated when the new makespan (due to the insertion of job π_d in factory f) improves the actual C_{max}^{ref}.

Simple relative local search, RLS1

The relative local search RLS1 searches for better solutions by inserting jobs from one factory to another. More specifically, each job of the factory with maximum makespan is tried to be scheduled in all positions of each factory verifying Property 12.2.1. If the global makespan improves, then the job is scheduled in the factory that minimises the makespan. The procedure then restarts first by looking for the new factory with maximum makespan until each job assigned to the factory with maximum makespan is tried without solution improvement. The pseudo code of RSL1 is shown in Figure 12.4.

Relative local search based in exchange, RLS2

The relative local search RLS2 exchanges jobs between factories. Thereby, each job of the factory with the maximum makespan is tried to be exchanged with each job of the rest of the factories. In order to apply Taillard's acceleration, each of both jobs is removed of the sequences of the factories and inserted in the other factory looking for the best position there. This procedure is repeated for each pair of jobs of both factories choosing the pair of jobs and positions minimizing the makespan of the chosen factories. If the new maximum makespan of both factories is lower than in the last iteration, the procedure begins again

```
Function ConstructionFunction($\pi, \pi_D$)
    for $d = 1$ to $D$ do
        Test job $\pi_D[d]$ in any possible position of $\pi^{f=1}$ (using Taillard's accelerations) and denote $C_{max}^{ref}$ the best makespan.
        $t_{min}$ := minimum processing time of job $\pi_D[d]$ in any machine $i$;
        for $f = 2$ to $F$ do
            $C_{max}^{f}$ := makespan of the factory $f$;
            if $C_{max}^{f} + t_{min} < C_{max}^{ref}$ then
                Test job $\pi_D[d]$ in any possible position of the factory $f$ (using Taillard's accelerations) and denote $C_{max}^{0}$ the best makespan.
                if $C_{max}^{0} < C_{max}^{ref}$ then
                    $C_{max}^{ref} = C_{max}^{0}$;
                end
            end
        end
        $\pi$ := permutation obtained by inserting $\pi_D[d]$ in the factory and in the position with less makespan;
    end
    return $\pi$;
end
```

Figure 12.3: Procedure ConstructionFunction

```
Procedure RLS1($\pi$)
    $h(f) = 1$, for $f = 1 \cdots F$;
    $i(f) = 1$, for $f = 1 \cdots F$;
    $\pi_b := \pi$ being $\pi = [\pi^1, \cdots, \pi^f, \cdots, \pi^F]$;
    Determine the factory $f_{max}$ with maximum makespan ($C'_{max}$)
    while $i(f_{max}) < n_{f_{max}}$ do
        $j := h(f_{max}) \bmod n_{f_{max}}$;
        $\pi_0$ := remove job $\pi^{f_{max}}[j]$ from $\pi^{f_{max}}$;
        $C_{max}^{f_{max}}$ := makespan of the sequence $\pi_0$;
        $t_{min}$ := minimum processing time of job $\pi^{f_{max}}[j]$ in any machine $i$;
        for $f = 1$ to $F$ do
            $C_{max}^{f}$ := makespan in the factory $f$;
            if $C_{max}^{f} + t_{min} < C'_{max}$ then
                Test job $\pi^{f_{max}}[j]$ in each position of the factory $f$ (using Taillard's accelerations)
            end
        end
        $\pi$ := permutation obtained by inserting $\pi^{f_{max}}[j]$ in the factory and in the position with less makespan;
        Determine the factory $f_{max}$ with maximum makespan ($C_{max}$)
        if $C_{max} < C'_{max}$ then
            $C'_{max} = C_{max}$;
            $i(f_{max}) = 1$;
            $\pi_b := \pi$;
        else
            $i(f_{max}) + +$;
        end
        $h(f_{max}) + +$;
    end
    return $\pi_b$;
end
```

Figure 12.4: Relative Local Search RLS1

Source	Df	Chi-Square	Sig.
Parameter d	4	43.453	0.000
Parameter T	4	4.353	0.360
Parameter L	3	5.717	0.126

Table 12.1: Kruskal-Wallis for the parameters d, L and T

by the factory with maximum makespan. The procedure stops when each job of the factory of maximum makespan is tested without improving the solution. The pseudo code of this relative local search is shown in Figure 12.5.

Experimental parameter tuning

The proposed algorithm is composed of three parameters T, d, L. In order to find the best values of the parameters, a full factorial design of experiment is performed for the BSIG with the following level of the parameters:

- $T \in [0.1, 0.2, 0.3, 0.4, 0.5]$
- $d \in [3, 4, 5, 6, 7]$
- $L \in [15, 20, 25, 30]$

The BSIG is evaluated by means of $RPD3$, see Expression (9.7), where $Base$ is the solution obtained by an alternative algorithm (VNDa).

Each combination of parameters has been tested for 96 combination of n, m and F, i.e. $n \in [20, 50, 100, 200]$, $m \in [5, 10, 15, 20]$ and $F \in [2, 3, 4, 5, 6, 7]$ using 5 instances for each one representing a total of 480 instances where the processing times of each job in each machine was uniformly distributed between 1 and 99. Note that we perform 5 runs of each instance due to the randomness of the algorithm. Each replicate is stopped when the CPU time reaches a limit of $n \cdot m \cdot F \cdot 1.5$ milliseconds. Since the normality and homoscedasticity assumptions were not confirmed, a non-parametric Kruskal-Wallis analysis is used to determine statistically difference between the parameters. A summary of the results is shown in Table 12.1. It can be observed that there are statistically significant differences only between the levels of parameter d, being the level of all other parameters non-statistically significant. Additionally, the analysis reveals that the best values for the parameters were found for $d = 5$, $L = 20$ and $T = 0.4$, so these are the values used in the subsequent computational experience.

```
Procedure RLS2($\pi$)
    $h(f) = 1$, for $f = 1 \cdots F$;
    $i(f) = 1$, for $f = 1 \cdots F$;
    $\pi_b := \pi$ being $\pi = [\pi^1, \cdots, \pi^f, \cdots, \pi^F]$;
    Determine the factory $f_{max}$ with maximum makespan $(C'_{max})$
    while $i(f_{max}) < n_{f_{max}}$ do
        $C^{aux}_{max} = C'_{max}$;
        $flag :=$ false;
        $j := h(f_{max})$ mod $n_{f_{max}}$;
        for $f = 1$ to $F$ do
            for $g = 1$ to $n_f$ do
                if $f \neq f_{max}$ then
                    $\pi_0 :=$ remove job $\pi^{f_{max}}[j]$ from $\pi^{f_{max}}$;
                    $\pi_1 :=$ remove job $\pi^f[g]$ from $\pi^f$;
                    Best makespan $C^1_{max}$ due to testing job $\pi^{f_{max}}[j]$ in each position of $\pi_1$ denoting $Pos_f$ the
                    chosen position (using Taillard's acceleration).
                    Best makespan $C^0_{max}$ due to testing job $\pi^f[g]$ in each position of $\pi_0$ denoting $Pos_{f_{max}}$ the
                    chosen position (using Taillard's acceleration).
                    if $C^1_{max} < C^{aux}_{max} \& C^0_{max} < C^{aux}_{max}$ then
                        $flag :=$ true;
                        $BestPos_{f_{max}} := Pos_{f_{max}}$;
                        $BestPos_f := Pos_f$;
                        $chosen_g := g$;
                        $chosen_f := f$;
                        $C^{aux}_{max} = max\left(C^0_{max}, C^1_{max}\right)$;
                    end
                end
            end
        end
        if $flag$ then
            $\pi^{f_{max}} :=$ permutation obtained by inserting $\pi^{chosen_f}[chosen_g]$ in the factory $f_{max}$ and in the position
            $BestPos_{f_{max}}$;
            $\pi^{chosen_f} :=$ permutation obtained by inserting $\pi^{f_{max}}[j]$ in the factory $chosen_f$ and in the position
            $BestPos_f$;
            Update $\pi$ with $\pi^{f_{max}}$ and $\pi^{chosen_f}$;
            Determine the factory $f_{max}$ with maximum makespan $(C_{max})$
        end
        if $C_{max} < C'_{max}$ then
            $C'_{max} = C_{max}$;
            $i(f_{max}) = 1$;
            $\pi_b := \pi$;
        else
            $i(f_{max}) + +$;
        end
        $h(f_{max}) + +$;
    end
    return $\pi_b$;
end
```

Figure 12.5: Relative Local Search with interchange RLS2

12.3 Computational evaluation

The proposed BSIG is compared with the best available algorithms for the DPFSP: MIG, EDA and TS. In order to define the most efficient algorithm, the same computer conditions have to be used using a PC with 2.80 GHz Intel Core i7-930 processor and 16 GBytes of RAM memory. The algorithms are evaluated for all instances presented by [124] which are available in http://soa.iti.es. A total of 720 instances are included in this testbed varying the number of jobs, machines and factories according to the following values $n \in [20, 50, 100, 200, 500]$, $m \in [5, 10, 15, 20]$ and $F \in [2, 3, 4, 5, 6, 7]$ and using 10 instances for each combination of parameters. In order to increase the power of the analysis, 5 runs have been performed per instance for each algorithm. Regarding the stopping criteria, we have used three different stopping criteria based on computation time for the heuristics: $n \cdot m \cdot F \cdot 0.5$, $n \cdot m \cdot F \cdot 1$ and $n \cdot m \cdot F \cdot 2$ milliseconds (see e.g. [174, 105] for similar stopping criteria in the literature). Thereby, each heuristic has been stopped when the computation time reaches these values.

The performance of the algorithms is again evaluated by means of $ARPD2$, where UB is the best known solution taken from `http://soa.iti.es`.

The $ARPD2$ values are shown in Table 12.2 and 12.3 for the stopping criterion $n{\cdot}m{\cdot}F{\cdot}0.5$ milliseconds, in Tables 12.4 and 12.5 for the stopping criterion $n \cdot m \cdot F \cdot 1$ and, finally, in Tables 12.6 and 12.7 for for $n \cdot m \cdot F \cdot 2$ milliseconds yielding e.g 1.43 for the BSIG heuristic, 3.11 for TS algorithm, 6.95 for the EDA and 2.09 for the MIG heuristic according to the first stopping criterion. The results show that the BSIG heuristic outperforms the rest of the heuristics for all stopping criteria. In fact, BSIG outperforms MIG, TS and EDA for each size of the problem regardless the stopping criterion. Additionally, the BSIG algorithm finds the best solution for 93.0% instances, while MIG, TS and EDA found the best solution for 1.9%, 3.0% and 12.6%, respectively. Comparing the results with the best known solution for the largest CPU time, in 263 of the 720 instances (36.53%) new best solutions are found by BSIG. In contrast, only 0 new best solutions were found for TS, 38 instances for MIG and 0 for EDA.

Additionally, a paired samples t-test is carried out in order to compare the heuristics for each stopping criterion. These tests can be applied since the random variables ($ARPD2$) are related (the same test bed is used for each algorithm) and the hypothesis of independence can be rejected. The results of the tests (see Tables 12.8, 12.9 and 12.10) show that BSIG statistically improves each other algorithm being the maximum p-value 0.000.

n x m	BSIG	TS	EDA	MIG
20 x 5	4.94	8.75	5.99	5.34
20 x 10	4.28	7.14	5.17	4.44
20 x 20	3.69	5.78	4.28	3.80
50 x 5	0.50	2.63	4.77	1.64
50 x 10	0.92	3.46	5.48	1.73
50 x 20	0.85	3.01	4.59	1.32
100 x 5	0.14	0.90	5.81	1.02
100 x 10	0.40	1.59	7.76	1.41
100 x 20	0.62	1.66	7.22	1.37
200 x 10	0.23	0.69	9.58	0.93
200 x 20	0.44	0.97	10.27	1.18
500 x 20	0.19	0.70	12.44	0.92
Average	1.43	3.11	6.95	2.09

Table 12.2: $ARPD2$ (by n and m) of the heuristics BSIG, TS, MIG and EDA for the stopping criteria of $n \cdot m \cdot F \cdot 0.5$ milliseconds

f	BSIG	TS	EDA	MIG
2	1.45	2.35	6.27	1.78
3	1.22	2.52	6.57	1.80
4	1.12	2.84	6.71	1.82
5	1.12	3.07	6.86	1.85
6	1.47	3.49	7.26	2.27
7	2.25	4.36	8.01	3.03
Average	1.43	3.11	6.95	2.09

Table 12.3: $ARPD2$ (by f) of the heuristics BSIG, TS, MIG and EDA for the stopping criteria of $n \cdot m \cdot F \cdot 0.5$ milliseconds.

n x m	BSIG	TS	EDA	MIG
20 x 5	4.80	8.67	5.66	5.08
20 x 10	4.18	7.20	4.93	4.29
20 x 20	3.60	5.81	4.19	3.69
50 x 5	0.21	2.64	3.70	0.97
50 x 10	0.55	3.53	4.72	1.10
50 x 20	0.55	3.06	3.96	0.89
100 x 5	-0.01	0.93	4.84	0.46
100 x 10	0.11	1.56	6.83	0.72
100 x 20	0.32	1.69	6.55	0.71
200 x 10	-0.02	0.62	8.69	0.33
200 x 20	0.14	0.89	9.48	0.43
500 x 20	-0.09	0.54	11.95	0.23
Average	1.20	3.10	6.29	1.58

Table 12.4: $ARPD2$ (by n and m) of the heuristics BSIG, TS, MIG and EDA for the stopping criteria of $n \cdot m \cdot F \cdot 1$ milliseconds

f	BSIG	TS	EDA	MIG
2	1.16	2.34	5.63	1.45
3	0.95	2.50	5.83	1.34
4	0.88	2.79	6.00	1.28
5	0.87	3.04	6.23	1.31
6	1.27	3.50	6.64	1.67
7	2.05	4.40	7.43	2.41
Average	1.20	3.10	6.29	1.58

Table 12.5: $ARPD2$ (by f) of the heuristics BSIG, TS, MIG and EDA for the stopping criteria of $n{\cdot}m{\cdot}F{\cdot}1$ milliseconds.

n x m	BSIG	TS	EDA	MIG
20 x 5	4.72	8.74	5.55	4.92
20 x 10	4.11	7.16	4.88	4.19
20 x 20	3.57	5.76	4.19	3.64
50 x 5	0.00	2.65	2.99	0.43
50 x 10	0.26	3.50	4.01	0.70
50 x 20	0.31	3.03	3.45	0.60
100 x 5	-0.18	0.89	3.92	0.10
100 x 10	-0.14	1.61	6.04	0.20
100 x 20	0.01	1.66	5.99	0.24
200 x 10	-0.22	0.66	7.81	-0.01
200 x 20	-0.12	0.88	8.86	-0.02
500 x 20	-0.34	0.43	11.44	-0.13
Average	1.00	3.08	5.76	1.24

Table 12.6: $ARPD2$ (by n and m) of the heuristics BSIG, TS, MIG and EDA for the stopping criteria of $n \cdot m \cdot F \cdot 2$ milliseconds

f	BSIG	TS	EDA	MIG
2	0.96	2.30	4.96	1.15
3	0.74	2.50	5.26	0.98
4	0.65	2.74	5.46	0.94
5	0.67	3.05	5.72	0.94
6	1.06	3.52	6.18	1.32
7	1.89	4.39	6.99	2.09
Average	1.00	3.08	5.76	1.24

Table 12.7: $ARPD2$ (by f) of the heuristics BSIG, TS, MIG and EDA for the stopping criteria of $n{\cdot}m{\cdot}F{\cdot}2$ milliseconds.

Algorithm	Mean	SEM	IC - Lower	IC - Upper	t	Significance
TS vs BSIG	1.672	1.525	1.561	1.784	29.452	0.000
EDA vs BSIG	5.511	3.715	5.240	5.783	39.839	0.000
MIG vs BSIG	0.656	0.457	0.623	0.690	38.548	0.000

Table 12.8: Paired samples t-test for stopping criterion of $n \cdot m \cdot F \cdot 0.5$ milliseconds.

Algorithm	Mean	SEM	IC - Lower	IC - Upper	t	Significance
TS vs BSIG	1.899	1.522	1.788	2.011	33.504	0.000
EDA vs BSIG	5.096	3.607	4.832	5.359	37.930	0.000
MIG vs BSIG	0.379	0.315	0.356	0.402	32.329	0.000

Table 12.9: Paired samples t-test for stopping criterion of $n \cdot m \cdot F \cdot 1$ milliseconds.

Algorithm	Mean	SEM	IC - Lower	IC - Upper	t	Significance
TS vs BSIG	2.084	1.500	1.974	2.194	37.306	0.000
EDA vs BSIG	4.763	3.497	4.507	5.019	36.572	0.000
MIG vs BSIG	0.239	0.247	0.221	0.258	26.077	0.000

Table 12.10: Paired samples t-test for stopping criterion of $n \cdot m \cdot F \cdot 2$ milliseconds.

f	Average Discarded Factories (%)	Decrease in the number of iterations (%)
2	30.71%	14.74%
3	33.05%	12.71%
4	32.99%	13.93%
5	32.48%	9.01%
6	32.31%	9.85%
7	32.33%	9.86%
Average	32.31%	11.68%

Table 12.11: Impact of the bounded search mechanism with the number of factories.

Impact of reduction of the search space

The proposed BSIG includes a mechanism (using Property 12.2.1) to reduce the number of solutions to be evaluated. In this section, the impact of this mechanism on the effectiveness of the heuristic is analysed by comparing the performance of the proposed algorithm with and without the bounded search mechanism for each stopping criterion. The results are shown in Table 12.11 and in Table 12.12 aggregated by the number of factories, and by n and m, respectively. In both tables, the second column indicates the average percentage of branches (factories) discarded in the functions that use this mechanism, whereas the third column shows the average reduction in the number of iterations in the BSIG when employing this mechanism. Additionally, the results have been calculated averaging for the three stopping criteria. Summarizing, it was obtained that a 32.31% of the branches (factories) are discarded in the construction phase and in the RLS1 of the proposed iterated algorithm. Note that there is a substantial decrease of the discarded factories with the increase in the number of machines of the problem for the same number of jobs. This is due to the fact that the chosen lower bound is $min_i\,(t_{i,l})$ with $i \in [1, \cdots, m]$ and it therefore less tight as m increases. Furthermore, the number of iterations of the proposed BSIG for each analysed stopping criterion is increased an 11.7% in average. The difference between this value for large and small size instances is due to RLS2, which does not include the bounded search and needs large computational time when used. Both the number of discarded factories and the decrease in the number of iterations stress the importance of the bounded search mechanism in the algorithm.

n x m	Average Discarded Factories (%)	Decrease in number of iteration (%)
20 x 5	40.42%	8.50%
20 x 10	25.73%	5.07%
20 x 20	19.54%	4.04%
50 x 5	45.79%	6.13%
50 x 10	30.94%	4.79%
50 x 20	19.30%	3.16%
100 x 5	51.90%	21.11%
100 x 10	35.80%	12.65%
100 x 20	21.64%	8.58%
200 x 10	40.27%	27.47%
200 x 20	25.18%	14.86%
500 x 20	31.22%	23.84%
Average	32.31%	11.68%

Table 12.12: Impact of the bounded search mechanism order with the problem size n and m.

12.4 Conclusions

The Distributed Permutation Flowshop Scheduling Problem (DPFSP) consists of two interrelated decision problems: First, jobs are assigned to be processed in one of the f identical factories of the company. Secondly, the sequence of jobs in each factory is determined taking into account that each job has the same manufacturing flow through each one of the m machines. To solve the problem, we have presented a new algorithm (BSIG) consisting of an iterative destruction and greedy construction of the solution with three local search phases. BSIG employs a property of the problem to estimate the makespan of a factory when a new job is inserted, so the search space can be reduced. The evaluation of the performance of BSIG was compared with that of the existing algorithms TS, EDA and MIG using the instances presented by [124] for three different stopping criteria. Each algorithm was implemented under the same conditions. Furthermore, paired samples t-tests were carried out to determine statistically differences between the heuristics. The comparison shows that the proposed BSIG outperforms existing heuristics, thus being the most efficient iterative improvement algorithm for the problem (with a p value of 0.000). On the one hand, comparing the four heuristics, the best solution of the four heuristics was found by BSIG 2008 times out of a total of 2160 instances (summarising results of the three stopping criteria). On the other hand, using the proposed heuristic, a new best known solution was found in 263 of the 720 instances (36.5%) using the stopping criterion of $n \cdot m \cdot F \cdot 2$ milliseconds. Additionally, the effect of the bounded search method in the algorithm was analysed reporting a decrease in the computational times of 32.31% whenever applied.

Part V

CONCLUSIONS

Chapter 13

Conclusions, results and future research lines

13.1 Conclusions

In this Thesis, we have addressed the permutation flowshop scheduling problem, which is one of the most studied scheduling problem in the literature due to its direct application to real manufacturing layouts, and deals with establishing the sequence of jobs in the shop according to a specific objective function. Due to the high complexity of this type of scheduling problems, most of the research has traditionally be focused in proposing approximate algorithms to solve the problem. The goal of this Thesis is therefore to provide a further insight into this important problem, both deeply analysing the influence of the different input parameters and developing new efficient techniques to solve it. In order to deal with this goal, several general research objectives were identified in Section 1.1, which have been addressed along the five parts of this Thesis as follows:

GO1. To review the PFSP for the most common objectives, i.e. makespan, total completion time and due-date-based objectives (total tardiness, and total earliness and tardiness).

The PFSP for makespan minimisation was considered in Section 4.2. For this problem, many heuristic and metaheuristic methods have been published, including several review papers. This chapter covers the last 10 years of highly effective procedures for the problem. Firstly, heuristics implemented for the problem were reviewed. Most of them are variants of the traditional NEH. In addition, we proposed a classification to identify these variants, which depends on three fields: initial order; tie-breaking mechanism; and reversibility property. Regarding the metaheuristics, several fields were identified for each contribution (e.g. the set of instances used, the average relative percentage deviations and the average CPU times of the proposed heuristics,...). However, it was pointed out that frequently new methods are not properly compared with the existing state-of-the-art solutions and misleading conclusions might be obtained.

Regarding the PFSP to minimise total flowtime and due-date related objectives, they were addressed in Sections 4.3 and 4.4, respectively. On the one hand, for total flowtime, the 14 efficient heuristics were extensively reviewed and described. On the other hand, regarding due-date related objectives, we focused in the $Fm|prmu|\sum T_j$ and $Fm|prmu|\sum E_j + T_j$ problems, where the most relevant heuristics for these problems are reviewed.

GO2. To analyse the influence of the processing times and due dates of the jobs on the PFSP.

In Chapter 5, we analysed in detail the problem depending on the processing times and due dates of the jobs. Several properties, theorems and dominance rules were proposed to better understand the structures of the problem under consideration. Firstly, several properties were presented for the $Fm|prmu|\sum T_j$ and $Fm|prmu|\sum E_j + \sum T_j$, as well as the generations of due dates are analysed. It was shown that under several conditions of the due dates of the jobs, the $Fm|prmu|\sum T_j$ can be reduced to the $Fm|prmu|\sum C_j$, to a problem where the EDD rule is optimal or even, to a trivial problem where each sequence is optimal. Analogously, the $Fm|prmu|\sum E_j + \sum T_j$ can be reduce to the $Fm|prmu| - \sum C_j$ ($Fm|prmu|\sum C_j$) in case of extremely loose (tight) due dates.

Additionally, in Section 5.3, we addressed the boundary lines between the PFSP and the SMSP. It was shown that the $Fm|prmu|\sum C_j$ ($Fm|prmu|C_{max}$) problem is equivalent to the $1||\sum C_j$ ($1||C_{max}$) when several conditions of the processing times of the most loaded machine are fulfilled. Additionally, several approximate boundary lines were presented to define when the PFSP tends to turn to a SMSP, given an instance of a problem. These lines give an idea to the decision makers to better identify their manufacturing layouts.

Finally, we identified the advantages and disadvantages of different functions to measure the relationship between the processing time of an operation and the amount of resources assigned to that operation.

GO3. To provide the schedulers with fasters and more efficient heuristics and metaheuristics to solve the PFSP for the makespan, total completion time, total tardiness, and total earliness and tardiness minimisation.

We developed several heuristics and metaheuristics in order to minimise the above objectives. The algorithms were compared under the same conditions with the best algorithms in the literature for each decision problem. Each proposed algorithm was shown to be efficient for the problem in which is proposed. The main proposed approximate procedures are summarised as follows:

- In Chapter 6, a new tie-breaking mechanism was proposed for the $Fm|prmu|C_{max}$ problem. The mechanism is included in the NEH and IG algorithms, which results in a significant improvement in the quality of the solutions. The proposed algorithm was compared with the best 26 algorithms found for the problem in the literature.

- Two new constructive heuristics were proposed for the $Fm|prmu|\sum C_j$ problem in Chapter 7. The heuristics were compared with the so-far-efficient heuristics of the problem. As a result, the proposed algorithms clearly statistically outperformed each constructive heuristic for the problem.

- In Chapter 8, several tie-breaking mechanisms were proposed for the $Fm|prmu|\sum T_j$ problem which improved the most efficient constructive heuristic without increasing the computational effort. These mechanisms also resulted in a significant improvement when they were included in one of the best metaheuristic for the problem.

- In Chapter 9, several efficient constructive and improvement heuristics were proposed for the $Fm|prmu|\sum E_j + \sum T_j$ problem taking advantage of the special properties, proposed for the problem.

GO4. To demonstrated the efficiency and good performance of the solution procedures developed in GO3.

In this Thesis, each proposed approximate procedure was always compared with the state-of-the-art algorithms for that problem. To address it, in Chapter 3, a new indicator to measure the computational effort required by the procedures was proposed, as well as the conditions to carry out a fair comparison were explained in detail (such as recoding of all algorithms in C#, the use of the same computer, among others). Nevertheless, for some scheduling problems, either the state-of-the-art algorithms cannot been well defined or there are many algorithms among the efficient ones. As a consequence, several extensive computational evaluations were also carried out to validate the efficiency of the proposed algorithms. More specifically:

- A computational evaluation of heuristics and metaheuristic for the $Fm|prmu|C_{max}$ (see Section 6.4). The proposed tie-breaking mechanism was compared in a computational evaluation which include a total of 25 approximate algorithms.

- A computational evaluation of heuristics for the $Fm|prmu|\sum C_j$ (see Section 7.5). The two proposed heuristics for that problem were compared with the set of 14 efficient heuristics. After that, the resulting efficient heuristics was compared against the best metaheuristic for the problem.

- A computational evaluation of heuristics for the $Fm|prmu|\sum E_j + T_j$. The proposed heuristics for that problem were compared against the efficient heuristics implemented for the $Fm|prmu|\sum E_j+T_j$ and adaptations of related scheduling problems. As a result, 10 algorithms were compared under the same conditions.

- A computational evaluation of heuristics for the $Fm|block|\sum C_j$. This computational evaluation was composed of all heuristics implemented for the problem so far, as well as efficient algorithms of related scheduling problem. As a consequence, a total of 36 heuristics was fully recoded and exhaustively compared under the same conditions.

GO5. To extend the proposals to constrained PFSP, based on real manufacturing environments.

This objective was extensively addressed in Part IV which can be summarised as follows:

- A non-population metaheuristic was proposed for the $Fm|prmu|\epsilon(C_{max}/T_{max})$ problem in Chapter 10. The algorithm includes an adaptation of the known Taillard's accelerations, originally proposed for the $Fm|prmu|C_{max}$ problem. As a result, the proposed metaheuristic clearly outperformed each other metaheuristic proposed for the problem.

- A beam-search-based constructive heuristic for the $Fm|block|\sum C_j$ problem as well as a speed-up procedure were proposed in Chapter 11. The heuristic was compared in an extensive computational evaluation as established above. Results showed the excellent performance of the proposed algorithm in terms of quality of the solution and computational effort. In fact, the proposed algorithm improved the so-far best metaheuristic for the problem as well as new upper bounds are found for 27.6% of the instances.

- A variation of the iterated greedy algorithm was proposed for the parallel PFSP in Chapter 12, which exploits the specific structure of solutions by means of two dominance rules. The use of these properties reported a decrease in the computational times of 32.31% whenever applied. Computational results showed that the proposed metaheuristic obtained the best result among the implemented heuristics for the 93% of the instances.

13.2 Results

SCI indexed journals

Parts of this Thesis have been published in SCI indexed journals. The following publications are directly derived from this Thesis.

1. Fernandez-Viagas, V., Framinan, J.M., (2015). Efficient non-population-based algorithms for the permutation flowshop scheduling problem with makespan minimisation subject to a maximum tardiness. *Computers & Operations Research*, 64, 86 - 96 (2014 Impact Factor: 1.861, Q1, T1).

2. Fernandez-Viagas, V., Framinan, J.M., (2015). NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness. *Computers & Operations Research*, 60, 27 - 36 (2014 Impact Factor: 1.861, Q1, T1).

3. Fernandez-Viagas, V., Framinan, J.M., (2015). A new set of high-performing heuristics to minimize flowtime in permutation flowshops. *Computers & Operations Research*, 53, 68 - 80 (2014 Impact Factor: 1.861, Q1, T1).

4. Fernandez-Viagas, V., Framinan, J.M., (2015). A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, 53, 1111 - 1123 (2014 Impact Factor: 1.477, Q2, T1).

5. Fernandez-Viagas, V., Framinan, J.M., (2015). Controllable Processing Times in Project and Production Management: Analysing the Trade-Off between Processing Times and the Amount of Resources. *Mathematical Problems in Engineering*, 1 - 19 (2014 Impact Factor: 0.762, Q3, T2).

6. Fernandez-Viagas, V., Framinan, J.M., (2014). On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers & Operations Research*, 45, 60 - 67 (2014 Impact Factor: 1.861, Q1, T1).

7. Fernandez-Viagas, V., Leisten, R., Framinan, J.M. A computational evaluation of constructive and improvement heuristics for the blocking flow shop to minimise total flowtime. Under review in *Expert Systems with Applications*.

8. Fernandez-Viagas, V., Ruiz, R., Framinan, J.M. A new vision of approximate methods for the permutation flowshop to minimise makespan: state-of-the-art and computational evaluation. Under review in *European Journal of Operational Research*.

9. Fernandez-Viagas, V., Framinan, J.M. Reduction of Permutation Flowshop Problems to Single Machine Problems. Under review in *Computers & Operations Research.*

10. Fernandez-Viagas, V., Framinan, J.M. Efficient constructive and composite heuristics for the Permutation Flowshop to minimise total earliness and tardiness. Under review in *Computers & Operations Research.*

11. Fernandez-Viagas, V., Framinan, J.M. A beam-search-based constructive heuristic for the PFSP to minimise total flowtime. Under review in *International Journal of Production Economics.*

Additionally, during the development of this Thesis, the following papers on related scheduling problems have been published in SCI indexed journals:

12. Dios, M., Molina-Pariente J.M., Fernandez-Viagas, V., Andrade-Pineda J.L., Framinan, J.M., (2015). A decision support system for operating room scheduling. Computers and Industrial Engineering. *Computers & Industrial Engineering*, 88, 430-443 (2014 Impact Factor: 1.783, Q2, T1).

13. Molina-Pariente, J. M., Fernandez-Viagas, V., Framinan, J.M., (2015). Integrated operating room planning and scheduling problem with assistant surgeon dependent surgery durations. *Computers & Industrial Engineering*, 82, 8-20 (2014 Impact Factor: 1.783, Q2, T1).

14. Fernandez-Viagas, V., Framinan, J.M., (2014). Integrated Project Scheduling and Staff Assignment with Controllable Processing Times. *Scientific World Journal*, 1 - 16 (2013 Impact Factor: 1.219, Q2, T1).

Papers in conference proceedings

Regarding contributions in international conferences, we below mention the most related ones:

1. Fernandez-Viagas, V., Framinan, J.M. A constructive heuristic for the permutation flowshop to minimise total earliness and tardiness. 15th International Conference on Project Management and Scheduling (PMS 2016). Valencia (Spain), April 19 - 22, 2016.

2. Fernandez-Viagas, V., Framinan, J.M. Boundary lines between permutation flowshop problems and single machine problems. Proceedings of 2015 International Conference on Industrial Engineering and Systems Management (IESM 2015). Seville (Spain), October 21-23, 2015.

3. Perez-Gonzalez P., Dios M., Fernandez-Viagas, V., Framinan, J.M. Heuristic Methods for Single Machine Scheduling with Periodic Maintenance. Multidisciplinary International Scheduling Conference (Mista 2015). Prague (Czech Republic), August 25-28, 2015.

4. Fernandez-Viagas, V., Dios M., Perez-Gonzalez P., Framinan, J.M. A framework of constructive heuristics for permutation-type scheduling problems. Multidisciplinary International Scheduling Conference (Mista 2015). Prague (Czech Republic), August 25-28, 2015.

5. Dios M., Fernandez-Viagas, V., Perez-Gonzalez P., Framinan, J.M. Manufacturing Scheduling Systems: What are they made of?. Multidisciplinary International Scheduling Conference (Mista 2015). Prague (Czech Republic), August 25-28, 2015.

6. Fernandez-Viagas, V., Framinan, J.M. A new fast heuristic to minimize flowtime in permutation flowshops. 14th International Conference on Project Management and Scheduling (PMS 2014). Munich (Germany), March 30 - April 2, 2014.

7. Fernandez-Viagas, V., Framinan, J.M. Approximate algorithms for simultaneous project scheduling and resource allocation with controllable processing times. 25th European Conference on Operational Research. Vilnius (Lithuania), July 8-11, 2012.

Research projects

Finally, this Thesis has been carried out carried out under grant "Predoctoral Research Fellow (FPU12/01935)" funded by the Ministry of Education, Culture and Sport, and has been developed within the framework of several manufacturing scheduling research projects:

- ADDRESS - "Advanced design of dynamic robust extended scheduling systems" funded by the Spanish The Ministry of Economy and Competitiveness (reference DPI-2013-44461-P).
- "e-Fábrica" funded by the Technological Corporation of Andalusia (reference PI-1366/2014).
- SEAMAR funded by the Industrial Technology Development Center (reference PI-1031/2012).
- SCORE – Scheduling and Control for Customer Responsive Production funded by the Interministerial Commission for Science and Technology, CICYT, (reference DPI2010- 15573).
- SCOPE funded by Regional Government of Andalusia (reference P08-TEP-03630).
- PUVENSA funded by IMP Consultores (reference P08-TEP-03630).

13.3 Future research lines

In this section, the main future research lines of this Thesis are discussed.

In this Thesis, the mechanism for due date generation proposed by [147] has been chosen to build the testbed. However, some deficiencies were found in the generation of due dates, since some range of values of indicator v (loose due dates) are not included, as well as most of instances are generated for $v \in [0.15, 0.35]$. As a consequence, further analysis could be conducted to develop more extensive testbeds, including bigger and more uniform intervals for indicator v.

Regarding the analysis of the processing times in the PFSP, although the presented Thesis represents an important point in the study of the relationship between both the PFSP and the SMSP problem, the boundary lines between them are not yet completely defined. Further enhancements may focus on the following issues:

- The variance of the processing times on the saturated machine probably plays an important role in the relationship between both problems.
- The presented study has used an uniform distribution for the processing times. Further analyses can use of several different distributions, extending the boundary lines between the problems.
- The presented analysis may probably be extended to other scheduling layouts.
- The PFSP has been compared with the SMSP of the most saturated machine. Further analysis may compare the PFSP with a SMSP combining the processing times of different machines.

Additionally, new relations may be considered for discrete resources since there are almost no papers using them. To the best of our knowledge, only the hyperbola has been used to represent the inverted U-shaped of the productivity in project management. Due to the difficulties to determine the constants of this relation, new relations may be considered in order to represent the excess of communication and the lack of specialization together.

Regarding the approximate procedures proposed in this Thesis, we have found that for the PFSP to minimise makespan, the best metaheuristics and heuristics include special characteristics of the problem as Taillard's accelerations and tie-breaking mechanisms. In our opinion, these facts highlight that future advances in this field will come from a better understanding of the problem and its properties, which should also be extended to other objective functions.

Finally, several of the proposed approximate procedures can easily be adapted to both other manufacturing scheduling problems and related scheduling problems outside production management, such as [40], [32] and [119], which were also developed during this Thesis as explained in Section 13.2.

Bibliography

[1] A. Agarwal, S. Colak, and E. Eryarsoy. Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach. *European Journal of Operational Research*, 169(3):801–815, 2006.

[2] F. Ahmadizar. A new ant colony algorithm for makespan minimization in permutation flow shops. *Computers and Industrial Engineering*, 63(2):355–361, 2012.

[3] A. Al-Salem. A heuristic to minimize makespan in proportional parallel flow shops. *International Journal of Computing & Information Sciences*, 2(2):98–107, 2004.

[4] K. R. Baker and G. D. Scudder. Sequencing with earliness and tardiness penalties. a review. *Operations Research*, 38(1):22–36, 1990.

[5] Y. Bao, L. Zheng, and H. Jiang. An improved hs algorithms for the blocking flow shop scheduling problems. *Proceedings - 2012 International Conference on Computer Science and Information Processing, CSIP 2012*, pages 1289–1291, 2012.

[6] R. M. Belbin. *Management Teams*. Butterworth-Heinemann, Amsterdam; Oxford, 2010.

[7] R. A. Bilas. *Microeconomic theory*. McGraw-Hill, 1971.

[8] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Handbook on Scheduling: From Theory to Applications*. Springer, 2007.

[9] D. Cao and M. Chen. Parallel flowshop scheduling using tabu search. *International Journal of Production Research*, 41(13):3059–3073, 2003.

[10] J. Carlier. Ordonnancements a contraintes disjonctives. *RAIRO Recherche Operationnelle*, 12(4):333–350, 1978.

[11] O. Cepek, M. Okada, and M. Vlach. Nonpreemptive flowshop scheduling with machine dominance. *European Journal of Operational Research*, 139(2):245–261, 2002.

[12] K. Chakravarthy and C. Rajendran. Heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization. *Production Planning and Control*, 10(7):707–714, 1999.

[13] F.T.S. Chan, S.H. Chung, and P.L.Y. Chan. An adaptive genetic algorithm with dominated genes for distributed scheduling problems. *Expert Systems with Applications*, 29(2):364–371, 2005.

[14] H.K. Chan and F.T.S. Chan. Comparative study of adaptability and flexibility in distributed manufacturing supply chains. *Decision Support Systems*, 48(2):331–341, 2010.

[15] P. Chandra, P. Mehta, and D. Tirupati. Permutation flow shop scheduling with earliness and tardiness penalties. *International Journal of Production Research*, 47(20):5591–5610, 2009.

[16] P.-C. Chang and M.-H. Chen. A block based estimation of distribution algorithm using bivariate model for scheduling problems. *Soft Computing*, 18(6):1177–1188, 2014.

[17] P.-C. Chang, M.-H. Chen, M.K. Tiwari, and A.S. Iquebal. A block-based evolutionary algorithm for flow-shop scheduling problem. *Applied Soft Computing Journal*, 13(12):4536–4547, 2013.

[18] P.-C. Chang, S.-H. Chen, C.-Y. Fan, and C.-L. Chan. Genetic algorithm integrated with artificial chromosomes for multi-objective flowshop scheduling problems. *Applied Mathematics and Computation*, 205(2):550–561, 2008.

[19] P.-C. Chang, S.-H. Chen, C.-Y. Fan, and V. Mani. Generating artificial chromosomes with probability control in genetic algorithm for machine scheduling problems. *Annals of Operations Research*, 180(1):197–211, 2010.

[20] P.-C. Chang, J.-C. Hsieh, S.-H. Chen, J.-L. Lin, and W.-H. Huang. Artificial chromosomes embedded in genetic algorithm for a chip resistor scheduling problem in minimizing the makespan. *Expert Systems with Applications*, 36(3 PART 2):7135–7141, 2009.

[21] P.-C. Chang, W.-H. Huang, and C.-J. Ting. A hybrid genetic-immune algorithm with improved lifespan and elite antigen for flow-shop scheduling problems. *International Journal of Production Research*, 49(17):5207–5230, 2011.

[22] C.-L. Chen, Y.-R. Tzeng, and C.-L. Chen. A new heuristic based on local best solution for permutation flow shop scheduling. *Applied Soft Computing Journal*, 29:75–81, 2015.

[23] R.-M. Chen and F.-R. Hsieh. An exchange local search heuristic based scheme for permutation flow shop problems. *Applied Mathematics and Information Sciences*, 8(1 L):209–215, 2014.

[24] S.-H. Chen, P.-C. Chang, T.C.E. Cheng, and Q. Zhang. A self-guided genetic algorithm for permutation flowshop scheduling problems. *Computers & Operations Research*, 39(7):1450–1457, 2012.

[25] M. Cheng, S. Sun, and Y. Yu. A note on flow shop scheduling problems with a learning effect on no-idle dominant machines. *Applied Mathematics and Computation*, 184(2):945–949, 2007.

[26] S.H. Chung, F.T.S. Chan, and H.K. Chan. A modified genetic algorithm approach for scheduling of perfect maintenance in distributed production scheduling. *Engineering Applications of Artificial Intelligence*, 22(7):1005–1014, 2009.

[27] R.L. Daniels and R.J. Chambers. Multiobjective flow-shop scheduling. *Naval Research Logistics*, 37(6):981–995, 1990.

[28] P. Dasgupta and S. Das. A discrete inter-species cuckoo search for flowshop scheduling problems. *Computers and Operations Research*, 60(0):111 – 120, 2015.

[29] E. Demeulemeester, B. De Reyck, and W. Herroelen. The discrete time/resource trade-off problem in project networks: A branch-and-bound approach. *IIE Transactions (Institute of Industrial Engineers)*, 32(11):1059–1069, 2000.

[30] E. Demirkol, S. Mehta, and R. Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998.

[31] G. Deng, Z. Xu, and X. Gu. A discrete artificial bee colony algorithm for minimizing the total flow time in the blocking flow shop scheduling. *Chinese Journal of Chemical Engineering*, 20(6):1067–1073, 2012.

[32] M. Dios, J.M. Molina-Pariente, V. Fernandez-Viagas, J.L. Andrade-Pineda, and J.M. Framinan. A decision support system for operating room scheduling. *Computers and Industrial Engineering*, 88:430–443, 2015.

[33] B. Dodin and A.A. Elimam. Audit scheduling with overlapping activities and sequence-dependent setup costs. *European Journal of Operational Research*, 97(1):22–33, 1997.

[34] X. Dong, P. Chen, H. Huang, and M. Nowak. A multi-restart iterated local search algorithm for the permutation flow shop problem minimizing total flow time. *Computers & Operations Research*, 40(2):627–632, 2013.

[35] X. Dong, H. Huang, and P. Chen. An improved NEH-based heuristic for the permutation flowshop problem. *Computers & Operations Research*, 35(12):3962–3968, December 2008.

[36] A. Drexl. Scheduling of project networks by job assignment. *Management Science*, 37(12):1590–1602, 1991.

[37] B. Eksioglu, S.D. Eksioglu, and P. Jain. A tabu search algorithm for the flowshop scheduling problem with changing neighborhoods. *Computers and Industrial Engineering*, 54(1):1–11, 2008.

[38] O. Etiler, B. Toklu, M. Atak, and J. Wilson. A genetic algorithm for flow shop scheduling problems. *Journal of the Operational Research Society*, 55(8):830–835, 2004.

[39] L. Fanjul-Peyro and R. Ruiz. Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1):55–69, 2010.

[40] V. Fernandez-Viagas and J.M. Framinan. Integrated project scheduling and staff assignment with controllable processing times. *Scientific World Journal*, 2014, 2014.

[41] V. Fernandez-Viagas and J.M. Framinan. A new set of high-performing heuristics to minimise flowtime in permutation flowshops. *Computers & Operations Research*, 53:68–80, 2015.

[42] J. M. Framinan, R. Leisten, and C. 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(1):121–148, 2003.

[43] J.M. Framinan, J.N.D. Gupta, and R. Leisten. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55(12):1243–1255, 2004.

[44] J.M. Framinan and R. Leisten. An efficient constructive heuristic for flowtime minimisation in permutation flowshops. *OMEGA, The International Journal of Management Science*, 31:311–317, 2003.

[45] J.M. Framinan and R. Leisten. A heuristic for scheduling a permutation flowshop with makespan objective subject to maximum tardiness. *International Journal of Production Economics*, 99(1-2):28–40, 2006.

[46] J.M. Framinan and R. Leisten. Total tardiness minimization in permutation flow shops: A simple approach based on a variable greedy algorithm. *International Journal of Production Research*, 46(22):6479–6498, 2008.

[47] J.M. Framinan, R. Leisten, and R. Ruiz. *Manufacturing Scheduling Systems: An Integrated View on Models, Methods, and Tools.* Springer, 2014.

[48] J.M. Framinan, R. Leisten, and R. Ruiz-Usano. Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation. *European Journal of Operational Research*, 141(3):559–569, 2002.

[49] J.M. Framinan, R. Leisten, and R. Ruiz-Usano. Comparison of heuristics for flowtime minimisation in permutation flowshops. *Computers & Operations Research*, 32(5):1237–1254, 2005.

[50] J.M. Framinan and R. Pastor. A proposal for a hybrid meta-strategy for combinatorial optimization problems. *Journal of Heuristics*, 14(4):375–390, 2008.

[51] L. Fried. Team size and productivity in systems development. bigger does not always mean better. *Journal of Information Systems Management*, 8:27–35, 1991.

[52] Y. Gajpal and C. Rajendran. An ant-colony optimization algorithm for minimizing the completion-time variance of jobs in flowshops. *International Journal of Production Economics*, 101(2):259–272, 2006.

[53] J. Gao and R. Chen. A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Computational Intelligence Systems*, 4(4):497–508, 2011.

[54] J. Gao, R. Chen, and W. Deng. An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, 51(3):641–651, 2013.

[55] M.R. Garey, D.S. Johnson, and Ravi Sethi. Complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[56] Ludo F. Gelders and Narayanasamy Sambandam. Four simple heuristics for scheduling a flow-shop. *International Journal of Production Research*, 16(3):221–231, 1978.

[57] H. Gong, L. Tang, and C.W. Duin. A two-stage flow shop scheduling problem on a batching machine and a discrete machine with blocking and shared setup times. *Computers & Operations Research*, 37(5):960–969, 2010.

[58] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[59] N.G. Hall and C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3):510–525, 1996.

[60] S. Hamed Hendizadeh, H. Faramarzi, S.A. Mansouri, J.N.D. Gupta, and T. Y ElMekkawy. Meta-heuristics for scheduling a flowline manufacturing cell with sequence dependent family setup times. *International Journal of Production Economics*, 111(2):593–605, 2008.

[61] Y.-Y. Han, J.-H. Duan, Y.-J. Yang, M. Zhang, and B. Yun. Minimizing the total flowtime flowshop with blocking using a discrete artificial bee colony. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6839 LNAI:91–97, 2011.

[62] Y.-Y. Han, J.J. Liang, Q.-K. Pan, J.-Q. Li, H.-Y. Sang, and N.N. Cao. Effective hybrid discrete artificial bee colony algorithms for the total flowtime minimization in the blocking flowshop problem. *International Journal of Advanced Manufacturing Technology*, 67(1-4):397–414, 2013.

[63] Y.-Y. Han, Q.-K. Pan, J.-Q. Li, and H.-Y. Sang. An improved artificial bee colony algorithm for the blocking flowshop scheduling problem. *International Journal of Advanced Manufacturing Technology*, 60(9-12):1149–1159, 2012.

[64] A.N. Haq, T.R. Ramanan, K.S. Shashikant, and R. Sridharan. A hybrid neural network-genetic algorithm approach for permutation flow shop scheduling. *International Journal of Production Research*, 48(14):4217–4231, 2010.

[65] R. Hariharan and R.J. Golden Renjith Nimal. Solving flow shop scheduling problems using a hybrid genetic scatter search algorithm. *Middle - East Journal of Scientific Research*, 20(3):328–333, 2014.

[66] S. Hartmann and D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

[67] S. Hasija and C. Rajendran. Scheduling in flowshops to minimize total tardiness of jobs. *International Journal of Production Research*, 42(11):2289–2301, 2004.

[68] T. E. Hastings and A. S. M. Sajeev. A vector-based approach to software size measurement and effort estimation. *IEEE Transactions on Software Engineering*, 27:337–350, 2001.

[69] S. Hatami, R. Ruiz, and C. Andrés-Romano. The distributed assembly permutation flowshop scheduling problem. *International Journal of Production Research*, 2013.

[70] C. Heimerl and R. Kolisch. Work assignment to and qualification of multi-skilled human resources under knowledge depreciation and company skill level targets. *International Journal of Production Research*, 48(13):3759–3781, 2010.

[71] J. Heller. Some numerical experiments for an m x j flow shop and its decision-theoretical aspects. *Operations Research*, 8(2):178–184, 1960.

[72] J.C. Ho and J.N.D. Gupta. Flowshop scheduling with dominant machines. *Computers and Operations Research*, 22(2):237–246, 1995.

[73] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[74] W.J. Hopp and M.L. Spearman. *Factory Physics*. McGraw-Hill/Irwin, Boston, Massachusetts, 2000.

[75] C.-Y. Hsu, P.-C. Chang, and M.-H. Chen. A linkage mining in block-based evolutionary algorithm for permutation flowshop scheduling problem. *Computers and Industrial Engineering*, 83:159–171, 2015.

[76] W.Q. Huang and L. Wang. A local search method for permutation flow shop scheduling. *Journal of the Operational Research Society*, 57(10):1248–1251, 2006.

[77] B. Jarboui, S. Ibrahim, P. Siarry, and A. Rebai. A combinatorial particle swarm optimisation for solving permutation flowshop problems. *Computers and Industrial Engineering*, 54(3):526–538, 2008.

[78] H.Z. Jia, J.Y.H. Fuh, A.Y.C. Nee, and Y.F. Zhang. Integration of genetic algorithm and gantt chart for job shop scheduling in distributed manufacturing systems. *Computers and Industrial Engineering*, 53(2):313–320, 2007.

[79] H.Z. Jia, A.Y.C. Nee, J.Y.H. Fuh, and Y.F. Zhang. A modified genetic algorithm for distributed scheduling problems. *Journal of Intelligent Manufacturing*, 14(3-4):351–362, 2003.

[80] S.M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[81] J. Józefowska. *Just-in-time Scheduling*. Springer, New York, 2007.

[82] K.B. Kahn, G.A. Castellion, and A. Griffin. *The PDMA Handbook of New Product Development: Second Edition*. Wiley, 2004.

[83] P. J. Kalczynski and J. Kamburowski. On the NEH heuristic for minimizing the makespan in permutation flow shops. *OMEGA, The International Journal of Management Science*, 35(1):53–60, February 2007.

[84] P. J. Kalczynski and J. Kamburowski. An improved NEH heuristic to minimize makespan in permutation flow shops. *Computers & Operations Research*, 35(9):3001–3008, September 2008.

[85] P. J. Kalczynski and J. Kamburowski. An empirical analysis of the optimality rate of flow shop heuristics. *European Journal of Operational Research*, 198(1):93 – 101, 2009.

[86] P. J. Kalczynski and J. Kamburowski. On recent modifications and extensions of the NEH heuristic for flow shop sequencing. *Foundations of Computing and Decision Sciences*, 36(1):17–34, 2011.

[87] Y. Kara, C. Ögüven, N. Yalçin, and Y. Atasagun. Balancing straight and u-shaped assembly lines with resource dependent task times. *International Journal of Production Research*, 49(21):6387–6405, 2011.

[88] Y.-D. Kim. Heuristics for flowshop scheduling problems minimizing mean tardiness. *Journal of the Operational Research Society*, 44(1):19–28, 1993.

[89] Y.-D. Kim, J.-G. Kim, B. Choi, and H.-U. Kim. Production scheduling in a semiconductor wafer fabrication facility producing multiple product types with distinct due dates. *IEEE Transactions on Robotics and Automation*, 17(5):589–598, 2001.

[90] Y.-D. Kim, H.-G. Lim, and M.-W. Park. Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process. *European Journal of Operational Research*, 91(1):124–143, 1996.

[91] B. Kitchenham and E. Mendes. Software productivity measurement using multiple size measures. *IEEE Transactions on Software Engineering*, 30:1023–1035, 2004.

[92] I.-H. Kuo, S.-J. Horng, T.-W. Kao, T.-L. Lin, C.-L. Lee, T. Terano, and Y. Pan. An efficient flowshop scheduling algorithm based on a hybrid particle swarm optimization model. *Expert Systems with Applications*, 36(3 PART 2):7027–7032, 2009.

[93] D. Laha and U.K. Chakraborty. An efficient hybrid heuristic for makespan minimization in permutation flow shop scheduling. *International Journal of Advanced Manufacturing Technology*, 44(5-6):559–569, 2009.

[94] V. Lauff and F. Werner. Scheduling with common due date, earliness and tardiness penalties for multimachine problems: A survey. *Mathematical and Computer Modelling*, 40(5-6):637–655, 2004.

[95] R. Leisten and C. Rajendran. Variability of completion time differences in permutation flow shop scheduling. *Computers & Operations Research*, 54:155–167, 2014.

[96] X. Li, Q. Wang, and C. Wu. Efficient composite heuristics for total flowtime minimization in permutation flow shops. *Omega*, 37(1):155–164, February 2009.

[97] X. Li and C. Wu. An efficient constructive heuristic for permutation flow shops to minimize total flowtime. *Chinese Journal of Electronics*, 14(2):203–208, 2005.

[98] X. Li and M. Yin. A discrete artificial bee colony algorithm with composite mutation strategies for permutation flow shop scheduling problem. *Scientia Iranica*, 19(6):1921–1935, 2012.

[99] X. Li and M. Yin. A hybrid cuckoo search via Lévy flights for the permutation flow shop scheduling problem. *International Journal of Production Research*, 51(16):4732–4754, 2013.

[100] X. Li and M. Yin. An opposition-based differential evolution algorithm for permutation flow shop scheduling based on diversity measure. *Advances in Engineering Software*, 55:10–31, 2013.

[101] Z. Lian, X. Gu, and B. Jiao. A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan. *Applied Mathematics and Computation*, 175(1):773–785, 2006.

[102] Z. Lian, X. Gu, and B. Jiao. A novel particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan. *Chaos, Solitons and Fractals*, 35(5):851–861, 2008.

[103] C.-J. Liao, Chao-Tang Tseng, and P. Luarn. A discrete version of particle swarm optimization for flowshop scheduling problems. *Computers & Operations Research*, 34(10):3099–3111, 2007.

[104] Q. Lin, L. Gao, X. Li, and C. Zhang. A hybrid backtracking search algorithm for permutation flow-shop scheduling problem. *Computers and Industrial Engineering*, 2015.

[105] S.-W. Lin, K.-C. Ying, and C.-Y. Huang. Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. *International Journal of Production Research*, 51(16):5029–5038, 2013.

[106] B. Liu, L. Wang, and Y.-H. Jin. An effective pso-based memetic algorithm for flow shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(1):18–27, 2007.

[107] H. Liu, L. Gao, and Q. Pan. A hybrid particle swarm optimization with estimation of distribution algorithm for solving permutation flowshop scheduling problem. *Expert Systems with Applications*, 38(4):4348–4360, 2011.

[108] J Liu and CR Reeves. Constructive and composite heuristic solutions to the $P||\sum c_i$ scheduling problem. *European Journal of Operational Research*, 132:439–452, 2001.

[109] R. Liu, C. Ma, W. Ma, and Y. Li. A multipopulation pso based memetic algorithm for permutation flow shop scheduling. *The Scientific World Journal*, 2013, 2013.

[110] Y. Liu, M. Yin, and W. Gu. An effective differential evolution algorithm for permutation flow shop scheduling problem. *Applied Mathematics and Computation*, 248:143–159, 2014.

[111] Y.-F. Liu and S.-Y. Liu. A hybrid discrete artificial bee colony algorithm for permutation flowshop scheduling problem. *Applied Soft Computing Journal*, 13(3):1459–1463, 2013.

[112] C. Low, J.-Y. Yeh, and K.-I. Huang. A robust simulated annealing heuristic for flow shop scheduling problems. *International Journal of Advanced Manufacturing Technology*, 23(9-10):762–767, 2004.

[113] A. MacCormack, C. F. Kemerer, M. Cusumano, and B. Crandall. Trade-offs between productivity and quality in selecting software development practices. *IEEE Software*, 20:78–85, 2003.

[114] N. Madhushini, C. Rajendran, and Y. Deepa. Branch-and-bound algorithms for scheduling in permutation flowshops to minimize the sum of weighted flowtime/sum of weighted tardiness/sum of weighted flowtime and weighted tardiness/sum of weighted flowtime, weighted tardiness and weighted earliness of jobs. *Journal of the Operational Research Society*, 60(7):991–1004, 2009.

[115] Y. Marinakis and M. Marinaki. Particle swarm optimization with expanding neighborhood topology for the permutation flowshop scheduling problem. *Soft Computing*, 17(7):1159–1173, 2013.

[116] S.T. McCormick, M. L. Pinedo, S. Shenker, and B. Wolf. Sequencing in an assembly line with blocking to minimize cycle time. *Operations Research*, 37(6):925–935, 1989.

[117] R. M'Hallah. An iterated local search variable neighborhood descent hybrid heuristic for the total earliness tardiness permutation flow shop. *International Journal of Production Research*, 52(13):3802–3819, 2014.

[118] R. M'Hallah. Minimizing total earliness and tardiness on a permutation flow shop using vns and mip. *Computers and Industrial Engineering*, 75(1):142–156, 2014.

[119] J.M. Molina-Pariente, V. Fernandez-Viagas, and J.M. Framinan. Integrated operating room planning and scheduling problem with assistant surgeon dependent surgery durations. *Computers and Industrial Engineering*, 82:8–20, 2015.

[120] C. L. Monma and A. H. G. Rinnooy Kan. A concise survey of efficiently solvable special cases of the permutation flow-shop problem. *RAIRO - Operations Research - Recherche Opérationnelle*, 17(2):105–119, 1983.

[121] C. Moon, J. Kim, and S. Hur. Integrated process planning and scheduling with minimizing total tardiness in multi-plants supply chain. *Computers and Industrial Engineering*, 43(1-2):331–349, 2002.

[122] G. Moslehi and D. Khorasanian. Optimizing blocking flow shop scheduling problem with total completion time criterion. *Computers & Operations Research*, 40(7):1874–1883, 2013.

[123] G. Moslehi, M. Mirzaee, M. Vasei, M. Modarres, and A. Azaron. Two-machine flow shop scheduling to minimize the sum of maximum earliness and tardiness. *International Journal of Production Economics*, 122(2):763–773, 2009.

[124] B. Naderi and R. Ruiz. The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, 37(4):754–768, 2010.

[125] M.S. Nagano and J.V. Moccellin. A high quality solution constructive heuristic for flow shop sequencing. *Journal of the Operational Research Society*, 53(12):1374–1379, 2002.

[126] M.S. Nagano, R. Ruiz, and L.A.N. Lorena. A constructive genetic algorithm for permutation flowshop scheduling. *Computers and Industrial Engineering*, 55(1):195–207, 2008.

[127] M. Nawaz, Jr. E. E. Enscore, and I. Ham. A Heuristic Algorithm for the m-Machine, n-Job Flowshop Sequencing Problem. *OMEGA, The International Journal of Management Science*, 11(1):91–95, 1983.

[128] A.C. Nearchou. The effect of various operators on the genetic search for large scheduling problems. *International Journal of Production Economics*, 88(2):191–203, 2004.

[129] A.C. Nearchou. Flow-shop sequencing using hybrid simulated annealing. *Journal of Intelligent Manufacturing*, 15(3):317–328, 2004.

[130] A.C. Nearchou. A novel metaheuristic approach for the flow shop scheduling problem. *Engineering Applications of Artificial Intelligence*, 17(3):289–300, 2004.

[131] V. F. Nieva, E. A. Fleishman, and A. Rieck. *Team Dimensions: Their Identity, their Measurement and their Relationships*. Advanced Research Resources Organizations, Washington, DC, 1985.

[132] E. Nowicki and C. Smutnicki. Some aspects of scatter search in the flow-shop problem. *European Journal of Operational Research*, 169(2):654–666, 2006.

[133] E. Nowicki and S. Zdrzalka. A survey of results for sequencing problems with controllable processing times. *Discrete Applied Mathematics*, 26(2-3):271–287, 1990.

[134] G. Onwubolu and D. Davendra. Scheduling flow shops using differential evolution algorithm. *European Journal of Operational Research*, 171(2):674–692, 2006.

[135] I. Osman and C. Potts. Simulated annealing for permutation flow-shop scheduling. *Omega*, 17(6):551–557, 1989.

[136] Q.-K. Pan and R. Ruiz. Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research*, 222(1):31–43, 2012.

[137] Q.-K. Pan and R. Ruiz. A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers & Operations Research*, 40(1):117–128, 2013.

[138] Q.-K. Pan, M.F. Tasgetiren, and Y.-C. Liang. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers and Industrial Engineering*, 55(4):795–816, 2008.

[139] Q.-K. Pan and L. Wang. Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. *Omega*, 40(2):218–229, 2012.

[140] Q.-K. Pan, L. Wang, and B.-H. Zhao. An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. *International Journal of Advanced Manufacturing Technology*, 38(7-8):778–786, 2008.

[141] S.S. Panwalkar, R.A. Dudek, and M.L. Smith. Sequencing research and the industrial problem. In *Symposium on the Theory of Scheduling*. Springer, Berlin, 1973.

[142] S.S. Panwalkar, M.L. Smith, and A. Seidmann. Common due date assignment to minimize total penalty for the one machine scheduling problem. *Operations Research*, 30(2):391–399, 1982.

[143] P. C. Pendharkar and J. A. Rodger. An empirical study of the impact of team size on software development effort. *Information Technology and Management*, 8:253–262, 2007.

[144] P. Perez-Gonzalez and J.M. Framinan. Scheduling permutation flowshops with initial availability constraint: Analysis of solutions and constructive heuristics. *Computers and Operations Research*, 36(10):2866–2876, 2009.

[145] R. S. Pindyck and L. D. Rubinfeld. *Microeconomics*. Prentice Hall, 2008.

[146] M. Pinedo. *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, 1995.

[147] C.N. Potts and L.N. Van Wassenhove. A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters*, 1(5):177–181, 1982.

[148] G. Prabhaharan, B.S.H. Khan, and L. Rakesh. Implementation of grasp in flow shop scheduling. *International Journal of Advanced Manufacturing Technology*, 30(11-12):1126–1131, 2006.

[149] B. Qian, L. Wang, R. Hu, W.-L. Wang, D.-X. Huang, and X. Wang. A hybrid differential evolution method for permutation flow-shop scheduling. *International Journal of Advanced Manufacturing Technology*, 38(7-8):757–777, 2008.

[150] S. F. Rad, R. Ruiz, and N. Boroojerdian. New high performing heuristics for minimizing makespan in permutation flowshops. *OMEGA, The International Journal of Management Science*, 37(2):331–345, April 2009.

[151] C. Rajendran. Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. *International Journal of Production Economics*, 29(1):65–73, February 1993.

[152] C. Rajendran and H. Ziegler. An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. *European Journal of Operational Research*, 103:129–138, 1997.

[153] C. Rajendran and H. Ziegler. Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times. *European Journal of Operational Research*, 149(3):513–522, 2003.

[154] C. Rajendran and H. Ziegler. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, 155(2):426–438, 2004.

[155] R. Rajkumar and P. Shahabudeen. An improved genetic algorithm for the flowshop scheduling problem. *International Journal of Production Research*, 47(1):233–249, 2009.

[156] N. Raman. Minimum tardiness scheduling in flow shops: Construction and evaluation of alternative solution approaches. *Journal of Operations Management*, 12(2):131–151, 1995.

[157] T.R. Ramanan, R. Sridharan, K.S. Shashikant, and A.N. Haq. An artificial neural network based heuristic for flow shop scheduling problems. *Journal of Intelligent Manufacturing*, 22(2):279–288, 2011.

[158] C.R. Reeves. A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1):5–13, 1995.

[159] G.V. Reklaitis. Review of scheduling of process operations. *AIChE Symposium Series*, 78(214):119–133, 1982.

[160] S. Reza Hejazi and S. Saghafian. Flowshop-scheduling problems with makespan criterion: A review. *International Journal of Production Research*, 43(14):2895–2929, 2005.

[161] I. Ribas and R. Companys. Efficient heuristic algorithms for the blocking flow shop scheduling problem with total flow time minimization. *Computers and Industrial Engineering*, 87:30–39, 2015.

[162] I. Ribas, R. Companys, and X. Tort-Martorell. Comparing three-step heuristics for the permutation flow shop problem. *Computers & Operations Research*, 37(12):2062–2070, December 2010.

[163] I. Ribas, R. Companys, and X. Tort-Martorell. An iterated greedy algorithm for the flowshop scheduling problem with blocking. *Omega*, 39(3):293–301, 2011.

[164] I. Ribas, R. Companys, and X. Tort-Martorell. A competitive variable neighbourhood search algorithm for the blocking flow shop problem. *European Journal of Industrial Engineering*, 7(6):729–754, 2013.

[165] I. Ribas, R. Companys, and X. Tort-Martorell. An efficient discrete artificial bee colony algorithm for the blocking flow shop problem with total flowtime minimization. *Expert Systems with Applications*, 42(15-16), 2015.

[166] A. H. G. Rinnooy Kan. *Machine Scheduling Problems: Classification, Complexity and Computations*. Martinus Nijhoff, The Hague, 1976.

[167] D. Rodriguez, M.A. Sicilia, E. Garcia, and R. Harrison. Empirical findings on team size and productivity in software development. *Journal of Systems and Software*, 85(3):562–570, 2012.

[168] D. P. Ronconi and E. G. Birgin. Mixed-integer programming models for flowshop scheduling problems minimizing the total earliness and tardiness. In Roger Z. Rios-Mercado and Yasmin A. Rios-Solis, editors, *Just-in-Time Systems*, Springer Optimization and Its Applications, pages 91–105. Springer New York, 2012.

[169] D.P. Ronconi. A note on constructive heuristics for the flowshop problem with blocking. *International Journal of Production Economics*, 87(1):39–48, 2004.

[170] R. Ruiz and A. Allahverdi. No-wait flowshop with separate setup times to minimize maximum lateness. *International Journal of Advanced Manufacturing Technology*, 35(5-6):551–565, 2007.

[171] R. Ruiz and A. Allahverdi. Minimizing the bicriteria of makespan and maximum tardiness with an upper bound on maximum tardiness. *Computers & Operations Research*, 36(4):1268–1283, 2009.

[172] R. Ruiz and C. Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494, 2005.

[173] R. Ruiz, C. Maroto, and J. Alcaraz. Two new robust genetic algorithms for the flowshop scheduling problem. *Omega*, 34(5):461–476, 2006.

[174] R. Ruiz and T Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

[175] R. Ruiz and T. Stützle. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143–1159, 2008.

[176] M. Saravanan, A. Noorul Haq, A.R. Vivekraj, and T. Prasad. Performance evaluation of the scatter search method for permutation flowshop sequencing problems. *International Journal of Advanced Manufacturing Technology*, 37(11-12):1200–1208, 2008.

[177] M.K. Sayadi, R. Ramezanian, and N. Ghaffari-Nasab. A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. *International Journal of Industrial Engineering Computations*, 1(1):1–10, 2010.

[178] J. Schaller. Scheduling a permutation flow shop with family setups to minimise total tardiness. *International Journal of Production Research*, 50(8):2204–2217, 2012.

[179] J. Schaller and J.M.S. Valente. A comparison of metaheuristic procedures to schedule jobs in a permutation flow shop to minimise total earliness and tardiness. *International Journal of Production Research*, 51(3):772–779, 2013.

[180] T. Sen and S.K. Gupta. A state-of-art survey of static scheduling research involving due dates. *Omega*, 12(1):63–76, 1984.

[181] S.P. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz, and W. Kubiak. Sequencing of parts and robot moves in a robotic cell. *International Journal of Flexible Manufacturing Systems*, 4(3-4):331–358, 1992.

[182] D. Shabtay and G. Steiner. A survey of scheduling with controllable processing times. *Discrete Applied Mathematics*, 155(13):1643–1666, 2007.

[183] D. Shabtay and G. Steiner. Scheduling to maximize the number of just-in-time jobs: A survey. In Roger Z. Rios-Mercado and Yasmin A. Rios-Solis, editors, *Just-in-Time Systems*, Springer Optimization and Its Applications, pages 3–20. Springer New York, 2012.

[184] R. Slowinski. Two approaches to problems of resource allocation among projet activities - a comparative study. *Journal of the Operational Research Society*, 31(8):711–723, 1980.

[185] K. G. Smith, K. A. Smith, J. D. Olian, H. P. Sims Jr., D. P. O'Bannon, and J. A. Scully. Top management team demography and process: The role of social integration and communication. *Administrative Science Quarterly*, 39:412–438, 1994.

[186] M. Solimanpur, P. Vrat, and R. Shankar. A neuro-tabu search heuristic for the flow shop scheduling problem. *Computers & Operations Research*, 31(13):2151–2164, 2004.

[187] T Stützle. Applying iterated local search to the permutation flow shop problem. *Technical report, AIDA-98-04, FG Intellektik, FB Informatik, TU Darmstadt*, 1998.

[188] Y. Sun, C. Zhang, L. Gao, and X. Wang. Multi-objective optimization algorithms for flow shop scheduling problem: A review and prospects. *International Journal of Advanced Manufacturing Technology*, 55(5-8):723–739, 2011.

[189] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74, 1990.

[190] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[191] K.-C. Tan, R. Narasimhan, P.A. Rubin, and G.L. Ragatz. A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times. *OMEGA, The International Journal of Management Science*, 28(3):313–326, 2000.

[192] M.F. Tasgetiren, Y.-C. Liang, M. Sevkli, and G. Gencyilmaz. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, 177(3):1930–1947, 2007.

[193] V. T'Kindt and J.-C. Billaut. *Multicriteria Scheduling: Theory, Models and Algorithms.* Springer, New York, second edition, 2006.

[194] L.-Y. Tseng and Y.-T. Lin. A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 198(1):84–92, 2009.

[195] Y.-R. Tzeng and C.-L. Chen. A hybrid eda with acs for solving permutation flow shop scheduling. *International Journal of Advanced Manufacturing Technology*, 60(9-12):1139–1147, 2012.

[196] G. Vairaktarakis and M. Elhafsi. The use of flowlines to simplify routing complexity in two-stage flowshops. *IIE Transactions (Institute of Industrial Engineers)*, 32(8):687–699, 2000.

[197] E. Vallada and R. Ruiz. Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *Omega*, 38(1-2):57–67, 2010.

[198] E. Vallada, R. Ruiz, and J.M. Framinan. New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, 240:666–677, 2015.

[199] E. Vallada, R. Ruiz, and G. Minella. Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research*, 35(4):1350–1373, 2008.

[200] V. Valls, A. Pérez, and S. Quintanilla. Skilled workforce scheduling in service centres. *European Journal of Operational Research*, 193(3):791–804, 2009.

[201] D. Vasiljevic and M. Danilovic. Handling ties in heuristics for the permutation flow shop scheduling problem. *Journal of Manufacturing Systems*, 35:1–9, 2015.

[202] T. E. Vollman, W. L. Berry, and D. C. Wybark. *Manufacturing Planning and Control Systems*. McGraw-Hill, New York, 1997.

[203] J.-B. Wang, F. Shan, B. Jiang, and L.-Y. Wang. Permutation flow shop scheduling with dominant machines to minimize discounted total weighted completion time. *Applied Mathematics and Computation*, 182(1):947–954, 2006. cited By 9.

[204] L. Wang, Q.K. Pan, and M.F. Tasgetiren. Minimizing the total flow time in a flow shop with blocking by using hybridm harmony search algorithms. *Expert Systems with Applications*, 37(12):7929–7936, 2010.

[205] S.-Y. Wang, L. Wang, M. Liu, and Y. Xu. An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *International Journal of Production Economics*, 145(1):387–396, 2013.

[206] J.P. Watson, L. Barbulescu, L.D. Whitley, and A.E. Howe. Contrasting structured and random permutation flow-shop scheduling problems: Search-space topology and algorithm performance. *INFORMS Journal on Computing*, 14(2):98–123, 2002.

[207] H.-S. Woo and D.-S. Yim. A heuristic algorithm for mean flowtime objective in flowshop scheduling. *Computers & Operations Research*, 25(3):175–182, 1998.

[208] Z. Xie, C. Zhang, X. Shao, W. Lin, and H. Zhu. An effective hybrid teaching-learning-based optimization algorithm for permutation flow shop scheduling problem. *Advances in Engineering Software*, 77:35–47, 2014.

[209] B. Yagmahan and M.M. Yenisey. Ant colony optimization for multi-objective flow shop scheduling problem. *Computers and Industrial Engineering*, 54(3):411–420, 2008.

[210] K.-C. Ying and C.-J. Liao. An ant colony system for permutation flow-shop sequencing. *Computers & Operations Research*, 31(5):791–801, 2004.

[211] K.-C. Ying and S.-W. Lin. A high-performing constructive heuristic for minimizing makespan in permutation flowshops. *Journal of Industrial and Production Engineering*, 30(6):355–362, 2013.

[212] S.H. Zanakis, J.R. Evans, and A.A. Vazacopoulos. Heuristic methods and applications: A categorized survey. *European Journal of Operational Research*, 43(1):88–110, 1989.

[213] S.H. Zegordi, K. Itoh, and T. Enkawa. A knowledgeable simulated annealing scheme for the early/tardy flow shop scheduling problem. *International Journal of Production Research*, 33(5):1449–1466, 1995.

[214] C. Zhang, J. Ning, and D. Ouyang. A hybrid alternate two phases particle swarm optimization algorithm for flow shop scheduling problem. *Computers and Industrial Engineering*, 58(1):1–11, 2010.

[215] C. Zhang and J. Sun. An alternate two phases particle swarm optimization algorithm for flow shop scheduling problem. *Expert Systems with Applications*, 36(3 PART 1):5162–5167, 2009.

[216] C. Zhang, J. Sun, X. Zhu, and Q. Yang. An improved particle swarm optimization algorithm for flowshop scheduling problem. *Information Processing Letters*, 108(4):204–209, 2008.

[217] J. Zhang, C. Zhang, and S. Liang. The circular discrete particle swarm optimization algorithm for flow shop scheduling problem. *Expert Systems with Applications*, 37(8):5827–5834, 2010.

[218] L. Zhang and J. Wu. A pso-based hybrid metaheuristic for permutation flowshop scheduling problems. *The Scientific World Journal*, 2014, 2014.

[219] X. Zhang and S. Van De Velde. Approximation algorithms for the parallel flow shop problem. *European Journal of Operational Research*, 216(3):544–552, 2012.

[220] Y. Zhang, X. Li, and Q. Wang. Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. *European Journal of Operational Research*, 196(3):869–876, 2009.

[221] T. Zheng and M. Yamashiro. Solving flow shop scheduling problems by quantum differential evolutionary algorithm. *International Journal of Advanced Manufacturing Technology*, 49(5-8):643–662, 2010.

[222] G.I. Zobolas, C.D. Tarantilis, and G. Ioannou. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research*, 36(4):1249–1267, 2009.

[223] G. Zäpfel, R. Braune, and M. Bögl. *Metaheuristic Search Concepts*. Springer, 2010.