UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ALEXANDER JAVIER BENAVIDES ROJAS

Heuristics for Flow Shop Scheduling: Considering Non-Permutation Schedules and a Heterogeneous Workforce

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Prof. Dr. rer. nat. Marcus Ritt
Advisor

Prof. Dr. Cristóbal Miralles
Coadvisor

Porto Alegre, December 2015

CIP – CATALOGING-IN-PUBLICATION

Benavides Rojas, Alexander Javier

Heuristics for Flow Shop Scheduling: Considering Non-Permutation Schedules and a Heterogeneous Workforce / Alexander Javier Benavides Rojas. – Porto Alegre: PPGC da UFRGS, 2015.

189 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2015. Advisor: Marcus Ritt; Coadvisor: Cristóbal Miralles.

1. Heuristics. 2. Scheduling. 3. Flow shop. 4. Non-permutation. 5. Heterogeneous workers. I. Ritt, Marcus. II. Miralles, Cristóbal. III. Heuristics for Flow Shop Scheduling: Considering Non-Permutation Schedules and a Heterogeneous Workforce.

A mi hija Manuela, nueva razón y alegría de mi vida. A mis padres Carmen y Oscar, y a mi bisabuelo Manuel, quienes me enseñaron lo que significa la perseverancia y el trabajo duro.

ACKNOWLEDGMENTS

This work is supported by CAPES-Brazil and MEC-Spain (project CAPES-DGU258-12/PHB-0012-PC), by Petrobras (project 6741/PRH-PB217), by CNPq (project 478847/2013-0), and by the National Center for Supercomputing at UFRGS.

Quiero expresar en estas líneas mi gratitud y reconocimiento a todas las personas que hicieron posible esta tesis.

En primer lugar, me gustaría dar las gracias a mis a mis orientadores, Marcus Ritt y Cristóbal Miralles, por la paciencia, la guia y el apoyo brindados durante estos años. Mi investigación no habría llegado tan lejos sin ellos. Especialmente agradezco a Marcus, por trabajar codo a codo conmigo en muchas oportunidades para mejorar los resultados y lograr las publicaciones.

Agradezco también a los profesores Luciana Buriol, Rubén Ruiz, Marcelo Nagano, y Paulo França, por sus invaluables comentarios durante la evaluación de esta tesis. Especialmente agradezco a Luciana, por la oportunidad que me dio cuando orientó mi maestría, y por después acompañar de lejos mi crecimiento como investigador científico.

También le debo mucho a innumerables amigos que conocí durante estos años: funcionarios, profesores y estudiantes de la UFRGS y miembros de la Comunidad Salesiana de Porto Alegre. Agradezco especialmente a doña Graça Fuhrmann, Padre Marcio Lacoski, Lucas Volpatto, Mario Machado, Rafael Borges, Rodrigo Wilkens, Fernando Stefanello, Arton Dorneles, Ali Karaali, y Aasim Khurshid, por su apoyo y amistad desinteresada.

Finalmente, quiero expresar el amor y gratitud que le tengo a mi grande familia. Agradezco a mis padres, Oscar y Carmen, mi hermano, Eduardo, mi hermana Boni, y a mis abuelos, tíos, y primos, por el amor, los ánimos y el apoyo que me dieron en los momentos difíciles. Y en especial, quiero agradecer con todo mi amor a mi esposa, Nicole, por la paciencia que me tiene y por los momentos felices que trae a mi vida constantemente.

HEURISTICS FOR FLOW SHOP SCHEDULING: CONSIDERING NON-PERMUTATION SCHEDULES AND A HETEROGENEOUS WORKFORCE

ABSTRACT

The flow shop scheduling problem (or FSSP) is a very common model of production systems that is well studied in the literature. However, almost all the literature focuses on the permutation FSSP, disregarding optimal and near optimal solutions that are non-permutation schedules. Besides, common practice standardizes the processing times of each operation, even when those times may vary depending on different capabilities of the machine operators, whose diversity must be considered in the scheduling process when it is significant, e.g., in Sheltered Work centers for Disabled (SWDs). In this thesis, we propose methods to solve the non-permutation FSSP, using the same time and effort as state-of-the-art methods for the permutation FSSP, and producing non-permutation schedules with better quality than permutation and non-permutation schedules produced by state-of-the-art methods. We also propose methods to solve the combined heterogeneous workforce assignment and flow shop scheduling problem (or Het-FSSP), producing solutions that compensate the different capabilities and disabilities of the workers with minor or null losses in the productivity objectives. Moreover, the heterogeneous workforce assignment may be integrated into other shop scheduling models, as we did with the heterogeneous workforce assignment and job shop scheduling problem (or Het-JSSP) with similar results.

Keywords: Heuristics, scheduling, flow shop, non-permutation, heterogeneous workers.

HEURÍSTICAS PARA ESCALONAMENTO EM FLOW SHOPS: CONSIDERANDO ESCALONAMENTOS NÃO-PERMUTACIONAIS E TRABALHADORES HETEROGÊNEOS

RESUMO

O problema de escalonamento num *flow shop* (ou *flow shop scheduling problem*, FSSP) é um modelo de sistemas de produção muito comum que é bem estudado na literatura. No entanto, quase toda a literatura foca-se em escalonamentos permutacionais, desconsiderando soluções ótimas e quase ótimas que são escalonamentos não-permutacionais. Além disso, a prática comum padroniza os tempos de processamento de cada operação, mesmo que estes tempos variem dependendo das diferentes capacidades dos operadores das máquinas, cuja diversidade deve ser considerada no processo de escalonamento quando seja significativa, e.g., em centros de emprego para deficientes (CEDs). Nesta tese, propomos métodos para resolver o FSSP não-permutacional, usando o mesmo tempo e esforço que os métodos do estado da arte usam para o FSSP permutacional, e produzindo escalonamentos não-permutacionais com melhor qualidade do que escalonamentos permutacionais e não-permutacionais produzidos por métodos do estado da arte. Também propomos métodos para resolver o problema combinado de designação de trabalhadores heterogêneos e escalonamento de tarefas num flow shop (ou *heterogeneous workforce assignment and flow shop scheduling problem*, Het-FSSP), produzindo soluções que compensam as diferentes capacidades e deficiências dos trabalhadores com pequenas perdas nos objetivos da produção. Além do mais, a designação de trabalhadores heterogêneos pode ser integrada em outros problemas de escalonamento, como fizemos com o problema combinado de designação de trabalhadores heterogêneos e escalonamento de tarefas num job shop (ou *heterogeneous workforce assignment and job shop scheduling problem*, Het-JSSP).

Palavras-chave: Heurísticas, escalonamento de tarefas, flow shop, não-permutacional, trabalhadores heterogêneos.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ACRONYMS

SWD	Sheltered Work centers for Disabled	20
Optimization Criteria:		
$C_{\max}$	Makespan or maximum completion time	30
C_{sum}	Total completion time	30
Scheduling problems:		
FSSP	Flow Shop Scheduling Problem	31
JSSP	Job Shop Scheduling Problem	31
OSSP	Open Shop Scheduling Problem	31
Het-FSSP	Heterogeneous workforce assignment and Flow Shop Scheduling Problem	135
Het-JSSP	Heterogeneous workforce assignment and Job Shop Scheduling Problem	135
PFSISP	Permutation Flow Shop Insertion and Scheduling Problem	161
HPFSISP	Hybrid Permutation Flow Shop Insertion and Scheduling Problem	161
Heuristic methods:		
BRN	Best-improvement Reduced-neighbourhood local search for the Non-permutation FSSP	112
IGA	Iterated Greedy Algorithm	76
ILS	Iterated Local Search	71
NEH	Constructive heuristic of Nawaz, Enscore & Ham (1983)	44
NFS	Constructive heuristic for the Non-permutation FSSP	88
SBP	Shifting Bottleneck Procedure	57

CONTENTS

Part I

Introduction

1 INTRODUCTION

This chapter introduces our research, and presents its motivation, objectives, contributions, and scope. In the last section it describes the organization of the thesis.

1.1 Motivation

1.1.1 Considering non-permutation schedules in flow shops

Scheduling is a decision-making process used in many manufacturing and service industries. It consists in allocating limited resources to activities that must be accomplished. Resources may be machines in an assembly plant, CPU, memory and I/O devices in a computer system, runways at an airport, crews at a repair-shop, etc. Activities may be manufacturing process operations, computer programs, landings and take-offs, and so on. The optimal allocation will optimize the objectives of the enterprise, reducing time and costs. Thus, efficient scheduling is critical for the industries to survive in the competitive business market.

Scheduling problems are considered among the hardest combinatorial optimization problems. The flow shop scheduling problem (or FSSP) is one of the most studied scheduling problems in the last six decades. In a flow shop, there are n jobs that must be processed on m machines following the same machine sequence. Each job J_j has a specific processing time p_{ij} on each machine M_i. A schedule is an allocation of the jobs to the machines over the time. The FSSP consists in finding an schedule such that some criteria are optimized. Minimizing the completion time of the last job on the last machine (or makespan) and minimizing the sum of the completion times of all the jobs on the last machine (or total completion time) are the two most commonly studied optimization criteria in the literature.

A permutation schedule has the same processing order of jobs for all machines, and a non-permutation schedule may have a different processing order of jobs for some machines. The non-permutation (or general) FSSP allows permutation and non-permutation schedules, while the permutation FSSP only allows permutation schedules. The consideration of only one permutation of jobs for the processing order of all the machines, instead of a different permutation of jobs for each machine, reduces substantially the number of candidate schedules, making the FSSP easier to approach, but it also eliminates the best candidate schedules, even eliminating all optimal schedules in some cases. Thus, the reduction of the FSSP to permutation schedules pays the price of significantly inferior solutions, and the consideration of non-permutation schedules is necessary to obtain optimal or near-optimal solutions for most of the FSSP instances (TANDON; CUMMINGS; LEVANU, 1991). Nevertheless, almost all the literature on the FSSP focuses on the permutation FSSP, as Chapter 3 shows.

We have observed that optimal or near-optimal non-permutation schedules result in many cases from limited job reordering between machines, conserving almost the same permutation of jobs for the processing order of all the machines. Thus, we want to use this observation to reduce the number of candidate schedules without eliminating optimal or near-optimal non-permutation schedules, in order to find near-optimal non-permutation schedules with the same time and effort that other researchers use to find the best possible permutation schedules.

1.1.2 Considering disabilities as workforce heterogeneity in scheduling

A disability does not influence the person's general capacity to work, it only restrains his capability to perform certain tasks. Even though some disabled persons with the right opportunities, adaptations and support are able to make a major contribution at all levels of the economy and society, a disabled person is not always considered suitable for employment. The International Labour Organization estimates that the number of persons with disabilities is about one billion, or 15 per cent, of the world's population, out of whom between 785 and 975 million are estimated to be of working age, and usually suffering much higher unemployment rates and economic inactivity compared to non-disabled persons (ILO, 2012).

Many governments have implemented different policies for the vocational rehabilitation and integration of persons with disabilities into the labour market, such as reserving a percentage of jobs in companies, or creating Sheltered Work centers for Disabled (SWDs). SWDs facilitate jobs for disabled workers, both as stable workplaces and as transition workplaces for training and rehabilitation towards complete social employment integration. This model of socio-labour integration removes the traditional stereotype that considers disabled people as unable to develop continuous professional work, by overcoming certain prejudices, and by considering the workforce as heterogeneous as it actually is, taking into account their limitations and aiming to evolve positively their capabilities and capacities (MIRALLES et al., 2010).

Although SWDs receive some institutional support, they compete in real labour markets where they need to be efficient and competitive. Besides their survival, SWDs also must be able to grow, to give better conditions to their current workers, and to promote new job opportunities for persons with higher levels of disability.

Most of the management science approaches and tools standardize the processing time of every operation assuming that any worker would perform the same operation in the same average time. This assumption is not realistic and may cause serious planning and control problems. For instance, the master schedule definition and the workplace assignments become harder when the production manager has to cope with rigid information systems that disregard the heterogeneity of the workforce's skills. Knowing the clear differences between the reality and the scenarios defined in the information systems, the production manager may try to compensate manually the existing deviations of the workers performance with rules of thumb or with a not very efficient "worst case" scenario.

These rectifications result in the adoption of suboptimal solutions and in the loss of the control and validation of the corresponding indicators, due to the differences between the modeled scenarios and the reality, to the aggregated effects from the planning compensations, and because of rush adaptations made by the workers themselves.

The correct and more reliable approach would be to assume the workers heterogeneity, to estimate a priori the deviations, and to feed the planning/scheduling process with more realistic data that introduces this heterogeneity in the information systems.

The term "machine" in the flow shop literature usually refers to a work center in which the process performed may be automated or manual. In a flow shop environment where the workers are heterogeneous, the processing time of an operation performed on a machine will also depend on the worker assigned to operate it. Furthermore, in such a scenario, the complete solution of the problem consists of two elements: as usual, a schedule of jobs that optimizes a given objective function, and, additionally, the optimal allocation of workers to work centers (or machines) that helps to get the best possible solution.

The scheduling and rescheduling caused by worker rotation, turnover and absenteeism, demand a rapid response of production managers, mainly in SWDs where periodical health and psychological support are mandatory. Therefore, sophisticated solution methods are necessary in order to obtain an optimal worker allocation and the corresponding optimal schedule in reasonable computation time.

Section 3.5 presents examples in the literature that successfully introduce human diversity in the planning and scheduling processes of productive systems. To the best of our knowledge, the joint problem of assigning diverse workers when scheduling jobs in flow shop systems has not been addressed in the literature. Thus, we want to contribute to narrow the gap between research and practice with our work in this problem.

1.2 Research objective and contribution

1.2.1 Objective of this research

The main focus of this research is to propose heuristic methods to solve the flow shop scheduling problem, considering non-permutation schedules and heterogeneous workers, in order to reduce the makespan and the total completion time.

Specific objectives of this research include:

- Identify the theoretical concepts and state-of-the-art literature that are relevant to the non-permutation flow shop scheduling problem and to the integration of the heterogeneous workforce assignment problem and the flow shop scheduling problem.
- Propose heuristics to solve the non-permutation FSSP with the makespan criterion with the same computational effort and better result than state-of-the-art methods for the permutation FSSP.
- Propose heuristics to solve the non-permutation FSSP with the total completion time criterion with the same computational effort and better result than state-of-the-art methods for the permutation FSSP.
- Formulate the integrated problem of heterogeneous workforce assignment and flow shop scheduling, together with a mathematical model and a benchmark of instances.
- Propose heuristics to solve the integrated problem of heterogeneous workforce assignment and flow shop scheduling.

- Generalize the method used to integrate the heterogeneous workforce assignment and flow shop scheduling problem for other scheduling problems, e.g., for the job shop scheduling problem.

1.2.2 Contribution of this research

This thesis presents two major contributions. The first contribution is the proposal and comparison of heuristics to solve flow shop scheduling problems, considering non-permutation schedules, using the same time and effort that state-of-the-art methods use to solve simplified permutation flow shop scheduling problems, and producing non-permutation schedules with better quality than permutation and non-permutation schedules produced by state-of-the-art methods. The second contribution is the integration of heterogeneous workers and the problem of their assignment into the shop scheduling problems, with the proposal of new models, test instances and heuristics for the heterogeneous workforce assignment and flow shop scheduling problem (or Het-FSSP) and the heterogeneous workforce assignment and job shop scheduling problem (or Het-JSSP).

1.2.3 Contribution to the literature

The publications to be considered for the partial fulfillment of the requirements for the degree of Ph.D. are:

- BENAVIDES, A. J.; RITT, M.; MIRALLES, C. Flow shop scheduling with heterogeneous workers. *European Journal of Operational Research*, Elsevier, v. 237, n. 2, p. 713–720, 2014.

  The European Journal of Operational Research (EJOR) has an Impact Factor of 2.358 and a 5-Year Impact Factor of 2.911, and it is classified by the CAPES WebQualis as **A1** for the computer science area. According to Google Scholar, this article has been already cited seven times in the first two years of its publication. The contributions of this article are explained in Chapers 8 and 9.

- BENAVIDES, A. J.; RITT, M. Two simple and effective heuristics for minimizing the makespan in non-permutation flow shops. *Computers & Operations Research*, Elsevier, v. 66, p. 160–169, 2016.

  The Computers & Operations Research (COR) journal has an Impact Factor of 1.861 and a 5-Year Impact Factor of 2.454, and it is classified by the CAPES WebQualis as **A1** for the computer science area. The contributions of this article are explained in Chaper 6.

- BENAVIDES, A. J.; RITT, M. Iterated local search heuristics for minimizing total completion time in permutation and non-permutation flow shops. In: *Twenty-Fifth International Conference on Automated Planning and Scheduling*. Jerusalem, Israel: AAAI Publications, 2015. p. 34–41.

  The International Conference on Automated Planning and Scheduling (ICAPS) has an H-index of 49 and a 5-Year H-index of 25, and it is classified by the CAPES Qualis as **A2** for the computer science area. The contributions of this article are explained in Chaper 5.

Another publication, which was presented in an international workshop not classified by the CAPES Qualis, and whose contributions are explained in Chapter 10, is:

- BENAVIDES, A. J.; RITT, M.; MIRALLES, C. Heterogeneous workforce in job shop scheduling. In: *VIII ALIO/EURO Workshop on Applied Combinatorial Optimization*. Montevideo, Uruguay: ALIO/EURO, 2014.

Another paper with the contributions explained in Chapter 7 is being prepared to be submited to the European Journal of Operational Research (EJOR).

1.3 Scope of the thesis

There are several issues that could be considered in scheduling and worker assignment problems. We had to limit some of them for different reasons. Here, we list some limitations and their implications.

We initially wanted to integrate the heterogeneous workforce assignment problem with the three most important shop scheduling problems (flow shop, job shop, and open shop). We could also consider other extended shop scheduling models that allow reentrant or missing operations, or that allow parallel machines in some stages of the shop like the hybrid flow shop and the flexible job shop. But this ambitious goals were mainly limited to the flow shop scheduling problem, with a small extension to the job shop scheduling problem.

There are different optimization criteria for scheduling problems in the literature, such as minimizing flowtime, lateness, or tardiness. But many of them require additional data that is not available with the commonly used benchmarks, such as release dates, due dates, or weight factors for the jobs. Thus, we limit our research to the minimization of the makespan and the total completion time criteria, because they only need the processing times of the operations for their calculations, and they are the most commonly studied in the literature.

The literature presents different approaches to deal with heterogeneous workers, such as their classification into categories (newbie, regular, expert), or with proportionally related execution times. There are also different approaches in the case of inclusion of workers with disabilities, such as inserting a low percentage of heterogeneous workers, assigning them into working cells (or groups), and taking into account their inability to operate a subset of the machines (for therapeutic or strategic reasons). We limit our research to a model that assumes personalized operation processing times for each worker and contemplates their inabilities, because we believe that it is general enough to be adapted to other models.

1.4 Overview of the thesis

This thesis is organized in four parts.

The first part introduces the flow shop scheduling problem and has three chapters. This chapter introduces and describes the motivation, objectives, contribution, and context of this research. Chapter 2 describes the shop scheduling problems, with their characteristics and formulations, focusing on the flow shop scheduling problem and its two variations: permutation and non-permutation. It also presents theoretical concepts of shop scheduling that may be applied to the non-permutation flow shop scheduling problem. Chapter 3 presents a brief review of the literature on the permutation and the non-permutation flow shop scheduling problem. It also reviews the relevant literature on the job shop scheduling problem and on the inclusion of the diversity of workers in production planning.

The second part presents heuristics for the non-permutation FSSP and has four chapters. Chapter 4 explores the theoretical differences between the permutation and non-permutation FSSP, and introduces the concept of non-permutation insertion of a job with anticipation or delay that is used in the other three chapters of this part. We study the practical differences between the permutation and the non-permutation FSSP with the total completion time criterion in Chapter 5, and with the makespan criterion in Chapter 6. To this end, we study in both chapters the degree of effort needed to find good non-permutation schedules with the proposed iterated local search and iterated greedy heuristics, we compare the buffer requirements for the resulting permutation and non-permutation schedules, and we assess the amount of job reordering in the resulting non-permutation schedules. In Chapter 7, we propose a novel permutation representation for non-permutation schedules, an acceleration method to calculate the makespan that results of a job insertion in a constructive heuristic for non-permutation schedules, and two local searches for non-permutation schedules. The proposed heuristics are embedded in iterated greedy algorithms to evaluate their effectiveness in finding good non-permutation schedules.

The third part present heuristics for the heterogeneous workforce assignment in scheduling problems and has four chapters. Chapter 8 introduces the combined problem of flow shop scheduling and worker assignment with a didactic example. It also mathematically defines the heterogeneous workforce assignment and flow shop scheduling problem (or Het-FSSP) and the heterogeneous workforce assignment and job shop scheduling problem (or Het-JSSP), discusses their complexity, and introduces new instances for them. We propose a scatter search with path relinking for solving the Het-FSSP in Chapter 9 and a multi-start local search heuristic for solving the Het-JSSP in Chapter 10. In both chapters we describe the components of the proposed methods and study their effectiveness. Chapter 11 briefly describes a related research on the inclusion of one or two heterogeneous workers in a flow shop that was performed by a fellow Master's degree student.

The final part, Chapter 12, presents the concluding remarks of this research, and the possible routes for research in the near future.

2 THEORETICAL CONCEPTS RELATED TO FLOW SHOP SCHEDULING

This chapter describes the shop scheduling problems and their characteristics, focusing on the flow shop scheduling problem and its variations. With this aim, the characteristics of the scheduling problems are introduced in Section 2.1, and formulations for the main shop scheduling problems are presented in Section 2.2. As the general flow shop scheduling problem may be considered a special case of the job shop scheduling problem, Section 2.3 describes theoretical concepts that apply to both problems, such as active and semi-active schedules, disjunctive graph representation, and critical paths, and Section 2.4 shows representations for job shop schedules that can symbolize non-permutation flow shop schedules. Section 2.5 introduces theoretical concepts of multi-objective optimization.

2.1 Shop Scheduling Characterization and Classification

In a general scheduling problem, a given set of jobs must be processed on a set of machines, and the problem consists in finding an efficient schedule according to one or more optimization criteria. An schedule is an allocation of jobs to machines over time. The scheduling problems usually aim to optimize production time and thus reduce costs. General characteristics of the scheduling problems in this thesis are:

- Each job is independent of the others.
- Each job consists of a set of operations.
- Each operation is processed on one machine for a known processing time.
- Each job is processed no more than once on any machine, i.e., different operations of the same job can not be processed on the same machine.
- Each job is processed only on one machine at a time, i.e., no parallel operations of the same job are allowed.

- Each machine can process at most one operation at a time, i.e., operations of different jobs can not be processed on the same machine at the same time.

Besides these, many parameters define a specific scheduling problem, for example machine characteristics, job characteristics, flow pattern, optimization criteria, scheduling environment. Graham et al. (1979) introduced a classification of scheduling problems. The characteristics of each scheduling problem are described by a three-field notation $\alpha|\beta|\gamma$. The α field describes the machine environment, the β field describes jobs and constraints, and the γ field denotes an optimality criterion or objective function.

To explain this classification, we must define first some data associated to jobs. We refer to jobs with the subscript j and to machines with the subscript i.

A processing time (p_{ij}) is the time that machine M_i will spend to process job J_j. The subscript i is omitted when machines are homogeneous.

A release date (r_j) is the earliest time at which job J_j can start its processing.

A due date (d_j) is the latest time for the completion of job J_j without penalization. A later completion may be acceptable, but will be penalized. A hard deadline $(\bar{d}_j)$ is a due date that must be respected.

A weight (w_j) is a priority factor that reflects the importance of a job J_j.

2.1.1 Machine Environment (α field)

Machines are classified as parallel or dedicated. Parallel machines have the same capabilities, for example, multiple CPUs in a server or a grid system. In contrast, a dedicated machine is specialized for the execution of a single task, for example cutting, sewing, ironing machines in a tailor's shop. The α field contains a single parameter $\alpha \in \{1, Pm, Qm, Rm, Fm, Jm, Om\}$, where m indicates the number of machines (when m is omitted, any number of machines is permited). Next, we explain the machine environments.

Single machine (1)**.** In this case, there is only one machine that will process every job only once.

Identical parallel machines (Pm)**.** There are m identical machines processing in parallel at the same speed. Each job J_j has a single operation that may be processed on any of the m machines.

Uniform parallel machines (Qm). There are m machines processing in parallel, but they operate at different speeds. The speed of each machine M_i is determined by a factor s_i. The execution time is $p_{ij} = p_j/s_i$ where p_j is the standard processing time for job J_j.

Unrelated parallel machines (Rm). There are m machines processing in parallel, and there is no relation between their speeds. Thus, the execution time depends on the processed job and the corresponding machine.

Flow shop (Fm). There are m different dedicated machines. All the jobs follow the same processing order through the machines.

Job shop (Jm). There are m different dedicated machines. Each job has its own processing order through the machines.

Open shop (Om). There are m different dedicated machines. Each job must be processed on each machine exactly once. The processing order is arbitrary.

2.1.2 Job Characteristics (β field)

The description of jobs and constraints may contain multiple entries:

Preemption $(pmtn)$. If $pmtn$ is included in the β field, job preemption (splitting) is allowed. Jobs can be preempted and later resumed, possibly on a different machine (in a parallel machine environment). Otherwise, if $pmtn$ is not included in the β field, once a job operation initiates on a given machine, it must be processed to completion without interruption.

No wait $(no{-}wait)$. Indicates that jobs are not allowed to wait between operations on successive machines. If $no{-}wait$ is not specified, the jobs can wait between operations for an unlimited time.

Precedence constraints $(prec, chains, intree, outtree, tree, sp{-}graph)$. A precedence constraint indicates that some jobs must be completed before certain other jobs can initiate their processing. Precedence constraints can be represented as a directed acyclic graph, where each node represents a job and each arc indicates that the job at the tail must be processed before the job at the head. The shape of the directed acyclic graph that represents the precedence constraints is described by different entries. The most general constraint, $prec$, indicates a directed acyclic graph. $chains$ indicates that every job has in-degree and out-degree of at most 1. $intree$ indicates that every job in the graph has out-degree of at most 1. $outtree$ indicates that every job in the graph has in-degree of at most

1. $tree$ indicates an $intree$ or an $outtree$. $sp{-}graph$ indicates that the graph is a series-parallel graph. If none of these entries is specified, jobs are not subject to precedence constraints.

Release dates (r_j)**.** This entry indicates that every job has a different release time. If it is not specified, the release time of every job is zero.

Processing times (p_j)**.** This entry indicates that the processing times of each job are restricted. $p_j = p$ indicates that all jobs have processing times equal to p units. $\underline{p} \leq p_j \leq \bar{p}$ indicates that no p_j is less than $\underline{p}$ or greater than $\bar{p}$. If it is not specified, jobs have arbitrary processing times.

Deadlines $(\bar{d})$**.** This entry indicates that deadlines are imposed on the jobs. If it is not specified, no deadlines are assumed, however, due dates may be defined as part of the optimality criteria.

Number of operations in jobs (n_j)**.** This entry applies only to job shop problems. The inequality $n_j \leq k$ indicates that each job is limited to at most k operations. If it is not present, the number of operations is unrestricted.

2.1.3 Optimality criterion (γ field)

The optimality criterion is the cost function that must be minimized or maximized. An optimality criterion is called *regular* if it is a non-decreasing function of the completion times of the jobs. The γ field represents the optimality criterion. The following entries are regular optimality criteria:

Makespan $(C_{\max})$**.** The goal is to minimize the makespan or maximum completion time over all jobs $C_{\max} = \max\{C_1, \ldots, C_n\}$, where C_j is the completion time of job J_j.

Total completion time $(\sum w_j C_j, \sum C_j, C_{\text{sum}})$**.** The goal is to minimize the total weighted completion time (denoted by $\sum w_j C_j$) or the total unweighted completion time (denoted by $\sum C_j$ or by C_{sum}).

Total flowtime $(\sum w_j F_j, \sum F_j)$**.** The flowtime is defined as $F_j = C_j - r_j$. The goal is to minimize the total weighted flowtime (denoted by $\sum w_j F_j$) or the total unweighted flowtime (denoted by $\sum F_j$). When all release dates are $r_j = 0$, the flowtime criterion is equivalent to the completion time criterion.

Maximum lateness $(L_{\max})$**.** The goal is to minimize the maximum lateness $L_{\max} = \max\{L_1, \ldots, L_n\}$, where $L_j = C_j - d_j$ is the lateness of job J_j.

Total tardiness ($\sum w_jT_j, \sum T_j$). The goal is to minimize the total weighted tardiness (denoted by $\sum w_jT_j$) or the total unweighted tardiness (denoted by $\sum T_j$), where $T_j = \max\{C_j - d_j, 0\}$ is the tardiness of a job J_j.

Number of tardy jobs ($\sum w_jU_j, \sum U_j$). The goal is to minimize the total weighted number of jobs that complete late (denoted by $\sum w_jU_j$) or the total unweighted number of jobs that complete late (denoted by $\sum U_j$), where $U_j = 1$ if $C_j > d_j$ and 0 otherwise.

To illustrate the $\alpha|\beta|\gamma$ notation, we present some examples. The problem $1|r_j, d_j|L_{\max}$ is to find a preemptive schedule on one machine for a set of jobs with a given release times r_j and a due date d_j that minimizes the maximum lateness $L_{\max}$. The problem $R|prec|\sum w_jC_j$ minimizes the total weighted completion time on a variable number of unrelated parallel machines, where the jobs are subject to precedence constraints. The problem $J3|p_{ij} = 1|C_{\max}$ is to minimize the maximum completion time of jobs with unit processing times in a job shop with three machines.

2.2 Formulations for the Main Shop Scheduling Problems

Besides the models of single and parallel machines, scheduling problems in the production industry usually involve different machines for different purposes. Shop scheduling problems are well known to model environments with dedicated machines. We can separate them based on the flow pattern that jobs follow through the machines. The most studied and well known models are the Flow Shop Scheduling Problem (FSSP), the Job Shop Scheduling Problem (JSSP) and the Open Shop Scheduling Problem (OSSP). Next we explain them.

2.2.1 The Flow Shop Scheduling Problem

The flow shop scheduling problem ($F||C_{\max}$) is defined as follows. Given n jobs to be processed on m machines in the same order, a schedule must be found, such that the maximum completion time is minimized.

Formally, let x_{ij} be the starting time of job J_j on machine M_i, let variable $y_{ijj'} \in \{0, 1\}$ indicate that job J_j, precedes job $J_{j'}$ on machine M_i, and let M be a large constant, then an integer linear program for the FSSP is

$$
\begin{aligned}
\textbf{min.} \quad & C_{\max}, && && (2.1)\\
\textbf{s.t.} \quad & x_{mj} + p_{mj} \le C_{\max}, && \forall j \in [n], && (2.2)\\
& x_{ij} + p_{ij} \le x_{(i+1)j}, && \forall i \in [m-1], j \in [n], && (2.3)\\
& x_{ij} + p_{ij} \le x_{ij'} + M(1 - y_{ijj'}), && \forall i \in [m], j \neq j' \in [n], && (2.4)\\
& y_{ijj'} + y_{ij'j} = 1, && \forall i \in [m], j \neq j' \in [n], && (2.5)\\
& x_{ij} \ge 0, && \forall i \in [m], j \in [n], && (2.6)\\
& y_{ijj'} \in \{0,1\}, && \forall i \in [m], j \neq j' \in [n]. && (2.7)
\end{aligned}
$$

In this formulation constraints (2.2) define the makespan, constraints (2.3) require a job to finish its process on a machine before starting on the following one, constraints (2.4) require the jobs J_j and J'_j to be processed in the order defined by the variables $y_{ijj'}$ for each machine M_i, and constraint (2.5) forces a linear ordering of the jobs on all machines.

It is common in the literature to limit the solutions to schedules that process all jobs in the same order on all machines. This variation of the problem is called Permutation Flow Shop Scheduling Problem (PFSSP). An integer linear program for the PFSSP reduces the number of constraints and variables by replacing $y_{ijj'}$ by $y_{jj'}$ for all machines as follows

$$
\begin{aligned}
\textbf{min.} \quad & C_{\max}, && && (2.8)\\
\textbf{s.t.} \quad & x_{mj} + p_{mj} \le C_{\max}, && \forall j \in [n], && (2.9)\\
& x_{ij} + p_{ij} \le x_{(i+1)j}, && \forall i \in [m-1], j \in [n], && (2.10)\\
& x_{ij} + p_{ij} \le x_{ij'} + M(1 - y_{jj'}), && \forall i \in [m], j \neq j' \in [n], && (2.11)\\
& y_{jj'} + y_{j'j} = 1, && \forall j \neq j' \in [n], && (2.12)\\
& x_{ij} \ge 0, && \forall i \in [m], j \in [n], && (2.13)\\
& y_{jj'} \in \{0,1\}, && \forall j \neq j' \in [n]. && (2.14)
\end{aligned}
$$

We discuss and compare the use of permutation and non-permutation schedules in Chapter 4.

2.2.2 The Job Shop Scheduling Problem

The job shop scheduling problem ($J||C_{\max}$) is defined as follows. Given n jobs to be processed on m machines in a different order for each job, a schedule must be found, such that the maximum completion time is minimized.

Formally, let $R_j = \{(i,i')|o_{ij} \prec o_{i'j}\}$ be the set of precedence relations of all pairs of consecutive operations o_{ij}, $o_{i'j}$ that belong to job J_j, let x_{ij} be the starting time of job J_j on machine M_i, let variable $y_{ijj'} \in \{0,1\}$ indicate that job J_j precedes job $J_{j'}$ on machine M_i, and let M be a large constant, then an integer linear program for the JSSP is

$$\begin{aligned}
\textbf{min.} \quad & C_{\max}, && && (2.15)\\
\textbf{s.t.} \quad & x_{mj} + p_{mj} \le C_{\max}, && \forall j \in [n], && (2.16)\\
& x_{ij} + p_{ij} \le x_{i',j}, && \forall (i,i') \in R_j, j \in [n], && (2.17)\\
& x_{ij} + p_{ij} \le x_{ij'} + M(1 - y_{ijj'}), && \forall i \in [m], j \neq j' \in [n], && (2.18)\\
& y_{ijj'} + y_{ij'j} = 1, && \forall i \in [m], j \neq j' \in [n], && (2.19)\\
& x_{ij} \ge 0, && \forall i \in [m], j \in [n], && (2.20)\\
& y_{ijj'} \in \{0,1\}, && \forall i \in [m], j \neq j' \in [n]. && (2.21)
\end{aligned}$$

In this formulation constraints (2.16) define the makespan, constraints (2.17) require a job to finish its processing on a machine before starting on the following one according to the precedence order of each job, constraints (2.18) require the jobs J_j and J_j' to be processed in the order defined by the variables $y_{ijj'}$ for each machine M_i, and constraint (2.19) forces a linear ordering of the jobs on all machines. For an n-jobs, m-machines job shop instance, there exist $n!^m$ possible solutions.

2.2.3 The Open Shop Scheduling Problem

The open shop scheduling problem($O||C_{\max}$) is defined as follows. Given n jobs to be processed on m machines without any specific order, a schedule must be found, such that the maximum completion time is minimized.

Formally, let x_{ij} be the starting time of job J_j on machine M_i, let variable $y_{ijj'} \in \{0,1\}$ indicate that job J_j precedes job $J_{j'}$ on machine M_i, let variable $z_{ii'j} \in \{0,1\}$ indicate that job J_j is processed on machine M_i before machine $M_{i'}$, and let M be a large constant, then an integer linear program for the OSSP is

$$\begin{aligned}
\textbf{min.} \quad & C_{\max}, && && (2.22)\\
\textbf{s.t.} \quad & x_{mj} + p_{mj} \le C_{\max}, && \forall j \in [n], && (2.23)\\
& x_{ij} + p_{ij} \le x_{ij'} + M(1 - y_{ijj'}), && \forall i \in [m], j \neq j' \in [n], && (2.24)\\
& y_{ijj'} + y_{ij'j} = 1, && \forall i \in [m], j \neq j' \in [n], && (2.25)\\
& x_{ij} + p_{ij} \le x_{i'j} + M(1 - z_{ii'j}), && \forall i \neq i' \in [m], j \in [n], && (2.26)\\
& z_{ii'j} + z_{i'ij} = 1, && \forall i \neq i' \in [m], j \in [n], && (2.27)
\end{aligned}$$

$$x_{ij} \geq 0, \qquad \forall i \in [m], j \in [n], \qquad (2.28)$$

$$y_{ijj'} \in \{0,1\}, \qquad \forall i \in [m], j \neq j' \in [n], \qquad (2.29)$$

$$z_{ii'j} \in \{0,1\}, \qquad \forall i \neq i' \in [m], j \in [n]. \qquad (2.30)$$

In this formulation constraints (2.23) define the makespan, constraints (2.24) require the jobs J_j and J'_j to be processed in the order defined by the variables $y_{ijj'}$ for each machine M_i, constraints (2.25) forces a linear ordering of the jobs on each machine, constraints (2.26) require each job J_j to be processed in the order defined by the variables $z_{ii'j}$ on machines M_i and $M_{i'}$, and constraints (2.27) forces a linear ordering of each job on all machines.

2.3 Other Theoretical Concepts

The general flow shop scheduling problem may be considered a special case of the job shop scheduling problem, where machine M_i must process the i-th operation of every job. For this reason, this section presents some theoretical concepts that apply to both problems.

2.3.1 Semi-active, active and non-delay schedules

A semi-active schedule has no operation that can start and finish earlier without changing the processing order or violating its feasibility. In other words, every operation is processed as early as the schedule commands. Figure 2.1a shows a Semi-active schedule. Creating an idle time on machine M_2 by delaying job J_2 would produce an example of a schedule that is not semi-active.

An active schedule has no operation that can start and finish earlier by changing the processing order without forcing another operation to start and finish later or violating its feasibility. In other words, there is no operation that fits into an earlier idle time. An active schedule is also semi-active, but not otherwise. Figure 2.1b shows an active schedule, because both operations on machine M_2 would be affected by a change in their processing order. If we change the processing order on machine M_2 of Figure 2.1a, only one operation is earlier and the other remains at the same time slot. Thus, Figure 2.1a shows a semi-active schedule that is not active.

A non-delay schedule has no idle time on any machine when an operation is waiting for processing. In other words, if a machine is free, it will start processing any available operation as soon as possible. A non-delay schedule is also active and

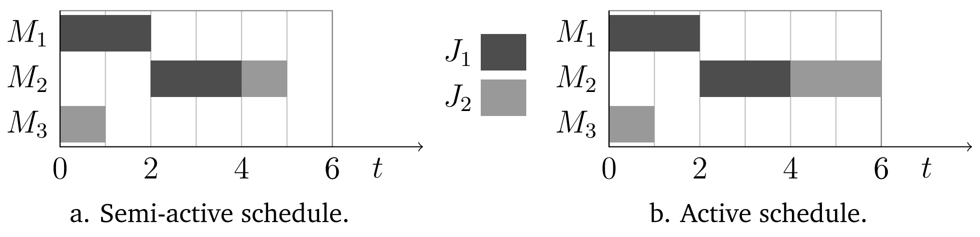


Figure 2.1: Semi-active and active schedules.

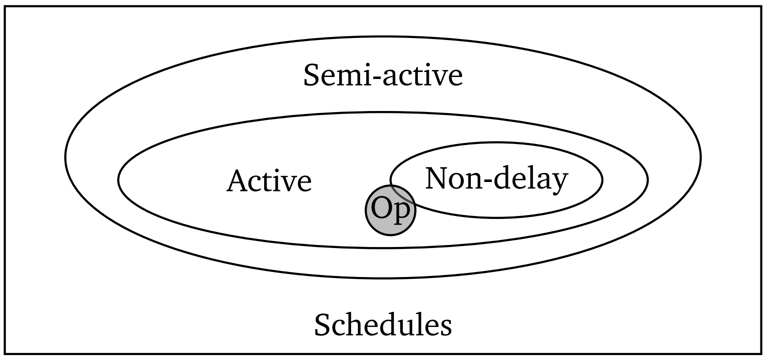


Figure 2.2: Venn diagram of classes of schedules. Set Op are optimal schedules for regular criteria.

hence semi-active, but not otherwise. Figure 2.1b shows an active schedule that is not non-delay, because there is an operation available to start processing on machine M_2 at time 1 that waits. An example of a non-delay schedule can be obtained by changing the processing order on machine M_2.

Figure 2.2 shows a Venn diagram of the different types of schedules. Optimal schedules for regular criteria are represented by the small grey set. All optimal schedules are active, but not every optimal schedule is non-delay.

2.3.2 A simple 3×3 job shop example

Table 2.1 introduces a simple instance of a JSSP given by Yamada & Nakano (1997). For each job's operation, it presents the required machine and the processing

Table 2.1: 3×3 instance of the JSSP.

Jobs	Operations		
	Machine (processing time)		
J_1	M_1 (3)	M_2 (3)	M_3 (3)
J_2	M_1 (2)	M_3 (3)	M_2 (4)
J_3	M_2 (3)	M_1 (2)	M_3 (1)

time. Each job has a different machine order. Usually the size of a shop scheduling instance is represented by $n \times m$, where n is the number of jobs and m is the number of machines. This instance must process three jobs on three machines, thus we refer to this instance as a 3×3 instance.

A solution for a JSSP instance is a job shop schedule and there are many ways to represent it. A Gantt chart is the most explanatory visual representation for a schedule. Figures 2.3 and 2.4 presents Gantt charts of an optimal solution for the 3×3 instance. A Gantt chart is a type of bar chart that illustrates in the x-axis the start and completion times for each operation. The y-axis may be arranged by jobs (Figure 2.3) to show their progress, or by machines (Figure 2.4) to show their occupation.

2.3.3 Disjunctive graph representation

A disjunctive graph $G = (V, C \cup D)$ formally describes a JSSP instance. Figure 2.5 presents a disjunctive graph for the 3×3 instance. A disjunctive graph comprises three sets:

The set of vertices (V) represents the set of all operations. Two artificial operations are added to this set: a *source* 0 and a *sink* $*$ that represent the begin and the end of the schedule. The vertex of operation o_{ij} is weighted with the processing time p_{ij} of job J_j on machine M_i.

The set of conjunctive arcs (C) represents the constraints of the given processing sequence for the operations through the machines. These arcs are represented by the arrows in Figure 2.5.

The set of disjunctive arcs (D) represents the constraints that require an order among operations to be processed on the same machine. These undirected arcs are represented by dashed lines in Figure 2.5. Note that the set of disjunctive arcs forms m complete subgraphs (*cliques*), one for each machine.

The shop scheduling problem consists in defining a processing order for the operations that cannot be processed at the same time. A selection of one direction for each disjunctive arc that does not produce a cycle in the graph defines a processing order for those operations. Such a selection consequently represents a solution for the shop scheduling problem. In Figure 2.6, each disjunctive arc of the 3×3 JSSP instance has been replaced by a directed grey arc. The topological sort of the directed graph that results of this selection defines the processing order of the optimal solution from Figure 2.4. A schedule obtained from a consistent complete selection is semi-active.

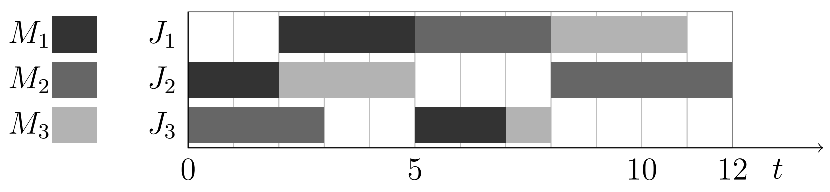


Figure 2.3: Gantt chart of an optimal schedule for the 3×3 JSSP instance arranged by jobs.

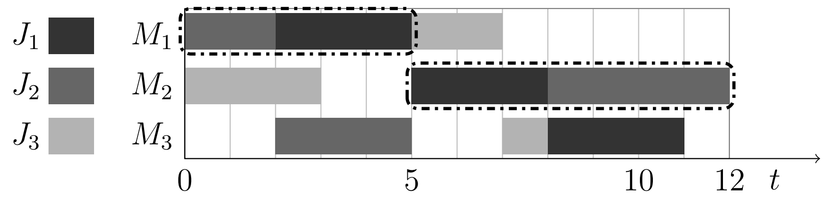


Figure 2.4: Gantt chart of an optimal schedule for the 3×3 JSSP instance arranged by machines. Dashed lines surround each block of operations of the critical path.

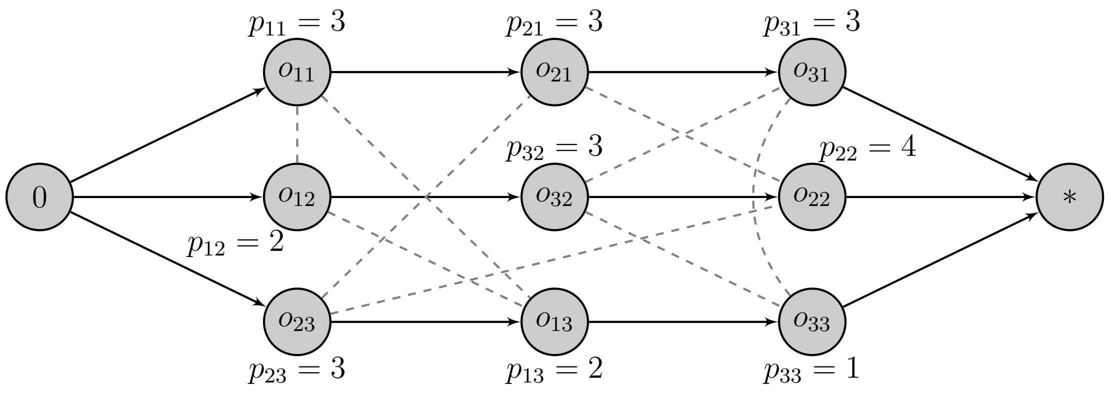


Figure 2.5: A disjunctive graph representation for the 3×3 JSSP instance.

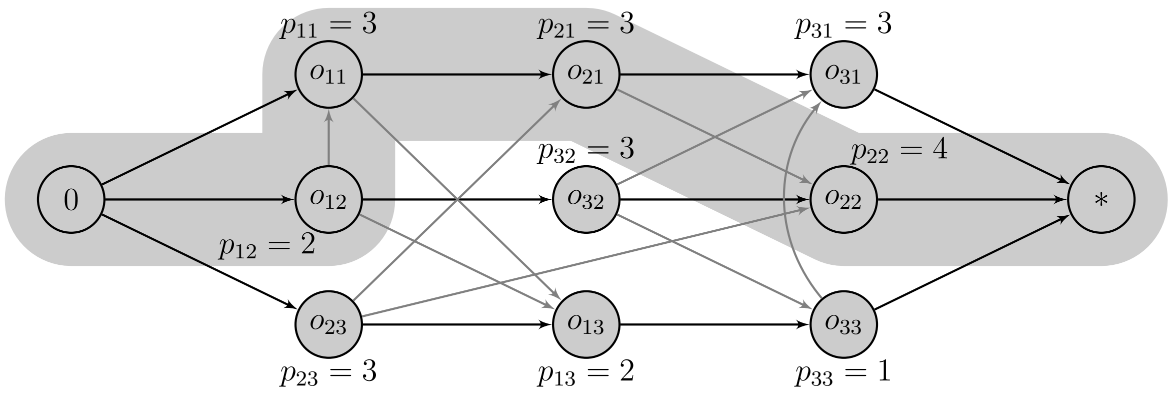


Figure 2.6: A graph representation of the optimal solution for the 3×3 JSSP instance from Figure 2.4. The critical path is marked in grey.

2.3.4 Makespan, critical paths and blocks of operations

The makespan is defined as the maximum completion time, or as the completion time of the latest job. In the graph representation, the makespan is determined by a path from the source 0 to the sink $*$ with the greatest sum of vertex weights. Such a path is called *critical path* and it is the longest sequence of consecutive operations in the schedule, as highlighted in Figure 2.6. The operations in a critical path are called *critical operations*. A sequence of consecutive critical operations on the same machine is called a *critical block*. The Gantt chart in Figure 2.4 shows the critical operations in two critical blocks surrounded by dashed lines.

2.4 Data structures for representing schedules

2.4.1 Binary representation

In the disjunctive graph representation, the selection of one direction of a disjunctive arcs defines the precedence order of two operations. A disjunctive arc that connects operations o_{ij} and $o_{ij'}$, corresponds to the variables $y_{ijj'}$ of the integer linear program formulation. Thus, a representation may be defined as a binary string of length $mn(n-1)/2$ that lists the values of the corresponding disjunctive arcs in a fixed order. For example, the string $S = 011100001$ is a binary representation of the optimal schedule from Figure 2.6, where the list of variables $y_{ijj'}$ that correspond to the disjunctive arcs from operation o_{ij} to $o_{ij'}$ such that $j < j'$ and ordered by i, j, and j'.

This binary representation shares the inconsistency problem with the disjunctive graph representation. A binary string may represent a cyclic selection in the disjunctive arcs of the graph, and consequently represents no valid schedule. For example, the string 010101010 represents no valid schedule, because it produces at least one cycle in each machine of the disjunctive graph from Figure 2.5. Modifications to a binary string may represent neighbourhoods to be explored, but also may generate inconsistencies that must be prevented or corrected.

2.4.2 Permutation representation with m-partitions

A schedule can also be represented by a sequence of m separated permutations of jobs $S = (\pi_1(1), \ldots, \pi_1(n), \pi_2(1), \ldots, \pi_2(n), \ldots, \pi_m(1) \ldots, \pi_m(n))$, where each permutation π_i gives the processing order for machine M_i. For example, the sequence $S = (\overbrace{2,1,3}^{\pi_1}, \overbrace{3,1,2}^{\pi_2}, \overbrace{2,3,1}^{\pi_3})$ is a permutation representation with m-partitions of the

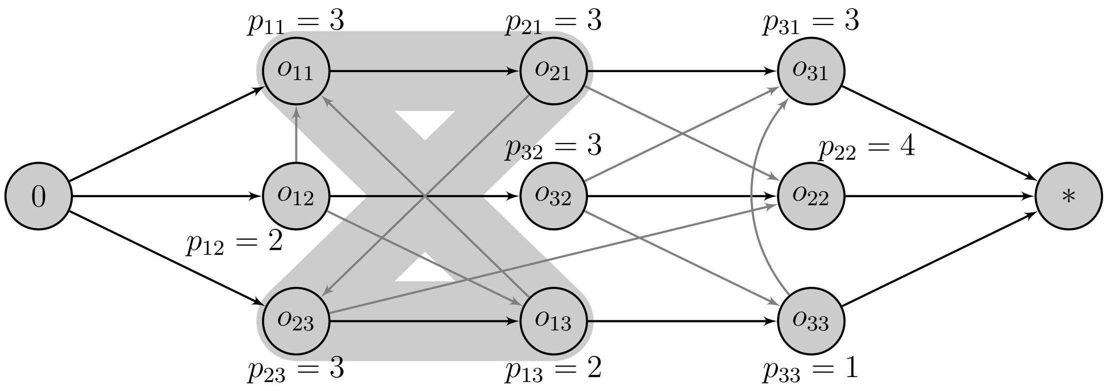


Figure 2.7: A cycle in an invalid graph representation of the 3×3 JSSP instance.

optimal schedule from Figure 2.6. Even though each permutation defines the processing order of a machine without producing a cycle in it, they may produce a cycle among the machines. For example, the sequence $(2, 3, 1, 1, 3, 2, 2, 3, 1)$ represents no valid schedule, because it produces a cycle as shown in Figure 2.7. Thus, inconsistencies introduced by a modification in a permutation representation with m-partitions must be prevented or corrected.

2.4.3 Permutation representation with m-repetitions

Another representation of a schedule is the permutation of a multiset (set with repeated elements) that results from the union of the m sets of jobs. A multiset permutation $\rho = (\rho(1), \rho(2), \ldots, \rho(n \times m))$ contains nm job numbers where each job number is repeated m times. For example, the permutation with m-repetitions $\rho = (2, 3, 2, 1, 3, 3, 1, 2, 1)$ represents the optimal schedule from Figure 2.6. Figure 2.8 shows the decodification method for this representation. Starting from the left, the k-th occurrence of a job number refers to its k-th operation, and the processing order determines the machine that processes this operation. Then, the schedule is created by following the available operations from left to right. In contrast to the permutation representation with m-partitions, all the job numbers of all machines are mixed in the

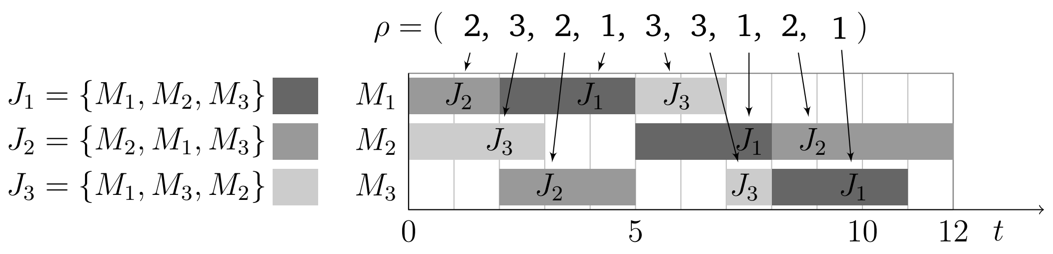
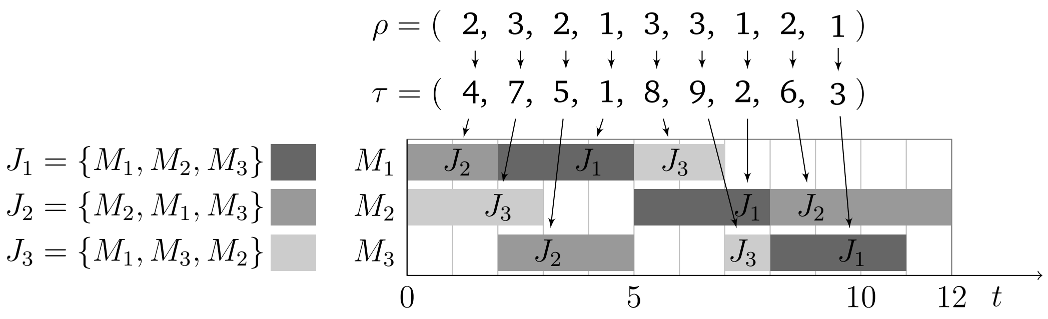

$\rho = ($ 2, 3, 2, 1, 3, 3, 1, 2, 1 $)$
$J_1 = \{M_1, M_2, M_3\}$
$J_2 = \{M_2, M_1, M_3\}$
$J_3 = \{M_1, M_3, M_2\}$
M_1
M_2
M_3
J_2
J_1
J_3
J_3
J_1
J_2
J_2
J_3
J_1
0
5
10
12
t

Figure 2.8: Decodification of a job permutation with m-repetitions of an optimal solution for the 3×3 JSSP instance. The processing order for each job J_j is on the left.



$\rho = (2, 3, 2, 1, 3, 3, 1, 2, 1)$
$\tau = (4, 7, 5, 1, 8, 9, 2, 6, 3)$
$J_1 = \{M_1, M_2, M_3\}$
$J_2 = \{M_2, M_1, M_3\}$
$J_3 = \{M_1, M_3, M_2\}$
M_1
M_2
M_3
J_2
J_1
J_3
J_3
J_1
J_2
J_2
J_3
J_1
0
5
10
12
t

Figure 2.9: An operation permutation of an optimal solution for the 3×3 JSSP instance as an intermediate decodification.

multiset permutation, eliminating the inconsistencies in the processing order for the operations of a job. Thus, any permutation of this representation produces a valid semi-active schedule without inconsistencies.

2.4.4 Operation permutation representation

If we assign a number from 1 to nm to each operation, such that $m(j-1)+1, m(j-1)+2, \ldots, mj$ are the operations of job J_j, a permutation of all the operations represents a schedule. An operation permutation may be seen as an intermediate representation between a job permutation representation and its decoded schedule. A job permutation with m-repetitions $\rho = (2,3,2,1,3,3,1,2,1)$ can be rewritten as an operation permutation $\tau = (4,7,5,1,8,9,2,6,3)$ by replacing the k-th occurrence of a job number with its respective operation number, as shown in Figure 2.9. The processing order is completely specified in an operation permutation, then there is no need for decodification. On the other hand, violations of the job processing order must be eliminated. For example, the permutation $(4,7,5,1,9,8,2,6,3)$ is invalid due to the order of operations 8 and 9.

2.5 Multi-objective optimization

In a multi-objective optimization problem, a solution x must be found in the set X of feasible solutions such that a set of functions $f_1(x), f_2(x), \ldots, f_k(x)$ is optimized. The objective functions are said to be *conflicting* if there is no single solution that simultaneously optimizes all the objectives. Figure 2.10 shows two solutions with two conflicting objective functions. Each solution is better than the other in different objective functions, i.e., neither solution is absolutely better than the other. The next definitions are necessary to deal with conflicting objectives.

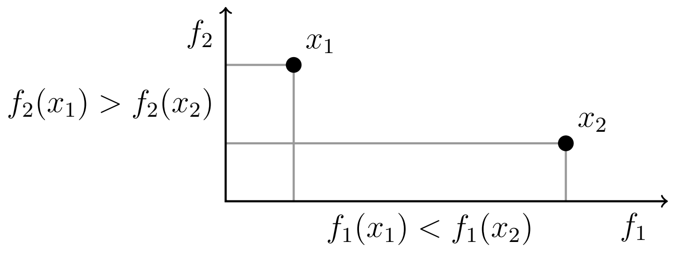


Figure 2.10: Optimal non-permutation and best possible permutation schedules for minimizing makespan.

Pareto dominance. A solution $x \in X$ is said to *Pareto dominate* another solution $x' \in X$ if $f_i(x) \leq f_i(x')$ $\forall i \in [k]$ and if $\exists i \in [k]$ such that $f_i(x) < f_i(x')$.

Non-dominated solution. A solution $x \in X$ is called *Pareto optimal* or *non-dominated* if there is no $x' \in X$ that dominates x.

Non-dominated set. A set of solutions is said to be a *non-dominated set* if no member of the set is dominated by any other member.

Pareto frontier. The *Pareto frontier* or *Pareto optimal set* contains all the non-dominated solutions in X.

Objectives scalarization. The *scalarization of objectives* is the formulation of a single-objective function based on the multiple objectives, such that an optimal solution of the scalarized single-objective optimization problem is part of the Pareto frontier of the multi-objective optimization problem. Note that only one optimal solution in the Pareto frontier is obtained with each scalarization.

Approaches for solving multi-objective optimization problems are divided into three classes:

A priori approach. The preferences of the decision maker are given *a priori* by a weighted combination of the objective functions. Thus, the multi-objective optimization problem is described as a scalarized single-objective optimization problem

$$\textbf{min.} \quad w_1 f_1(x) + w_2 f_2(x) + \cdots + w_k f_k(x), \tag{2.31}$$

$$\textbf{s.t.} \quad x \in X. \tag{2.32}$$

A posteriori approach. The decision maker selects *a posteriori* his preferred solution from the non-dominated set of the best solutions that the solving process was able to find by optimizing all the objectives simultaneously. Thus, the multi-objective optimization problem is described as

$$\textbf{min.} \quad f_1(x), f_2(x), \ldots, f_k(x), \tag{2.33}$$

$$\textbf{s.t.} \quad x \in X. \tag{2.34}$$

Interactive approach. The decision maker introduces his preferences at each step of the solving process, and the process determines the best solutions according to the introduced preferences and asks to the decision maker for new preferences before continuing the search.

2.6 Concluding remarks

The permutation and non-permutation variations of the flow shop scheduling are very similar, their only difference is the permutation restriction. This chapter described the shop scheduling problems and their characteristics, focusing on the flow shop scheduling problem. The non-permutation FSSP is special case of the JSSP, they share similar characteristics. This chapter also presented theoretical concepts of shop scheduling that may be applied to the non-permutation flow shop scheduling problem.

3 LITERATURE REVIEW

Six decades have passed since the initial paper of Johnson (1954) on the two machines flow shop scheduling problem was published. During the first two decades after that, many scheduling problems were modeled (such as the flow shop scheduling problem or FSSP and the job shop scheduling problem or JSSP), and the main research efforts were in polynomial-time algorithms for small problems (such as two machines or two jobs), enumeration algorithms (such as branch-and-bound), and lower bounds (based on relaxations of the problems).

After the development of computational complexity theory, the publications of Garey, Johnson & Sethi (1976) and Garey & Johnson (1979) proved that the flow shop, job shop, and other scheduling problems are NP-complete, and the publication of Graham et al. (1979) surveyed the scheduling literature, classified the existing problems with the corresponding complexity, and proposed a three-field notation $\alpha|\beta|\gamma$ to describe their characteristics (as explained in Section 2.1). The NP-completeness of the problems led the following scheduling research to focus on approximation and heuristic algorithms that produce near optimal solutions in a reasonable computation time, limiting the research on exact methods to small instances, and to cases that can (or need to) afford a considerable computation time. Thus, the computational complexity and the classification helped the scheduling to become a well structured research field with many growing research areas as we know today.

In this chapter, we review the most important contributions in the literature that are relevant to our research. We review the literature on the permutation flow shop scheduling problem (or PFSSP) in Section 3.1, and on the non-permutation (or general) FSSP in Section 3.2. As the non-permutation FSSP can be considered a specific case of the JSSP, Section 3.3 presents some relevant literature on the JSSP. Section 3.4 presents benchmarks of the commonly used instances for the flow shop and the job shop scheduling problems. Finally, Section 3.5 presents examples in the literature that successfully introduce human diversity in the planning and scheduling processes of productive systems.

The research field of shop scheduling is vast. *Web of Science* reports more than 7800 publications on flow shop and job shop scheduling, and *Scopus* reports more than 10000 publications. Thus, we limit our literature review to the most relevant and recent research in the area. If the reader wishes to have more detail on some area, we recommend the following bibliography. For more information on scheduling problems, we refer the reader to the reviews of Potts & Strusevich (2009) and Zobolas, Tarantilis & Ioannou (2008), and the books of Brucker (2004), Błażewicz et al. (2007), Pinedo (2012), and Framinan, Leisten & Ruiz García (2014). For more information on flow shop scheduling problems, we refer the reader to the reviews of Framinan, Gupta & Leisten (2004), Reza Hejazi & Saghafian (2005), Ruiz & Maroto (2005), Gupta & Stafford (2006), and Tyagi, Varshney & Chanramouli (2013), and the book of Emmons & Vairaktarakis (2012). The review of Pan & Ruiz (2013) presents more information on flow shop scheduling minimizing total completion time. For more information on multicriteria scheduling including the flow shop, we refer the reader to the reviews of Hoogeveen (2005), Minella, Ruiz & Ciavotta (2008) Sun et al. (2011), and Yenisey & Yagmahan (2014), and the book of T'kindt & Billaut (2006).

3.1 The permutation flow shop scheduling problem

Web of Science reports more than 3700 publications for a search of the words "flow shop" or "flowshop" in the topic, and the most important among them (with more than 800 citations) is the publication of Nawaz, Enscore & Ham (1983) that proposed the NEH constructive heuristic.

3.1.1 The constructive heuristic NEH for the PFSSP with makespan criterion

Algorithm 3.1 presents a pseudocode for the NEH constructive heuristic. First, the jobs are arranged to create a priority order π_o, according to a non-increasing value of their total processing time, which is defined as $\sum_{i\in[m]} p_{ij}$ for each job J_j. Then, a schedule is constructed by iteratively inserting the jobs into the current partial solution π at the position that minimizes the makespan until π becomes a complete solution. Assuming that π is the current sequence already determined for the first $j-1$ jobs, there are j insertion places for job $\pi_o(j)$ that produce j candidate sequences, and the sequence that yields the minimum makespan becomes the current sequence for the next iteration.

Algorithm 3.1 NEH constructive heuristic for PFSSP.

```
Input: The processing times $p_{ij}$ for each job $J_j$ on each machine $M_m$.
Output: A permutation schedule $\pi$.
1: function NEH_CONSTRUCTIVE_HEURISTIC( )
2:     $\pi_o := (\pi_o(1), \ldots, \pi_o(n))$ in non-increasing total processing time
3:     $\pi := (\pi_o(1))$
4:     for $\pi_o(j), j \in [2, n]$ do
5:         evaluate all the insertion positions of job $\pi_o(j)$ into $\pi$
6:         insert job $\pi_o(j)$ into $\pi$ at the position which minimizes $C_{\max}$
7:     end for
8:     return $\pi$
9: end function
```

The makespan calculation of a schedule has a complexity of $O(mn)$, as the completion time of all the n jobs in all the m machines must be calculated. Taillard (1990) proposed an acceleration that calculates the makespan of the n sequences that result of a job insertion with a complexity of $O(mn)$, by calculating the times of heads and tails for every insertion position once in $O(nm)$. This acceleration reduces the complexity of the NEH heuristic from $O(n^3m)$ to $O(n^2m)$.

Formally, the insertion of a job J_l into the partial permutation schedule $\pi = (\pi(1), \ldots, \pi(k))$ results into another partial (or complete) permutation $\pi' = (\pi'(1), \ldots, \pi'(k+1))$, and the acceleration of Taillard calculates the makespan values $M_1, \ldots, M_{k+1}$ for the insertion of job J_l into each position $j \in [k+1]$ as follows. The earliest completion time $e_{i,j}$ of each job $\pi(j)$ on each machine i is defined as

$$e_{i,j} = \max\{e_{i,j-1}, e_{i-1,j}\} + p_{i,\pi(j)}, \tag{3.1}$$

for $i \in [m]$ and $j \in [k]$, with $e_{0,j} = 0$ and $e_{i,0} = 0$. Similarly, the time $q_{i,j}$ between the end of processing and the latest starting time of each job j on each machine i is defined as

$$q_{i,j} = \max\{q_{i,j+1}, q_{i+1,j}\} + p_{i,\pi(j)}, \tag{3.2}$$

for $i \in [m]$ and $j \in [k]$, with $q_{m+1,j} = 0$ and $q_{i,k+1} = 0$. After the insertion of job J_l into π' at position $j \in [k+1]$, the earliest completion times of the jobs before the position j remain unchanged, and they are used to calculate the earliest relative completion time $f_{i,j}$ of job J_l (if it is inserted at position j) on machine i as

$$f_{i,j} = \max\{f_{i-1,j}, e_{i,j-1}\} + p_{i,l}, \tag{3.3}$$

for $i \in [m]$ and $j \in [k+1]$, with $f_{0,j} = 0$ and $e_{i,0} = 0$. Then, the makespan M_j of the permutation schedule π' that result of the insertion of job J_l at position $j \in [k+1]$ is defined as

$$M_j = \max_{i\in[m]}\{f_{i,j} + q_{i,j}\}. \tag{3.4}$$

Many researchers have proposed modifications to improve the results of NEH, for example: Nagano, Moccellin et al. (2002) Kalczynski & Kamburowski (2008), Dong, Huang & Chen (2008), Ribas, Companys & Tort-Martorell (2010), Fernandez-Viagas & Framinan (2014), and Vasiljevic & Danilovic (2015). The modifications add a very small computational cost by evaluating different initial priority orders, and by breaking ties in the initial priority order and in the selection of the insertion position. The current best constructive heuristic with complexity $O(n^2m)$ is the initial priority order of Dong, Huang & Chen (2008) followed by the tie-breaking mechanism of Fernandez-Viagas & Framinan (2014) with an average relative percent deviation of 2.897 from the best known values. The best constructive method with complexity $O(Ln^2m)$ with $L \leq 26$ and with an average relative percent deviation of 2.465 from the best known values was proposed by Vasiljevic & Danilovic (2015). Their method is not deterministic, they implement different versions of NEH with random tie breaks and keep the best result. The NEH algorithm has even been accelerated by implementing it into a GPU by Metlicka et al. (2014), presenting an average speedup factor of 1.62 and a largest speedup factor of 5.18 when compared to a CPU implementation.

3.1.2 The constructive heuristics FRB for the PFSSP with makespan criterion

Farahmand Rad, Ruiz & Boroojerdian (2009) proposed a set of five heuristics based on modifications of the NEH heuristic that evaluate the reinsertion of already inserted jobs. Algorithm 3.2 shows the FRB2 heuristic. It sorts all the operations in non-increasing order of their processing times. After that, it inserts the job of each operation just like the NEH heuristic. If the job was already inserted, it is removed and its reinsertion is evaluated. The FRB2 heuristic inserts or reinserts each one of the n jobs at most m times, and each time with a time complexity of $O(nm)$ using the acceleration of Taillard. Thus, it has a time complexity of $O(n^2m^2)$.

The FRB1 heuristic is similar to the FRB2 heuristic, but calculates the makespan using secondary values of the processing times that are initially zero. At each iteration, it copies the processing time of the current operation, and inserts (or reinserts) the corresponding job at the best position. The FRB1 heuristic has the same time complexity of the FRB2 heuristic, and obtained better results than NEH, but worse results than the other heuristics FBR.

Algorithm 3.2 FRB2 constructive heuristic for PFSSP.

Input: The processing times p_{ij} for each job J_j on each machine M_m.
Output: A permutation schedule π.

```
1: function FRB2_CONSTRUCTIVE_HEURISTIC( )
2:     $PV := \{p_{11}, p_{21}, \ldots, p_{m,1}, p_{1,2}, p_{2,2}, \ldots, p_{m,2}, \ldots, p_{1,n}, p_{2,n}, \ldots, p_{m,n}\}$
3:     sort $PV$ in non-increasing processing time
4:     $\pi := (PV[1])$
5:     $lastjob := job(PV[1])$
6:     for $step := 2$ to $mn$ do
7:         $j := job(PV[1])$
8:         if $lastjob \neq j$ then
9:             if $j \in \pi$ then
10:                remove $j$ from $\pi$
11:            end if
12:            evaluate all the insertion positions of job $\pi_o(j)$ into $\pi$
13:            insert job $\pi_o(j)$ into $\pi$ at the position which minimizes $C_{\max}$
14:        end if
15:    end for
16:    return $\pi$
17: end function
```

The last three heuristics explore the reinsertion of jobs after each new job is inserted. The FRB3 heuristic reinserts all the previously inserted jobs, while the FRB4 heuristic reinserts jobs that are at a distance of at most k positions from the last inserted job. We refer to each FRB4 heuristic with a different k as FRB4$_k$. Algorithm 3.3 shows the FRB4$_k$ heuristic. Heuristic FRB3 is equivalent to FRB4$_n$. First, it orders the jobs by their non-increasing total processing time. At each iteration, it inserts the next job $\pi_o(j)$ in the best position p and reinserts the jobs that are at most at k positions after or before position p. The inner reinsertion loop has a time complexity of $O(knm)$, and the outer loop has a time complexity of $O(kn^2m)$. With small values $k \ll n$, the time complexity of the FRB4$_k$ heuristic is the same as NEH $O(n^2m)$, but the computational effort grows with the value of k. The time complexity of FRB3 is $O(n^3m)$. The FRB4$_k$ heuristic was tested with $k \in \{2, 4, 6, 8, 10, 12\}$. All the FRB4$_k$ heuristics tested were faster than FRB2. FRB4$_{10}$ and FRB4$_{12}$ obtained better average results than FRB2, showing that FRB4$_{12}$ has the best cost/benefit. The FRB3 heuristic obtained the second best results among the proposed heuristics, but it is also the second most expensive.

The FRB5 heuristic explores the reinsertion of already inserted jobs after each new job is inserted with an insertion local search. It obtained the best results among the proposed heuristics, but it is also the most expensive. Farahmand Rad, Ruiz & Boroojerdian (2009) also tested the FRB2, FRB3, and FRB5 heuristics as seeds for the

Algorithm 3.3 FRB4$_k$ constructive heuristic for PFSSP.

```
Input: The processing times $p_{ij}$ for each job $J_j$ on each machine $M_m$,
       a limit $k$ of adjacent job to be reinserted.
Output: A permutation schedule $\pi$.
 1: function FRB4_CONSTRUCTIVE_HEURISTIC( $k$ )
 2:     $\pi_o := (\pi_o(1), \ldots, \pi_o(n))$ in non-increasing total processing time
 3:     $\pi := (\pi_o(1))$
 4:     for $\pi_o(j), j \in [2, n]$ do
 5:         evaluate all the insertion positions of job $\pi_o(j)$ into $\pi$
 6:         insert job $\pi_o(j)$ into $\pi$ at the position $p$ which minimizes $C_{\max}$
 7:         for $step2 := max(1, p-k)$ to $min(|\pi|, p+k)$ do
 8:             remove job $J_{j'} = \pi(step2)$ from $\pi$
 9:             evaluate all the reinsertion positions of job $J_{j'}$ into $\pi$
10:             reinsert job $J_{j'}$ into $\pi$ at the position $p$ which minimizes $C_{\max}$
11:         end for
12:     end for
13:     return $\pi$
14: end function
```

simulated annealing of Osman & Potts (1989), the ant colony algorithm of Rajendran & Ziegler (2004), the genetic algorithm of Ruiz, Maroto & Alcaraz (2006), and the iterated greedy algorithm of Ruiz & Stützle (2007). The use of the FRB5 heuristic as seed for the tested metaheuristics produced the best results.

3.1.3 Metaheuristics applied to the PFSSP with makespan criterion

The NEH constructive heuristic is fast and efficient, but it still leaves room for improvement with other methods. NEH gives the initial sequence for many metaheuristics in the literature, such as tabu search (GRABOWSKI; WODECKI, 2004; EKŞIOĞLU; EKŞIOĞLU; JAIN, 2008), scatter search (NOWICKI; SMUTNICKI, 2006; HAQ et al., 2007), genetic algorithms (NAGANO; RUIZ; LORENA, 2008; CHEN et al., 2012; CHANG et al., 2013b), particle swarm optimization (TASGETIREN et al., 2007), ant colony optimization (RAJENDRAN; ZIEGLER, 2004; AHMADIZAR, 2012), artificial bee colonies (LIU; LIU, 2013; PAN et al., 2014), differential evolution (ONWUBOLU; DAVENDRA, 2006; PAN; TASGETIREN; LIANG, 2008), evolutionary algorithms (CHANG et al., 2013a; HSU; CHANG; CHEN, 2015), iterated greedy algorithms (RUIZ; STÜTZLE, 2007; FERNANDEZ-VIAGAS; FRAMINAN, 2014; VALLADA; RUIZ, 2009), artificial immune systems (CHEN; CHANG; LIN, 2014), teaching learning optimization (XIE et al., 2014), cuckoo search (MARICHELVAM, 2012; DASGUPTA; DAS, 2015), hybrids of these methods (LIN; YING, 2009; TSENG; LIN, 2009; ZOBOLAS; TARANTILIS; IOANNOU, 2009). This list is just a small sample of the many algorithms in the literature. The iterated greedy algorithm of Fernandez-

Viagas & Framinan (2014) is currently the best performing method. They proposed the use of their tie-breaking scheme in an iterated greedy algorithm based on that proposed by Ruiz & Stützle (2007). Vallada & Ruiz (2009) proposed a parallel cooperative iterated greedy algorithm that run the iterated greedy algorithm proposed by Ruiz & Stützle (2007) in up to 12 processors. This method allows the parallel algorithm to share the best results among them and continue exploring in different areas of the search space. This parallel method produces better results.

3.1.4 Constructive heuristics for PFSSP with total completion time criterion

Pan & Ruiz (2013) compared 14 simple and 13 composite heuristics proposed in the literature for the permutation flow shop scheduling problem with total completion time criterion. They studied the quality of the solutions (average relative percentage deviation over the best known solution) and the CPU time (in seconds) of each heuristic as a bi-criteria optimization problem to create a set of non dominated heuristics. Many of the compared heuristics are based on the LR(k) heuristic of Liu & Reeves (2001), that has a computational complexity of $O(kn^3m)$. Fernandez-Viagas & Framinan (2015) proposed a new heuristic FF(k) with a computational complexity of $O(kn^2m)$ that can replace LR(k) in all the derived heuristics with better results in shorter time. This new set of heuristics based on FF for the PFSSP with total completion time criterion is the current state of the art in the literature.

The LR heuristic of Liu & Reeves (2001) iteratively appends an unscheduled jobs at the end of a partial permutation schedule. To do so, they use an index function ξ_{jk} to estimate the suitability for a job J_j to be appended after the already k scheduled jobs. It is defined as

$$\xi_{jk} = (n-k-2)IT_{jk} + AT_{jk}, \tag{3.5}$$

for a job J_j to be appended on position where IT_{jk} estimates the idle time induced by the scheduling of job J_j at position $k+1$, and is defined as

$$IT_{jk} = m\sum_{i=2}^{m} \frac{\max\{C_{i-1,\pi(k+1)} - C_{i,\pi(k)}, 0\}}{i + k(m-i)/(n-2)}, \tag{3.6}$$

and $AT_{jk} = C_{m,j} + C_{m,a}$ is an artificial flowtime defined as the sum of the completion time $C_{m,j}$ of job J_j to be appended at position $k+1$ and the completion time $C_{m,a}$ of an artificial job J_a that would be appended at position $k+2$, which has processing times that are the average of the processing times of the unscheduled jobs excluding job J_j to be appended at position $k+1$. Fernandez-Viagas & Framinan (2015) observed that the creation of an artificial job and the calculation of its completion time has a complexity of $O(mn)$, and that it does not influence the objective function as directly as the other two elements. Thus, they proposed a new index function ξ'_{jk} defined as follows:

$$\xi'_{jk} = \frac{n-k-2}{a} IT'_{jk} + AT'_{jk}, \tag{3.7}$$

$$IT'_{jk} = m \sum_{i=2}^{m} \frac{\max\{C_{i-1,\pi(k+1)} - C_{i,\pi(k)}, 0\}}{i-b+k(m-i+b)/(n-2)}, \tag{3.8}$$

$$AT'_{jk} = C_{m,j}, \tag{3.9}$$

also introducing two parameters a and b to balance the influence of idle times and completion time of the newly inserted job.

3.1.5 Metaheuristics applied to the PFSSP with total completion time criterion

Pan & Ruiz (2012) proposed four iterated local search based methods. The proposed iterated local search (ILS) has a perturbation scheme based on random shift of some jobs, and a local search based on the shift neighbourhood. The proposed iterated greedy algorithm (IGA) has a perturbation scheme based on the removal and reinsertion of a random subset of jobs, and it also applies the local search based on the shift neighbourhood. The other two methods are a population based iterated local search (pILS) and a population based iterated greedy algorithm (pIGA). The four methods use the acceptance criterion proposed by Metropolis et al. (1953). The proposed methods were compared to twelve state-of-the-art metaheuristics that include the discrete differential evolution of Pan, Tasgetiren & Liang (2008), the hybrid genetic algorithm of Zhang, Li & Wang (2009), the iterated local shearch of Dong, Huang & Chen (2009), the asynchronous genetic local search of Xu, Xu & Gu (2011), the discrete artificial bee colony of Tasgetiren et al. (2011), the stochastic local search of Dubois-Lacoste, López-Ibáñez & Stützle (2011). Their computational results show that the iterated greedy algorithm outperformed the other methods.

Dong, Huang & Chen (2009) proposed an iterated local search that has a perturbation scheme based on the swap of pairs of random adjacent jobs, and a local search based on the shift neighbourhood. Later, Dong et al. (2013) proposed a multi-restart scheme to enhance their iterated local search (or MRSILS). At each iteration, the MRSILS perturbs a solution randomly selected from a pool that maintains the best local minima found during the search. Their computational results show that MRSILS performs better when compared to the iterated local search of Dong, Huang & Chen (2009), the discrete differential evolution of Pan, Tasgetiren & Liang (2008), the hybrid genetic algorithm of Zhang, Li & Wang (2009), and the discrete artificial bee colony of Tasgetiren et al. (2011).

Recently, Dong et al. (2015) proposed a self-adaptive perturbation and multiple neighbourhood local search to enhance the iterated local search. The self-adaptive strategy evaluates the neighbourhoods around the local optimum and adjusts the perturbation strength according to this evaluation. The multiple neighbourhood local search explores the swap and the shift neighbourhoods. They evaluated the contribution to the performance of the multi-neighbourhood search and the self-adaptive perturbation independently, concluding that the best performing results are obtained with the use of the multi-neighbourhood search, although the self-adaptive perturbation gives a small improvement when used with the multi-neighbourhood search. The proposed iterated local search is compared to those of Dong et al. (2013), and Pan & Ruiz (2012). The method of Dong et al. (2015) clearly outperforms the method of Dong et al. (2013). The methods of Dong et al. (2015) and Pan & Ruiz (2012) present no significant difference for times of ρmn milliseconds ($\rho \in \{30, 60, 90\}$), but Dong et al. (2015) is superior for a larger time scale of ρmn^3 milliseconds ($\rho \in \{0.004, 0.012, 0.02\}$).

Other notorious methods are the differential evolution algorithm for permutations of Santucci, Baioletti & Milani (2014), the particle swarm optimization of Zhang & Wu (2014), and the hybrid particle swarm optimization of Akhshabi, Tavakkoli-Moghaddam & Rahnamay-Roodposhti (2014). Czapiński (2010) proposed a parallel simulated annealing with a genetic enhancement that found the best known values for almost all the instances of the Taillard's benchmark, and they are still the best known values for the first 90 instances. Since then, the methods of Pan & Ruiz (2012), Akhshabi, Tavakkoli-Moghaddam & Rahnamay-Roodposhti (2014), Santucci, Baioletti & Milani (2014), and Dong et al. (2015) have found new best known values for the 30 largest instances. Table 3.1 presents the current best known values for the 30 largest instances of the Taillard's benchmark and the methods that found them. A direct comparison to the methods of Dong et al. (2015) and Pan & Ruiz (2012) is difficult due to the different running times of Santucci, Baioletti & Milani (2014) and Akhshabi, Tavakkoli-Moghaddam & Rahnamay-Roodposhti (2014) and the lack of reference for the relative deviations of Zhang & Wu (2014). Thus, a future practical comparison of those methods is necessary.

3.1.6 Metaheuristics applied to the bi-objective PFSSP with makespan and total completion time criteria

We have reviewed the literature for the PFSSP with a single objective function: minimizing makespan or minimizing total completion time. Now, we review the literature that consider both criteria in a Pareto frontier approach. The review of

Table 3.1: Best known total completion times for the largest instances of Taillard.

Inst.	Best C_{sum}	Method	Inst.	Best C_{sum}	Method	Inst.	Best C_{sum}	Method
ta091	1046314	SA	ta101	1224734	ILS	ta111	6677283	ILS
ta092	1034195	SA	ta102	1239246	DEA	ta112	6723548	PSO
ta093	1045706	DEA	ta103	1254162	PSO	ta113	6697443	ILS
ta094	1029580	DEA	ta104	1233443	DEA	ta114	6741759	ILS
ta095	1034027	SA	ta105	1220117	DEA	ta115	6715997	ILS
ta096	1006195	SA	ta106	1223238	DEA	ta116	6723435	ILS
ta097	1052786	DEA	ta107	1237116	DEA	ta117	6674684	ILS
ta098	1044875	SA	ta108	1235460	PSO	ta118	6746086	ILS
ta099	1023315	DEA	ta109	1225186	DEA	ta119	6674879	ILS
ta100	1029198	DEA	ta110	1244200	DEA	ta120	6733095	ILS

SA:	Simulated annealing of Czapiński (2010).
DEA:	Differential evolution algorithm of Santucci, Baioletti & Milani (2014).
PSO:	Particle swarm optimization of Akhshabi, Tavakkoli-Moghaddam & Rahnamay-Roodposhti (2014).
ILS:	Iterated local search of Dong et al. (2015).

Minella, Ruiz & Ciavotta (2008) identified the multi-objective simulated annealing (MOSA-II) of Varadharajan & Rajendran (2005) as the most effective method by that time. After that, many authors have reimplemented it to compare their methods, such as the genetic algorithm with heterogeneous population (hMGA) of Yandra & Tamura (2007), the iterated greedy search (MOIGS) of Framinan & Leisten (2008), the simulated annealing (M-MOSA) of Mokotoff (2009), the ant colony algorithm (MOACA) of Rajendran & Ziegler (2009), the memetic algorithm (impr-MA) of Chiang & Fu (2010), the restarted iterated Pareto greedy (RIPG) of Minella, Ruiz & Ciavotta (2011), the two phase and Pareto local search (TP+PLS) of Dubois-Lacoste, López-Ibáñez & Stützle (2011), the bi-objective multi-start simulated annealing (BMSA) of Lin & Ying (2013). All these publications give to their reimplementation of MOSA-II a second place after their own methods. The best results are obtained by the methods of Chiang & Fu (2010), Minella, Ruiz & Ciavotta (2011), Dubois-Lacoste, López-Ibáñez & Stützle (2011), and Lin & Ying (2013), but they cannot be directly compared because of the different quality measures and running times that were used.

A recent study by Bezerra, López-Ibáñez & Stützle (2014) reimplemented many generic multi-objective evolutionary algorithms and evaluated their effectiveness on solving the PFSSP under the same conditions, concluding that the genetic algorithm (NSGA-II) of Deb et al. (2002) is the best performing among the evaluated, but their study excludes state-of-the-art methods such as the specific genetic algorithm (hMGA) of Yandra & Tamura (2007), although hMGA has already claimed better results than NSGA-II.

3.2 The non-permutation flow shop scheduling problem

The publication of Heller (1960) was the first that used the term “flow shop”. He studied the distribution of the makespan over randomly generated schedules for two instances. He evaluated 3000 permutation schedules for the 100×10 instance, 12000 permutation and 9037 non-permutation schedules for the 20×10 instance. The average makespan of the random non-permutation schedules were four times larger than that of the random permutation schedules. Thus, they concluded that it is better to search for schedules which have the same ordering on all machines. Since then, almost all the researches on the FSSP focus their efforts on finding permutation schedules of good quality. For example, *Web of Science* reports more than 3700 publications for a search of the words “flow shop” or “flowshop” in the topic, and only 37 of them have the words “non-permutation” or “nonpermutation” in the topic. Even though there is no loss of generality for the case of two machines, or for three machines if minimizing makespan, this simplification excludes better (optimal) non-permutation schedules for the FSSP with more machines.

Krone & Steiglitz (1974) proposed a two phase heuristic for the FSSP with average completion time criterion. The first phase builds a random permutation schedule, that is improved by a shift local search. The second phase improves the permutation schedule by exploring job passing neighbours, thus creating a slightly better non-permutation schedule. They concluded that the second phase gives a small but important improvement in 80% of the tested permutation schedules, just by considering non-permutation schedules.

Tandon, Cummings & LeVanu (1991) compared the quantity and quality of permutation and non-permutation schedules. The schedules were obtained by enumerative search for small problems (up to 6 jobs or 6 machines), and by simulated annealing for medium problems (up to 8 jobs or 11 machines). They observed that the number of non-permutation schedules better than permutation schedules increases with the number of either jobs or machines, and the quality improvement from permutation to non-permutation schedules depends on the range of processing times and on the size of the problem, concluding that the consideration of non-permutation schedules is necessary to find optimal or near optimal schedules.

Potts, Shmoys & Williamson (1991) were the first to estimate the gap between the makespan of permutation and non-permutation schedules and they presented a family of instances for which the makespan of the best permutation schedule is worse than that of the optimal non-permutation schedule by a factor $\Omega(\sqrt{m})$, leaving the upper bound of this function as an open question. This question was recently answered by

Nagarajan & Sviridenko (2009) with a $\Theta(\min\{\sqrt{m}, \sqrt{n}\})$ approximation algorithm for the PFSSP with makespan and with average completion time criteria.

Raman (1995) considered heuristic methods to minimize tardiness in the FSSP, and the results show that, although producing non-permutation schedules require more computational effort, non-permutation schedules are generally superior and thus worthy.

Koulamas (1998) proposed a two phase heuristic called HFC for the FSSP with makespan criterion. The first phase creates a permutation schedule with a job priority order based on the solution of the two machine FSSP with the rule of Johnson (1954) for each of the $m(m-1)/2$ pairs of machines. The second phase considers job passing for all the adjacent pairs of jobs in the permutation schedule produced within the first phase, using the values calculated with the rule of Johnson (1954), and only if it improves the current solution. The HFC is compared to NEH heuristic with Taillard acceleration ($O(n^2m)$). Although the HFC has a time complexity of $O(n^2m^2)$, it is faster than NEH for instances with 20 machines or less. It finds schedules with a shorter makespan in 90% of the instances, and half of them are non-permutation schedules. Later studies do not confirm these results on larger sets of instances (RUIZ; MAROTO, 2005), but the conclusion concerning the improvement achieved with the consideration of non-permutation schedules is still valid.

Jain & Meeran (2002) proposed a multilevel hybrid system called core and shell framework (CSF) for the non-permutation FSSP. The CSF was originally designed for the JSSP by Jain & Meeran (1998). Based on concepts of scatter search, CSF consists of a core that intensifies the search over the reference solutions, and layered shells that combine the solutions in different manners to produce new diverse solutions for the core. The system core has local search with a high degree of intensification, mainly based on the reduced neighbourhood (explained in Section 3.3) of Nowicki & Smutnicki (1996), and a dynamic tabu tenure. The system shells are neighbourhood search methods with different degrees of diversification, based on concepts of path relinking, recombination of schedules, and bottleneck scheduling of the machine with most critical operations. The CSF showed better results when compared to other methods for JSSP, like the tabu search of Nowicki & Smutnicki (1996), using FSSP instances.

Liao, Liao & Tseng (2006) proposed a tabu search that explores a shift neighbourhood with a best improvement strategy. The shift neighbourhood is reduced to reinsert a job in a place with a distance of at most $n/2$ from its original position. Liao, Liao & Tseng (2006) reimplemented the genetic algorithm of Reeves (1995) for comparison. After a permutation schedule is generated, a non-permutation schedule is generated iteratively by fixing the scheduling sequence in the previous machines and using a new sequence for the current and following machines. The new sequences are also generated by the associated algorithm (either tabu search or genetic algorithm). Liao, Liao & Tseng (2006) evaluated the deviations between the best permutation and non-permutation schedules found by the tabu search and the genetic algorithm for six different optimization criteria that include makespan, total completion time and total tardiness. Their computational results show that tabu search finds better results than the genetic algorithm. Non-permutation schedules present little improvements for the completion-time based criteria, but the improvement is significant with respect to due-date based criteria. The time invested to produce non-permutation schedules grows greatly with the instance size. Liao, Liao & Tseng (2006) concluded that the time investment may be worthy depending on the obtained improvement, for example with the total tardiness criterion. Later, Liao & Huang (2010) studied the non-permutation FSSP with the total tardiness criterion, proposing three mix integer linear programming models and two tabu search algorithms with custom neighbourhoods for non-permutation schedules. The best model solved optimally instances with five machines and up to eight jobs and within 24 hours. The optimal solutions presented similar job sequences for adjacent machines, and the position of the jobs varied slightly. The proposed tabu search algorithms explore the swap of adjacent operations of the same jobs in different machines, producing good quality non-permutation schedules that are better than the permutation schedules, for instances of $n \in \{20, 50, 100\}$ jobs and $m \in \{5, 10, 20\}$ machines.

Haq et al. (2007) proposed a scatter search for the FSSP with an operation shift neighbourhood as intensification method and a voting system as a combination method. Their scatter search showed better results when compared to the CSF of Jain & Meeran (2002) and the tabu search (for JSSP) of Nowicki & Smutnicki (1996), using FSSP instances.

Ying & Lin (2007) proposed a multi-heuristic desirability ant colony system (MHD-ACS) for the non-permutation FSSP and compared it to simple constructive heuristics. A non-permutation schedule is constructed by finding a shortest path that visits all the nodes in the disjunctive graph representation. A desirability function influences the decision of an ant to choose a path, and the multi-heuristic desirability is based on a set of schedules generated with dispatching rules. Later, Ying (2008) proposed an iterated greedy that was superior than the MHD-ACS. Lin & Ying (2009) proposed a hybrid simulated annealing and tabu search that was superior than the MHD-ACS and to the separated simulated annealing and tabu search. Both, the iterated greedy and the hybrid simulated annealing and tabu search, use a representation of three permutations for each schedule: one permutation for the first two machines, another for the last two machines, and a third permutation for the middle machines (machines M_3 to M_{m-2}, if $m > 4$). This representation reduces the number of candidate solutions from $m!^n$ to $m!^3$, achieving results faster, but also ignoring possibly better candidate solutions.

Sadjadi, Bouquard & Ziaee (2008) also presented an ant colony algorithm algorithm for the permutation FSSP, and applied a non-permutation local search to the obtained permutation solution. The non-permutation local search explores exchanges of two jobs on the first k or the last k machines. The exchanged two jobs must be adjacent themselves or at most adjacent to a common third job. Permutation and non-permutation results are superior compared to other permutation methods. Non-permutation results improve permutation results with a small computational cost.

Zheng & Yamashiro (2010) proposed a quantum differential evolutionary algorithm (QDEA) for the non-permutation FSSP. They also use three permutations to represent each schedule. QDEA shows better results compared to the CSF of Jain & Meeran (2002), the MHD-ACS of Ying & Lin (2007), and the scatter search of Haq et al. (2007).

Zhang et al. (2013) proposed a reinforcement learning algorithm for the non-permutation FSSP with makespan criterion. To do this, they formulated the problem as a Semi-Markov Decision Process, defining states with features, actions based on heuristic and dispatching rules, and a reward system based on the average machine idle time. The reinforcement learning algorithm was compared to the MHD-ACS of Ying & Lin (2007) and the iterated greedy of Ying (2008), showing small improvements in quality.

Rossi & Lanzetta (2014) proposed a native non-permutation ant colony system (NNP-ACS) for the non-permutation FSSP and compared it to the MHD-ACS of Ying & Lin (2007). A desirability function influences the decision of an ant to choose a path. Besides a different desirability function (earliest starting time), NNP-ACS uses an adaptive parameter q_0 equivalent to the number of iterations without improvement divided by the limit of maximum number of iterations without improvement, while MHD-ACS set this parameter to $q_0 = 0.9$. NNP-ACS also improves each produced schedule with the local search (for JSSP) of Nowicki & Smutnicki (1996). The results showed that NNP-ACS is superior to MHD-ACS. In another publication, Rossi & Lanzetta (2013a) compared the NNP-ACS to other ACOs like Rajendran & Ziegler (2004) (for permutation FSSP) and Sadjadi, Bouquard & Ziaee (2008) (for non-permutation FSSP), showing better results. These comparisons were unfair with the other methods, because NNP-ACS has a stop criterion based on stability that allows significantly larger running times.

Gharbi, Labidi & Louly (2014) proposed lower and upper bounds for the non-permutation FSSP. They described two adjustment procedures from the literature based on disjunction and preemption principles. They also proposed three new adjustment procedures based on the principles of disjunction, semi-preemption, and the same permutation on the first (or last) two-machines of the optimal schedule. These adjustments are used to create a new lower bounding procedure and five heuristics that produce upper bounds. Their procedure found better lower bounds compared to the one machine based lower bound for the majority of the instances, but with a higher computational cost. They created a set of instances with $n \in \{10, 20, 30, 40, 50, 60, 70, 80\}$ jobs and $m \in \{4, 5, 6\}$ machines for testing. Their heuristics presented a reduced gap to the lower bounds, finding the optimal values for many of the instances, but with a high computational cost. Their methods solved many instances of the non-permutation FSSP, but they do not present any statistic about differences between permutation and non-permutation in the produced schedules.

Finally, the consideration of non-permutation schedules has lead to better solutions in other special cases of the FSSP, for example with missing operations (RAJENDRAN; ZIEGLER, 2001; PUGAZHENDHI et al., 2003; PUGAZHENDHI et al., 2004a; PUGAZHENDHI et al., 2004b; TSENG; LIAO; LIAO, 2008; ROSSI; LANZETTA, 2013b), with limited buffers (BRUCKER; HEITMANN; HURINK, 2003; RONCONI, 2004; FÄRBER; COVES MORENO, 2006; FÄRBER; SALHI; COVES MORENO, 2008), with uncertain processing times (SWAMINATHAN et al., 2007), with sequence dependent setup times (LIN; YING; LEE, 2009; YING et al., 2010; MEHRAVARAN; LOGENDRAN, 2012), with dual resources (MEHRAVARAN; LOGENDRAN, 2013), with learning effects (VAHEDI-NOURI; FATTAHI; RAMEZANIAN, 2013b; VAHEDI-NOURI; FATTAHI; RAMEZANIAN, 2013a; VAHEDI-NOURI et al., 2014).

3.3 The job shop scheduling problem

The first paper about JSSP was published by Jackson (1956). This paper generalizes the algorithm of Johnson (1954) to the JSSP. Since then, many researchers have published works on the JSSP. *Web of Science* reports more than 4100 publications for a search of the words “job shop” or “jobshop” in the topic, and the most important among them (with about 600 citations) is the publication of Adams, Balas & Zawack (1988) that proposed the shifting bottleneck procedure (SBP). The SBP is the most important constructive heuristic for the JSSP. The SBP iteratively identifies and schedules the bottleneck machine. To do this, it solves the one-machine sequencing problem with release and due dates that minimizes the maximum lateness for each unscheduled machine, identifies the machine with the worst maximum lateness as the bottleneck machine, and schedules this bottleneck machine. After the scheduling of each machine, an optional reoptimization may be applied, by trying to reschedule each previously scheduled machine, one by one, solving the one-machine sequencing problem with updated release and due dates, and updating the partial schedule if there is a better one. This process is repeated until all machines are scheduled. Wenqi & Aihua (2004) proposed an improved version of the SBP that produces only feasible schedules, eliminating the deadlocks that the SBP had to detect and avoid. The improved SBP creates better solutions than the SBP, but with a higher computational cost.

Many metaheuristic approaches have been applied to the JSSP, such as tabu search (NOWICKI; SMUTNICKI, 1996; NOWICKI; SMUTNICKI, 2005; NASIRI; KIANFAR, 2012), genetic algorithms (VÁZQUEZ; WHITLEY, 2000; PÉREZ; POSADA; HERRERA, 2012; GONÇALVES; RESENDE, 2014; AMIRGHASEMI; ZAMANI, 2015), greedy randomized adaptive search procedure (BINATO et al., 2002; AIEX; BINATO; RESENDE, 2003), evolutionary algorithms (ESQUIVEL et al., 2002), artificial immune systems (COELLO; RIVERA; CORTÉS, 2003), ant colony optimization (PIN; XIAO-PING; HONG-FANG, 2004; ZHANG et al., 2006; HEINONEN; PETTERSSON, 2007), particle swarm optimization (GE et al., 2005; PONGCHAIRERKS; KACHITVICHYANUKUL, 2009), variable neighbourhood search (SEVKLI; AYDIN, 2006), global equilibrium search (PARDALOS; SHYLO, 2006), artificial bee colonies (CHONG et al., 2006; BANHARNSAKUN; SIRINAOVAKUL; ACHALAKUL, 2012), differential evolution (LIU et al., 2009; PONSICH; COELLO, 2010), backbone and big valley properties (PARDALOS; SHYLO; VAZACOPOULOS, 2010), cuckoo search (SINGH; SINGH, 2015), and hybrids of these methods (ZHANG et al., 2008; HUANG; LIAO, 2008; PONSICH; COELLO, 2013; ZHAO et al., 2015).

3.3.1 The neighbourhood of Nowicki & Smutnicki for the JSSP

One of the most important advances for the JSSP is the neighbourhood proposed by Nowicki & Smutnicki (1996). This neighbourhood is reduced to the interchange of the first two (or the last two) operations in a block that belongs to a single (arbitrarily selected) critical path, excluding the first two (and the last two) operations of that critical path. Figure 3.1 presents an example for the four moves allowed by this neighbourhood in a job shop schedule. The critical path of the schedule presents three blocks of operations that are surrounded by dashed lines. The first block on machine M_3 only interchanges its last two operations of jobs 5 and 2. The last block on machine M_2 only interchanges its first two operations of jobs 6 and 1. The block

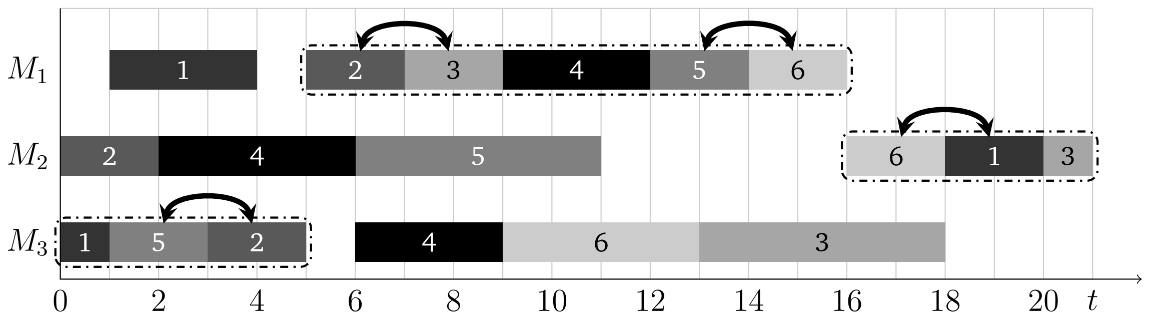


Figure 3.1: Neighbourhood of Nowicki & Smutnicki. Arrows indicate the possible movements. Dashed lines surround the blocks of the critical path.

on machine M_1 interchanges its first two operations of jobs 2 and 3 and its last two operations of jobs 5 and 6.

Nowicki & Smutnicki (1996) implemented this neighbourhood in a tabu search algorithm with back jump tracking (or TSAB), that stores each improved solution with its unexplored neighbourhood and the current tabu list, for future intensification in restarts after the search becomes stagnant. Their computational results showed that TSAB was capable of solving the large instances in very shorter times, performing better than all the contemporaneous methods in the known benchmarks, and becoming the base for future research.

Nowicki & Smutnicki (2005) improved the TSAB using a path relinking for diversification and proposed the i-TSAB. Watson, Howe & Whitley (2006) presented a detailed study of the i-TSAB and its components. They concluded that the good performance of i-TSAB is due to both the reduced neighbourhood operator and a balanced combination of intensification and diversification, showing that other methods that use the reduced neighbourhood achieve similar performance. Thus, since the publication of Nowicki & Smutnicki (1996), almost every intensification method uses their reduced neighbourhood.

3.4 Benchmarks from the literature

The most widely used benchmarks for scheduling problems were proposed by Taillard (1993). The benchmark of Taillard (1993) for FSSP comprises 120 instances, 10 instances for each size (20×5, 20×10, 20×20, 50×5, 50×10, 50×20, 100×5, 100×10, 100×20, 200×10, 200×20, and 500×20). The benchmark of Taillard (1993) for JSSP comprises 80 instances, 10 instances for each size (15×15, 20×15, 20×20, 30×15, 30×20, 50×15, 50×20, and 100×20). The benchmark of Taillard (1993) for the open shop comprises 60 instances, 10 instances for each size (4×4, 5×5, 7×7,

10×10, 15×15, and 20×20). The literature usually refers to the 120 Taillard FSSP instances are ta001–ta120, and to the 80 Taillard JSSP instances are tai01–tai80.

The next most commonly used benchmark for FSSP and JSSP was proposed by Demirkol, Mehta & Uzsoy (1998). They presented many instances with additional data for setup times and lateness, but many researchers use only the processing times and precedence constrains for completion time criteria. These and other commonly used benchmarks such as those proposed by Reeves (1995), Adams, Balas & Zawack (1988), Fischer & Thompson (1963), Carlier (1978), Applegate & Cook (1991), are available at the Operations Research Library maintained by Beasley (2005) with exception of Demirkol instances that are available at the websites of Oddi (2005) and Shylo (2005).

Recently, Vallada, Ruiz & Framinan (2015) discussed that the reduced number of open instances (or instances with unknown optimum and different upper and lower bounds), in the Taillard's benchmark for the FSSP, and the small relative deviations from the optimum obtained by heuristic methods difficult the evaluation and comparison of new methods. Consequently, they proposed a new benchmark, with 240 large instances for testing of heuristics (10 of each size with $n \in \{100, 200, \ldots, 800\}$ and $m \in \{20, 40, 60\}$), and 240 small instances for testing of exact methods (10 of each size with $n \in \{10, 20, \ldots, 60\}$ and $m \in \{5, 10, 15, 20\}$).

3.5 Heterogeneous workforce in the literature

The research regarding workforce planning focuses in many cases on mathematical models, disregarding the real life consequences of the simplifications made during the modeling process. The research regarding workforce management gives an extensive description of the human implications of management decisions, but fail to use customized mathematical models to provide support to their decisions. The area of Operations Research and the Management Sciences is progressively proposing approaches that address human diversity in the procedures for design, planning and control of productive systems. Next, we describe some examples of diversity inclusion that contribute to narrow the gap between research models and practice. De Bruecker et al. (2015) presented a review and classification of the literature regarding workforce planning problems incorporating skills.

Daniels, Mazzola & Shi (2004) studied the assignment problem of skilled workers to each operation of a the FSSP. The processing time of an operation depends on the number of workers assigned to the corresponding work station (machine) in that period. Only trained (skilled) workers can be assigned to a station, and each worker is trained to work on at least one station. They proposed small instances of this new problem and a branch-and-bound algorithm that solves them optimally.

Zhang, Song & Wu (2012) studied an stochastic model for the JSSP with minimization of the expected total weighted tardiness as optimization criterion. In their model, the processing times are independent random variables with known expected value and variance. They proposed a particle swarm optimization with two stages for the problem. In the first stage, the solutions are evaluated with a fast lower bound and are improved with a local search, quickly converging to high-quality regions in the solution space. The second stage focuses the search on finding the best solutions, evaluating them with a more accurate (but expensive) Monte Carlo simulation. The method finds solutions of good quality.

Wang & Zhang (2015) studied the PFSSP with minimization of the weighted sum of makespan and total completion time with learning effects. The learning effect decreases the processing time of the operations as workers gain more experience. They proposed a constructive heuristic and a branch and bound algorithm for the problem. The constructive heuristic finds near optimal solutions in short time.

Li & Womer (2009) studied a project scheduling problem with assignment of multi-skilled personnel. The personnel members have (or miss) certain skills from a determined set of skills, and they have a limited availability. The jobs of the project have release and due dates, and may be performed only by personnel that fulfill the required skills. A hybrid algorithm that combines mixed-integer linear programming (MILP) and constraint programming (CP) was used to solve instances of the new problem, showing better results compared to pure MILP or CP methods.

Fırat & Hurkens (2012) studied the assignment of technicians to tasks with multi-level skill requirements proposed in the 2007 ROADEF Challenge. The technicians have different levels of specialization for each skill and are not available on all days of the planning horizon. Each task has a fixed duration, precedence relations, and the number of technicians for each skill level that it requires for its execution. Thus, a schedule must present the processing time for the tasks and the groups of technicians with the necessary skills to process each task. They proposed a flexible matching model to solve this problem that produces solutions of good quality.

Regarding assembly lines, the basic hypothesis of standardized task times was totally assumed in the classic literature with the only exception of Mansoor (1968) and Bartholdi & Eisenstein (1996), that analyzed the case in which workers have different work speeds in a particular assembly line, the Toyota Swen System. Later, Gel, Hopp & Van Oyen (2002) and Hopp, Tekin & Van Oyen (2004) studied the case where the workers were categorized as fast or slow according to their performance, and Erel, Sabuncuoglu & Sekerci (2005) studied an stochastic model for the assembly line balancing problem that presents a random variation in the processing times from item to item, due to the experience, willingness, distractions, fatigue and other factors affecting the performance of the workers. They proposed a beam search that finds good configurations for the assembly line minimizing the expected total costs. Also Corominas, Pastor & Plans (2008) studied the assembly line balancing problem to minimize the number of required temporary workers that must be hired to attain a production goal. They categorize the workers as skilled (permanent) and unskilled (temporary). The unskilled workers can only perform a subset of tasks with a processing time larger than that of a permanent worker by a fixed factor. The problem was taken from a real motorcycle assembly plant. The problem was modeled using binary linear programming and was optimally solved using the CPLEX optimizer.

Following this trend, Miralles et al. (2007) defined the Assembly Line Worker Assignment and Balancing Problem (ALWABP) that focuses in the heterogeneity of task times and the presence of incompatibilities, defining a new set of realistic hypotheses. The objective was the minimization of the cycle time given a set of tasks to be assigned to heterogeneous workers in successive stations, as it was the most common situation at SWD assembly lines that originally inspired the problem. The philosophy was to assign the workers those tasks they can handle while trying to avoid those more difficult or impossible tasks, thus making invisible the disabilities while maximizing productivity. Due to the problem complexity, other authors focused on heuristic and metaheuristic approaches such as clustering search (CHAVES; MIRALLES; LORENA, 2007; CHAVES; LORENA; MIRALLES, 2009), tabu search (MOREIRA; COSTA, 2009), beam search (BLUM; MIRALLES, 2011), constructive heuristics (MOREIRA et al., 2012), genetic algorithms (MOREIRA et al., 2012; MUTLU; POLAT; SUPCILLER, 2013), and branch and bound based methods (MIRALLES et al., 2008; VILÀ; PEREIRA, 2014; BORBA; RITT, 2014).

3.6 Concluding remarks

In this chapter we presented a brief survey on the flow shop scheduling problem, showing that the 99% of the literature focus on the permutation variation. We also presented a more detailed survey on the non-permutation variation of the flow shop scheduling problem. As the general flow shop scheduling problem may be considered a special case of the job shop scheduling problem, we also reviewed the literature on the job shop scheduling problem. Finally, we presented many examples in the literature that successfully introduce human diversity in the planning and scheduling processes of productive systems.

Part II

Heuristics for the non-permutation flow shop scheduling problem with makespan and total completion time criteria

4 THE PERMUTATION AND NON-PERMUTATION FLOW SHOPS

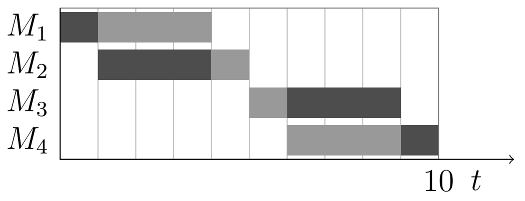
As defined in Section 2.2.1, a flow shop that has the same processing order on all machines, such as the schedule in Figure 4.1b, is called a *permutation flow shop*, because only one permutation of the jobs is necessary to represent a schedule. A flow shop that allows different processing orders on the machines is called *non-permutation flow shop*, *general flow shop*, or plainly *flow shop*, such as the schedule in Figure 4.1a that has two different processing orders, one on the two first and another on the two last machines.

Chapter 3 shows that almost all of the research on flow shop scheduling has focused on the development of procedures to obtain permutation schedules. The most probable reason is that the general or non-permutation version of the FSSP is more difficult than the permutation version. The number of feasible schedules is $n!^m$ for a flow shop instance with n jobs and m machines. The consideration of the same job permutation on all machines reduces the space of solutions to $n!$ possible permutations, disregarding the number of machines.

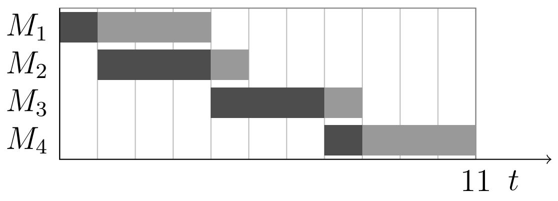
This enormous reduction of the search space is a direct consequence of excluding $m-1$ permutations, but the other direct consequence is the elimination of possible better non-permutation schedules. For example, the instance of the flow shop problem in Table 4.1 has a best possible permutation schedule with a makespan of 11 (Figure 4.1b), but if we also consider non-permutation schedules, there is a better non-permutation schedule with a makespan of 10 (Figure 4.1a) that is the optimal for this instance. Potts, Shmoys & Williamson (1991) give a family of flow shop instances whose makespan values between the best permutation schedule and the optimal non-permutation schedule may differ by a factor greater than $1/2(\sqrt{m}+1/2)$. Other practical studies such as Tandon, Cummings & LeVanu (1991) and Liao, Liao & Tseng (2006) show that the makespan of non-permutation schedules is in average 2% shorter than the makespan of permutation schedules for the tested instances.

Table 4.1: 2×4 instance of the FSSP.

Jobs	Operations			
	M_1	M_2	M_3	M_4
J_1	1	3	3	1
J_2	3	1	1	3



a. Non-permutation flow shop.

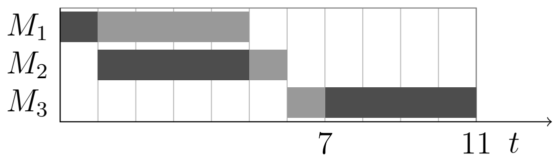


b. Permutation flow shop.

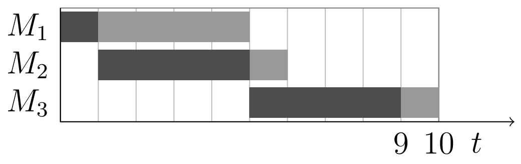
Figure 4.1: Optimal non-permutation and best possible permutation schedules for minimizing makespan.

Table 4.2: 2×3 instance of the FSSP.

Jobs	Operations		
	M_1	M_2	M_3
J_1	1	4	4
J_2	4	1	1



a. Non-permutation flow shop.



b. Permutation flow shop.

Figure 4.2: Optimal non-permutation and best possible permutation schedules for minimizing total completion time.

The situation is similar when considering the total completion time criterion in flow shop scheduling problems. The instance of the flow shop problem in Table 4.2 has a best possible permutation schedule with a total completion time of 19 (Figure 4.2b), but if we also consider non-permutation schedules, there is a better non-permutation schedule with a total completion time of 18 (Figure 4.2a) that is the optimal for this instance.

Conway, Maxwell & Miller (1967) proved that there exists an optimal solution having the same processing order (permutation) on the first two machines for any regular optimality criterion such as makespan and total completion time. As the makespan flow shop problem is symmetric, i.e., it is equivalent to the reverse problem with an inverted order of machines, they also proved that there exists an optimal solution having the same processing order on the last two machines for the makespan criterion. These properties reduce the number of feasible flow shop schedules to $n!^{(m-2)}$ for the makespan criterion, and to $n!^{(m-1)}$ for the total completion time criterion. These properties also show that the instances of Tables 4.1 and 4.2 are the smallest examples for the makespan and the total completion time criteria respectively, that present better non-permutation schedules.

4.1 Non-permutation insertion with anticipation or delay

We observed in preliminary experiments that non-permutation schedules with the shortest makespan values have similar processing orders in subsequent machines, with one or two jobs changing their positions in the processing order. An easy way to create non-permutation schedules with this characteristic is to modify the NEH heuristic (described in Section 3.1.1) to insert jobs with a slight difference in the processing order between two intermediate machines.

The NEH heuristic evaluates the insertion of the next job into a partial schedule straight in the same position on all machines, as shown in Figure 4.3. Now, we may choose to insert a job with anticipation after an intermediate machine as shown in Figure 4.4. An insertion with anticipation means that we insert a job in a certain position k on the first machines, and after some intermediate machine, we hold the operation scheduled immediately before position k, and insert the new job in position $k-1$ on the next machines. We also may choose to insert a job with delay after an intermediate machine as shown in Figure 4.5. An insertion with delay means that we insert a job in a certain position k on the first machines, and after some intermediate machine, we anticipate the operations scheduled immediately after position k and insert the new job in position $k+1$ on the next machines.

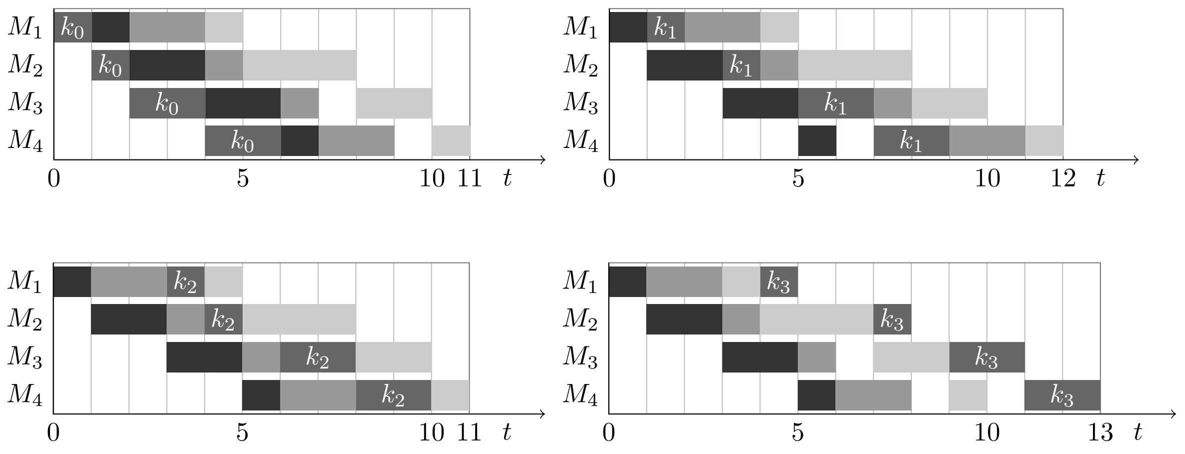
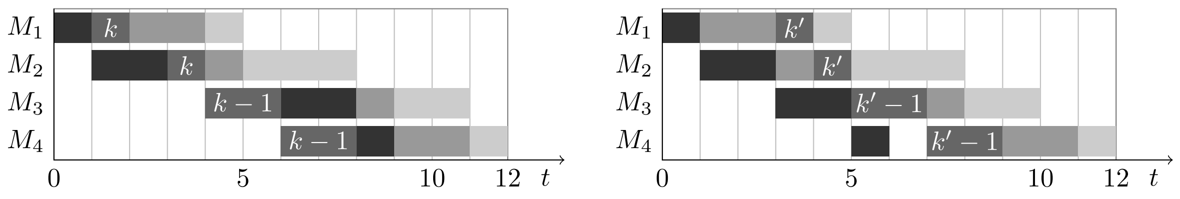


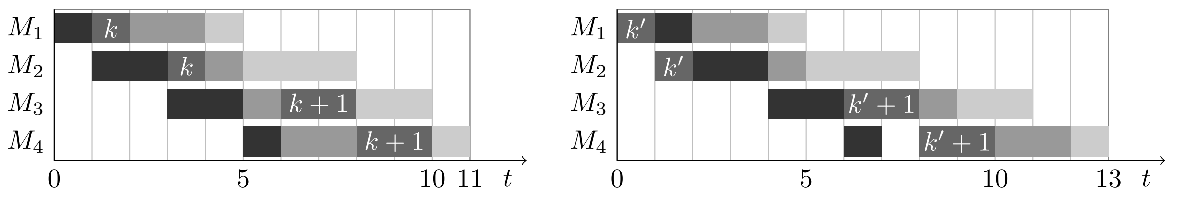
Figure 4.3: Possible positions to insert a job that the NEH heuristic evaluates. Note that the job is inserted straight in the same position on all machines.



a. Insertion of a job at position $k = 2$ with anticipation after machine M_2.

b. Insertion of a job at position $k' = 3$ with anticipation after machine M_2.

Figure 4.4: Example of job insertions with anticipation in non-permutation schedules.



a. Insertion of a job at position $k = 2$ with delay after machine M_2.

b. Insertion of a job at position $k' = 1$ with delay after machine M_2.

Figure 4.5: Example of job insertions with delay in non-permutation schedules.

Algorithm 4.1 A NEH-like constructive heuristic for the non-permutation FSSP.

```
Input: The processing times $p_{ij}$ for each job $J_j$ on each machine $M_m$.
Output: A permutation schedule $\pi$.
 1: function NEH_LIKE_CONSTRUCTIVE_HEURISTIC( )
 2:    $\pi_o := (\pi_o(1), \ldots, \pi_o(n))$ in non-increasing total processing time
 3:    $\pi := (\pi_o(1))$
 4:    for $\pi_o(j), j \in [2, n]$ do
 5:       for all insertion positions $k \in [j]$ do
 6:          evaluate insertion of $\pi_o(j)$ at $k$ with anticipation after $M_i$ with $i \in [2, m-2]$
 7:          evaluate insertion of $\pi_o(j)$ at $k$ with delay after $M_i$ with $i \in [2, m-2]$
 8:          evaluate insertion of $J_j$ at $k$ straight
 9:       end for
10:       Apply the best insertion of job $\rho_o(j)$ into $\pi$ which minimizes $C_{\max}$
11:    end for
12:    return $\pi$
13: end function
```

Algorithm 4.1 presents a NEH-like heuristic for the non-permutation FSSP that evaluates insertions with anticipation and delay, besides the straight insertions of the original NEH. The number of different insertion possibilities grows from $O(n)$ to $O(nm)$ due to the evaluation of anticipation and delay on different machines for each insertion position. If each total completion time evaluation costs $O(nm)$, a NEH-like heuristic for the non-permutation FSSP inserts n jobs with a time complexity of $O(n^3m^2)$. The acceleration technique of Taillard reduces the time complexity of NEH from $O(n^3m)$ to $O(n^2m)$, but it cannot be used directly with representations for non-permutation schedules, as we show in Chapter 6.

4.2 Concluding remarks

The main conclusion of this chapter is that the elimination of non-permutation schedules reduces the search space of the flow shop scheduling problem, but it also eliminates optimal non-permutation schedules. We propose a non-permutation insertion with anticipation or delay that introduces small changes in the processing order of subsequent machines to create non-permutation schedules with a NEH-like heuristic. We use this idea to propose heuristics for the non-permutation FSSP in the next three chapters. We redefine this idea in each chapter according to the corresponding representation and optimization criteria.

5 ITERATED LOCAL SEARCH HEURISTICS FOR MINIMIZING THE TOTAL COMPLETION TIME IN PERMUTATION AND NON-PERMUTATION FLOW SHOPS

Section 2.2.1 has defined the permutation and non-permutation FSSP formally, and Chapter 4 has explored the theoretical differences between the permutation and non-permutation FSSP. The permutation FSSP is easier to solve, but also eliminates the posibility of finding a better (or an optimal) non-permutation schedule. Optimal non-permutation schedules only show small changes in the processing order of subsequent machines. This observation may be exploited to reduce the search space of the non-permutation FSSP.

This chapter studies the practical differences between the permutation and the non-permutation FSSP with the total completion time criterion, and the degree of effort needed to find good non-permutation schedules with the proposed heuristics. Section 5.1 introduces the concepts of iterated local search, and describes the components of the iterated local search that is proposed to solve the permutation FSSP. Section 5.2 introduces the basic concepts of iterated greedy algorithms, and describes the iterated greedy algorithm that is proposed to solve the non-permutation FSSP. Section 5.4 presents and analyzes computational experiments. Finally, we conclude in Section 5.5. This chapter corresponds to the publication of Benavides & Ritt (2015).

5.1 An iterated local search for the permutation FSSP

An iterated local search (ILS) is an stochastic local search that uses iteratively an embedded heuristic with small perturbations, producing better solutions than by randomly repeating the same heuristic. Algorithm 5.1 presents the general structure of an ILS. The main components of an ILS are the *perturbation scheme*, the *local search scheme* and the *acceptance criterion*. ILS starts from an initial local minimum obtained

Algorithm 5.1 Iterated local search.

```
Input: An initial solution $s$.
Output: The best solution $s^*$ found during the search.
 1: function ILS($s$)
 2:     $s :=$ LOCAL_SEARCH($s$)
 3:     repeat
 4:         $s' :=$ PERTURB($s$)
 5:         $s' :=$ LOCAL_SEARCH($s'$)
 6:         if ACCEPTANCE_CRITERION($s, s'$) then
 7:             $s := s'$
 8:             Update $s^*$ if necessary
 9:         end if
10:     until some stopping criterion is satisfied
11:     return $s^*$
12: end function
```

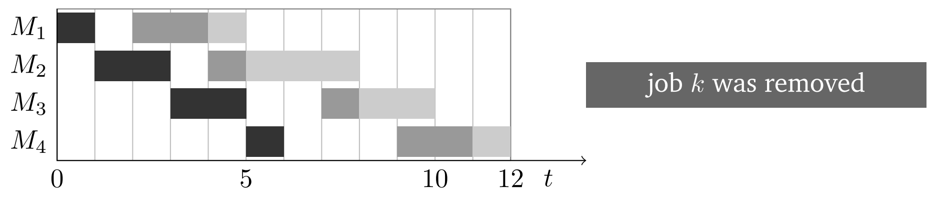
from a local search. Then, ILS repeatedly applies a perturbation to escape from the current local minimum, followed by a local search to find another local minimum, until some stopping criterion is satisfied. If the new local minimum is better than the current one, the corresponding solution is accepted. Sometimes, the acceptance criterion allows a slightly worse new local minimum to become the current local minimum, in order to diversify the search. Finally, ILS returns the best solution found during the search. The performance of an ILS depends on the perturbation strength, on the structure of the neighbourhoods, and on the pliability of the acceptance criterion. Next, we explain the ILS components to produce permutation schedules.

5.1.1 Perturbation scheme for permutation schedules

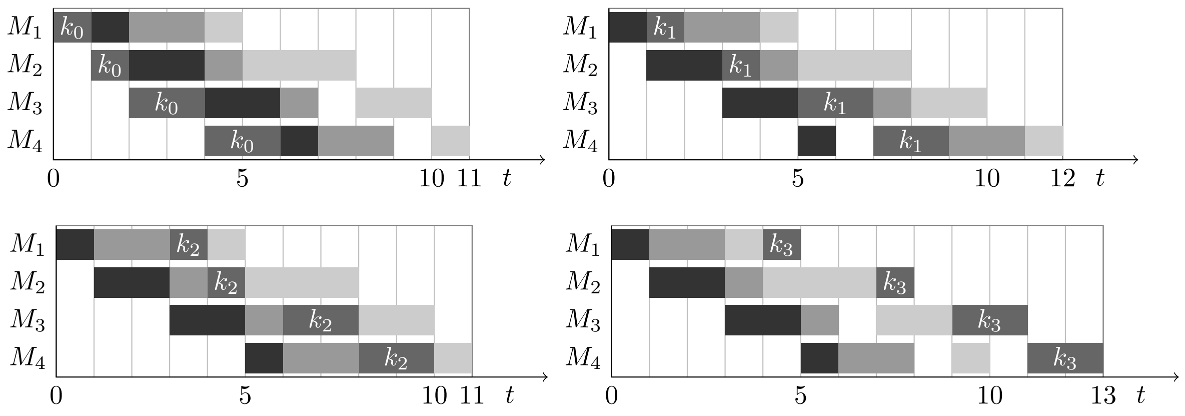
A perturbation which is too weak may lead to stagnation, while a too strong perturbation can turn the algorithm into a randomized multi-start local search. The perturbation strength of our algorithm is calibrated with the parameter d, that

Algorithm 5.2 Perturbation scheme for permutation schedules.

```
Input: A permutation schedule $\pi$, a number $d$ of jobs to perturb.
Output: A perturbed permutation schedule $\pi'$.
 1: function PERTURB_PERMUTATION_SCHEDULE($\pi$,$d$)
 2:     remove $d$ random jobs $J_1, \dots, J_d$ from $\pi$ to get $\pi'$
 3:     for $j \in [d]$ do
 4:         insert $J_j$ into $\pi'$ at the position which minimizes $C_{\text{sum}}(\pi')$
 5:     end for
 6:     return $\pi'$
 7: end function
```



a. Permutation schedule with a removed job.



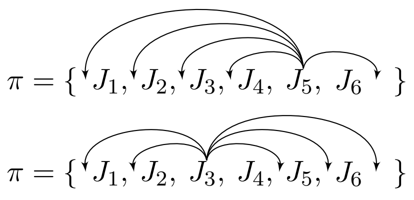
b. Possible job reinsertions that produce a permutation schedule.

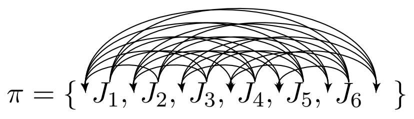
Figure 5.1: Example for perturbation scheme in permutation schedules.

represents the number of reconstructed jobs in the schedule. Algorithm 5.2 presents the perturbation scheme used to produce permutation schedules. It starts with a complete solution π, and removes d random jobs to create a partial solution π'. Then, it iteratively reinserts the removed jobs with a greedy construction heuristic into a position that minimizes the objective function of the partial schedule, until the solution is complete. Figure 5.1 gives an example for the permutation scheme: Figure 5.1a shows an schedule with a removed job, and Figure 5.1b shows the possible permutation schedules that result from the different job reinsertions.

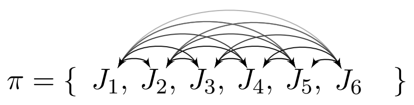
5.1.2 Local search scheme for permutation schedules

The most common neighbourhoods for local search on permutation schedules are swapping two arbitrary pairs of jobs and shifting a job to another position. They have repeatedly been identified as the most effective ones (PAN; RUIZ, 2013; DUBOIS-LACOSTE; LÓPEZ-IBÁÑEZ; STÜTZLE, 2011; TASGETIREN et al., 2011; RAJENDRAN; ZIEGLER, 2004; JARBOUI; EDDALY; SIARRY, 2009). Preliminary tests showed that the order in which the neighbours are evaluated has a small but consistent effect on the quality of the final solution. Consequently, our implementations visit each neighbourhood in a specific order. We also impose a repetition limit r on the number of neighbours to be evaluated within a run of each local search. Next we present details about both neighbourhoods and the corresponding local search algorithms.


$\pi = \{ J_1, J_2, J_3, J_4, J_5, J_6 \}$
$\pi = \{ J_1, J_2, J_3, J_4, J_5, J_6 \}$
a. Shift neighbourhood moves.



b. Complete shift neighbourhood.



c. Complete swap neighbourhood.

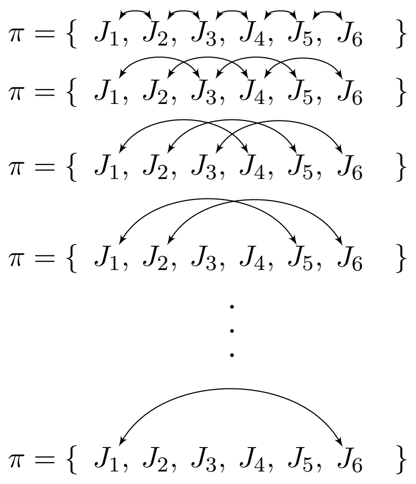

$\pi = \{ J_1, J_2, J_3, J_4, J_5, J_6 \}$
$\pi = \{ J_1, J_2, J_3, J_4, J_5, J_6 \}$
$\pi = \{ J_1, J_2, J_3, J_4, J_5, J_6 \}$
$\pi = \{ J_1, J_2, J_3, J_4, J_5, J_6 \}$
$\pi = \{ J_1, J_2, J_3, J_4, J_5, J_6 \}$
d. Incremental distance exploration for swap neighbourhood.

Figure 5.2: Examples for local search neighbourhoods.

5.1.2.1 *Randomized shift local search*

For a permutation schedule π, where π is a permutation of the index set $[n]$ of the jobs, a shift of the job at position $i \in [n]$ to position $j \in [n+1] \setminus \{i, i+1\}$ results in the permutation $(\pi(1), \dots, \pi(j-1), \pi(i), \pi(j), \dots, \pi(i-1), \pi(i+1), \dots, \pi(n))$. For example, Figure 5.2a shows possible shifts from jobs J_5 and J_3 to other positions in the permutation, and Figure 5.2b shows all the possible moves in the shift neighbourhood. The size of the shift neighbourhood is $(n-1)^2$. Algorithm 5.3 describes our randomized shift local search. It iteratively evaluates the relocation of a job in every position of the permutation. Note that the jobs to be relocated are examined in a random order. The parameter r limits the number of neighbours to be evaluated within a run of the local search.

Algorithm 5.3 Randomized shift local search.

Input: A permutation schedule π, a repetition limit r.
Output: A permutation schedule π' with $C_{\text{sum}}(\pi') \leq C_{\text{sum}}(\pi)$.

```
 1: function RANDOMIZED_SHIFT_LOCAL_SEARCH($\pi$,$r$)
 2:     repeat at most $r$ times
 3:         for $j \in [n]$ in some random order do
 4:             let $\pi'$ be the result of the best shift of job $\pi(j)$
 5:             let $\pi$:=$\pi'$ if $C_{\text{sum}}(\pi') < C_{\text{sum}}(\pi)$
 6:             return $\pi$, if the last $n$ shifts did not improve
 7:         end for
 8:     until $\pi$ did not improve
 9:     return $\pi$
10: end function
```

5.1.2.2 *Swap local search*

For a permutation schedule π, where π is a permutation of the index set $[n]$ of the jobs, a swap of positions $i, j \in [n]$ exchanges the jobs $\pi(i)$ and $\pi(j)$. For example, Figure 5.2c shows all the possible moves in the swap neighbourhood. The size of the swap neighbourhood is $\binom{n}{2}$. Algorithm 5.4 describes our swap local search. It iteratively evaluates the exchange of two jobs in the permutation, starting with adjacent jobs and later swapping jobs with an increasing distance, as shown in Figure 5.2d. When an improvement is found, the distance of the swapped jobs is reset to 1. Again, the parameter r limits the number of neighbours to be evaluated within a run of the local search.

Algorithm 5.4 Swap local search.

```
Input: A permutation schedule $\pi$, a repetition limit $r$.
Output: A permutation schedule $\pi'$ with $C_{\text{sum}}(\pi') \leq C_{\text{sum}}(\pi)$.
 1: function SWAP_LOCAL_SEARCH($\pi$,$r$)
 2:     $d := 1$
 3:     while $d \leq n$ and up to $rn^2$ swaps do
 4:         for $j \in [n-d]$ do
 5:             swap jobs $\pi(j)$ and $\pi(j+d)$ to get $\pi'$
 6:             let $\pi := \pi'$, if $C_{\text{sum}}(\pi') < C_{\text{sum}}(\pi)$
 7:         end for
 8:         $d := d + 1$
 9:         if $\pi$ improved then reset $d := 1$
10:     end while
11:     return $\pi$
12: end function
```

5.1.3 Acceptance criterion

The acceptance criterion is based on that proposed by Metropolis et al. (1953). The new solution s' replaces the current solution s with probability

$$P[\text{acceptance\_criterion}(s, s')] = \min\{e^{-(C_{\text{sum}}(s')-C_{\text{sum}}(s))/T}, 1\}$$

for a *temperature* $T = \alpha\bar{p}n/10$, where $\bar{p} = \sum_{j\in[n]}\sum_{i\in[m]} p_{ij}/nm$ is the average processing time of an operation. The pliability of the acceptance criterion is calibrated with the parameter α. A similar criterion has been proposed by Osman & Potts (1989) for a simulated annealing heuristic to minimize the makespan in permutation flow shops and has been successfully applied in several iterated local search algorithms. The temperature has been adjusted by a factor of n to take into account the higher values of the total completion time function.

5.1.4 Complete iterated local search for the permutation FSSP

Algorithm 5.5 presents our iterated local search for permutation schedules. The initial schedule is obtained by the constructive heuristic LR(n/m) of Liu & Reeves (2001) applied to n/m initial sequences, and the random shift local search. Then, the current solution is repeatedly perturbed and improved by a local search. Both local searches are used alternately to take advantage of their complementary neighbourhoods. The acceptance criterion determines the replacement of the current solution π by the newly produced solution π'. Finally, the ILS returns the best permutation schedule found before the stopping criterion was satisfied.

Algorithm 5.5 Iterated local search for permutation schedules.

```
Input: A number $d$ of jobs to perturb, a repetition limit $r$.
Output: The best permutation schedule $\pi^*$ found during the search.
 1: function ILS($d$,$r$)
 2:     $\pi :=$ LR$(n/m)$
 3:     $\pi :=$ RANDOM_SHIFT_LOCAL_SEARCH$(\pi, r)$
 4:     repeat
 5:         $\pi' :=$ PERTURB_PERMUTATION_SCHEDULE$(\pi, d)$
 6:         if current iteration is even then
 7:             $\pi' :=$ SWAP_LOCAL_SEARCH$(\pi', r)$
 8:         else
 9:             $\pi' :=$ RANDOM_SHIFT_LOCAL_SEARCH$(\pi', r)$
10:         end if
11:         $\pi := \pi'$, with probability $P[\text{acceptance\_criterion}(\pi, \pi')]$
12:         $\pi^* := \pi$, if $\pi$ is better than $\pi^*$
13:     until some stopping criterion is satisfied
14:     return $s^*$
15: end function
```

5.2 An iterated greedy algorithm for the non-permutation FSSP

An Iterated Greedy Algorithm (IGA) is a stochastic local search that iteratively produces solutions by perturbing a solution using a greedy construction heuristic. IGA is closely related to ILS with a greedy construction as the underlying heuristic. Algorithm 5.6 presents the general structure of an IGA. An IGA starts from some initial solution that may also be constructed with the greedy construction heuristic. Then, it repeatedly applies a perturbation to the current solution, by eliminating some of its elements in a destruction phase, to later reconstruct a new complete solution by applying a greedy construction phase. This is repeated until some stopping criterion is satisfied. If the new solution is better than the current one, it is accepted. As in the ILS, the acceptance criterion sometimes allows an slightly worse new solution to

Algorithm 5.6 Iterated greedy algorithm.

```
Input: A number $d$ of elements to perturb.
Output: The best solution $s^*$ found during the search.
 1: function IGA($d$)
 2:     $s$ := GENERATE_INITIAL_SOLUTION( )
 3:     repeat
 4:         $s_p$ := DESTRUCTION_PHASE($s, d$)
 5:         $s'$ := GREEDY_CONSTRUCTION_PHASE($s_p, d$)
 6:         if ACCEPTANCE_CRITERION($s, s'$) then
 7:             $s := s'$
 8:             Update $s^*$ if necessary
 9:         end if
10:     until some stopping criterion is satisfied
11:     return $s^*$
12: end function
```

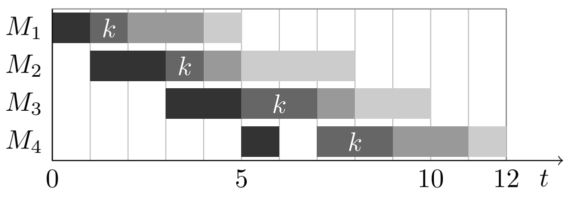
become the current solution, in order to diversify the exploration. Finally, an IGA returns the best solution found during the search.

5.2.1 Perturbation scheme for non-permutation schedules

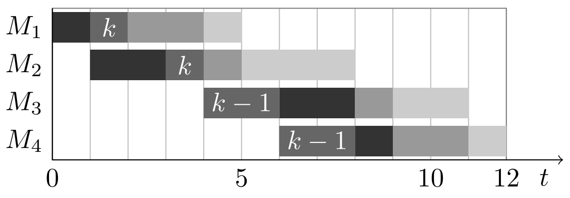
The main component of an IGA is the perturbation scheme that consists of the destruction and reconstruction phases. Algorithm 5.7 presents the perturbation scheme used to produce non-permutation schedules. The parameter d adjusts the number of elements to be destructed and reconstructed, thus calibrating the

Algorithm 5.7 Perturbation scheme for non-permutation schedules.

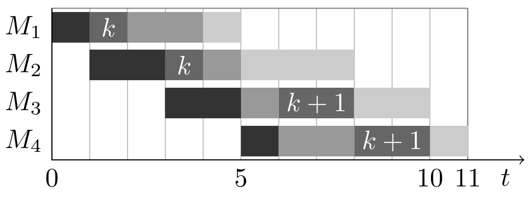
```
Input: A non-permutation schedule $s = (\pi_1, \dots, \pi_m)$,
       a number $d$ of jobs to perturb.
Output: A perturbed non-permutation schedule $s' = (\pi'_1, \dots, \pi'_m)$.
 1: function PERTURB_NON_PERMUTATION_SCHEDULE($s$,$d$)
 2:     remove $d$ random jobs $J_1, \dots, J_d$ from each $\pi_i \in s$ to get $\pi'_i \in s'$
 3:     for $j \in [d]$ do
 4:         for all positions $k \in [n]$ do
 5:             evaluate insertion of $J_j$ at $k$ with anticipation
                        after machine $M_i$ with $i \in [2, m-1]$
 6:             evaluate insertion of $J_j$ at $k$ with delay
                        after machine $M_i$ with $i \in [2, m-1]$
 7:             evaluate insertion of $J_j$ at $k$ straight
 8:         end for
 9:         apply the best insertion of $J_j$ to $s'$
10:     end for
11:     return $s'$
12: end function
```



a. Insertion of a job at a straight position $k = 2$.



b. Insertion of a job at position $k = 2$ with anticipation after machine M_2.



c. Insertion of a job at position $k = 2$ with delay after machine M_2.

Figure 5.3: Example of job insertions in non-permutation schedules.

strength of our perturbation. It starts with a complete non-permutation schedule $s = (\pi_1, \ldots, \pi_m)$, and it removes d random jobs to create a partial solution $s' = (\pi'_1, \ldots, \pi'_m)$. Then, it iteratively reinserts the removed jobs with a greedy construction heuristic until a new non-permutation schedule is complete.

Our main hypothesis for finding good non-permutation schedules is that only limited changes to the processing order between two consecutive machines need to be considered during the search. We propose an extended job insertion procedure which reflects this principle. When inserting a job at some position into a partial schedule, we evaluate job passing: the processing of a job may be anticipated or delayed at some intermediate machine. Figure 5.3 gives examples of job insertions without job passing, with anticipation, and with delay. Formally, let $\pi_1, \ldots, \pi_m$ be a partial non-permutation schedule, with j jobs. Inserting a job J at position $k \in [2, j+1]$ with anticipation after machine M_i with $i \in [2, m-1]$ results in putting job J at position k into $\pi_1, \ldots, \pi_i$, and at position $k-1$ into $\pi_{i+1}, \ldots, \pi_m$. Similarly, inserting a job J at position $k \in [j]$ with delay after machine M_i with $i \in [2, m-1]$ results in putting job J at position k into $\pi_1, \ldots, \pi_i$, and at position $k+1$ into $\pi_{i+1}, \ldots, \pi_m$.

5.2.2 Complete iterated greedy algorithm for the non-permutation FSSP

Algorithm 5.8 presents our iterated greedy algorithm for non-permutation schedules. It starts from some permutation schedule obtained by ILS. Then, the current solution is repeatedly destructed and reconstructed by the perturbation scheme for non-permutation schedules until a stopping criterion is satisfied. The acceptance criterion to determine the current solution replacement is the same of ILS. Finally, the IGA returns the best solution found during the search.

Algorithm 5.8 Iterated greedy algorithm for non-permutation schedules.

```
Input: A number $d$ of jobs to perturb, a repetition limit $r$.
Output: A non-permutation schedule $s = (\pi_1, \ldots, \pi_m)$.
 1: function IGA($d$,$r$)
 2:     $\pi :=$ ILS( )
 3:     let $s := (\pi, \pi, \ldots, \pi)$
 4:     repeat
 5:         $s' :=$ PERTURB_NON_PERMUTATION_SCHEDULE($s, d$)
 6:         $s := s'$, with probability $P[\text{acceptance\_criterion}(s, s')]$
 7:         $s^* := s$, if $s$ is better than $s^*$
 8:     until some stopping criterion is satisfied
 9:     return $s^*$
10: end function
```

5.3 Evaluation of schedules

A non-permutation schedule $s = (\pi_1, \ldots, \pi_m)$ is defined by the permutations π_i of the job index set $[n]$ on each of the m machines. A permutation schedule may be represented by a non-permutation schedule $s = (\pi_1 = \pi, \ldots, \pi_m = \pi)$ with the same permutation $\pi_i = \pi$ for all the machines. Then, $\pi_i(k)$ is the k-th job on machine M_i and its completion time is

$$C_{i,\pi_i(k)} = \max\{C_{i,\pi_i(k-1)}, C_{i-1,\pi_{i-1}(k)}\} + p_{i,\pi_i(k)},$$

where $C_{i,\pi_i(0)} = 0$ and $C_{0,\pi_0(k)} = 0$. The completion time of job J_j is $C_j = C_{m,j}$ and the total completion time is $C_{\text{sum}} = \sum_{j \in [n]} C_j$. Consequently, computing the total completion time needs time $\Theta(nm)$ for any permutation or non-permutation schedules.

Clearly, when the order of some operations has changed, only the modified completion times have to be updated. Moreover, Duan et al. (2013) observed that the completion times of the operations on the last machines depend only on the completion times of the operations on the critical path. These speed-up techniques can be used to reduce the number of completion times that must be updated. We use both techniques in our algorithms to compute updated total completion times.

The number of resulting schedules after the insertion of a job in a permutation schedule is $O(n)$, and the total time required to find the insertion position such that the total completion is minimized is $O(n^2m)$. The insertion of a job with anticipation or delay after some machine increases the number of resulting non-permutation schedules to $O(nm)$. Thus, the total time required to find the insertion position and the machine after which the job must be anticipated or delayed, such that the total completion is minimized is $O(n^2m^2)$.

5.4 Computational Experiments

We report the results of two computational tests. The first compares the quality of the permutation schedules obtained by our iterated local search to state-of-the-art methods from the literature. The second test compares the quality of non-permutation schedules to permutation schedules. Additionally, we study the amount of job reordering between consecutive machines in non-permutation schedules, and we compare the buffer requirements of permutation and non-permutation schedules.

5.4.1 Experimental methodology

We have tested our algorithms on the 120 instances proposed by Taillard (1993) and described in Section 3.4, which are the standard benchmark in the literature. We present the quality of the results as relative percentage deviations $(C_{\text{sum}} - C^*_{\text{sum}})/C^*_{\text{sum}} \times 100$ from the best known values C^*_{sum} reported by Pan & Ruiz (2012).

It is common to compare metaheuristics for flow shops using a time limit of $\tau nm\,\text{ms}$, for some constant τ. We use three different time scales in our experiments, as shown in Table 5.1. For the ILS we adopt the shortest time scale $\tau = 30$ used in the literature. We further use a time scale of $\tau = 60$ for the ILS and the IGA, since this is the total time needed for the two phases of the IGA. Finally, since finding non-permutation schedules is considerably harder, we report results for a longer time limit of $30nm^2$ for the IGA.

Table 5.1: Time scales for the experiments.

FSSP heuristic	ILS (permutation)	IGA (non-permutation)
Time limit (milliseconds)	$30nm$	
	$60nm$	$60nm$
		$30nm^2$

Our algorithms have been implemented in C++, compiled with g++ version 4.7.3 and run on a PC with an eight-core AMD FX-8150 processor running at 3.6 GHz and with 32 GB of main memory, using only one core in each execution.

5.4.2 Calibration of parameters

Both shift and swap local searches use a repetition limit $r = 3$ to control the number of evaluated neighbours. This value was calibrated from the set $\{1, 2, 3, 4, \infty\}$

in preliminary tests, confirming the value obtained by Dubois-Lacoste, López-Ibáñez & Stützle (2011).

Both ILS and IGA depend on two parameters: the number of jobs d to remove and reinsert in a perturbation and the temperature factor α. We use the R package irace (LÓPEZ-IBÁÑEZ et al., 2011) which implements an iterative F-Race (BALAPRAKASH; BIRATTARI; STÜTZLE, 2007) to tune these parameters. An F-Race generates random parameters settings and compares their performance by applying the non-parametric Friedman test for comparing multiple blocks and treatments and post-hoc tests to identify the best parameter settings. The iterated version repeats this process using the best parameter settings from the previous iteration to generate new ones. We chose the three hard instances ta061, ta071, and ta081 for these tests. The racing algorithm was run with a budget of 2000 executions for both ILS and IGA. Table 5.2 shows the tested ranges of values and the values obtained after the calibration for both ILS and IGA. The parameter search ranges were chosen based on values of the literature and preliminary tests.

Table 5.2: Calibration of parameters.

	Permutation ILS		Non-permutation IGA	
Parameter	Tested	Obtained	Tested	Obtained
α	$[0.01, 0.25]$	0.2353	$[0.0125, 2.0]$	0.146
d	$[1, 15]$	8	$[1, 10]$	2

5.4.3 Quality of permutation schedules

Table 5.3 compares our results to the best heuristic (PR) reported by Pan & Ruiz (2012) for the same time limits of $30nm$ ms and $60nm$ ms. PR is an iterated greedy algorithm reported as the best of 4 proposed heuristics and better than the re-implementation of 12 methods by Pan, Tasgetiren & Liang (2008), Tseng & Lin (2009), Tseng & Lin (2010), Zhang, Li & Wang (2009), Dong, Huang & Chen (2009), Jarboui, Eddaly & Siarry (2009), Xu, Xu & Gu (2011), Dubois-Lacoste, López-Ibáñez & Stützle (2011), Tasgetiren et al. (2011). The table reports the average of the relative percentage deviation from the best known values reported by Pan & Ruiz (2012). Each line of the table reports the average of 10 replications and 10 instances for our ILS, and of 5 replications and 10 instances for PR.

The machine of Pan & Ruiz (2012) is about 10% faster than our machine, but the average relative deviations of our algorithm in $30nm$ ms is still better than that of PR in $60nm$ ms. Indeed, in 89 of the 120 instances the average total completion time of

Table 5.3: Comparison to the best heuristic reported by Pan & Ruiz (2012).

Size	Our ILS		PR	
n×m	$30nm$	$60nm$	$30nm$	$60nm$
20×5	0.00	0.00	0.00	0.00
20×10	0.00	0.00	0.00	0.00
20×20	0.00	0.00	0.00	0.00
50×5	0.23	0.17	0.51	0.46
50×10	0.33	0.27	0.75	0.70
50×20	0.41	0.36	0.75	0.67
100×5	0.61	0.52	1.03	0.91
100×10	0.83	0.69	1.43	1.23
100×20	0.98	0.88	1.49	1.35
200×10	0.67	0.57	1.08	0.93
200×20	0.60	0.41	1.00	0.82
500×20	0.36	0.30	0.52	0.45
Averages	0.42	0.35	0.71	0.63

Table 5.4: New upper bounds on the total completion time for permutation schedules.

Instance	C_{sum}	Instance	C_{sum}
ta099	1025946	ta115	6728404
ta103	1268383	ta118	6771654
ta109	1234115	ta119	6710587

our algorithm is less than that of PR obtained in about the double of the time. Our algorithm found 6 new upper bounds on the total completion time, which are reported in Table 5.4.

We also compare our results to the multi-restart iterated local search (MRSILS) of Dong et al. (2013). They report a slightly better average relative deviation compared to the heuristics of Tasgetiren et al. (2011) and Pan, Tasgetiren & Liang (2008), that are also dominated by PR of Pan & Ruiz (2012). Dong et al. (2013) report the best values over 10 replications found in their experiments for instances with 50 and 100 jobs for a time limit of $400nm\,\mathrm{ms}$. We compare these values to the best values of our ILS for time limits of $30nm\,\mathrm{ms}$ and $60nm\,\mathrm{ms}$ over 10 replications in Table 5.5. Each line of the table reports the average for 10 instances of the best value of each instance. The machine of Dong et al. (2013) is about a factor of 3 slower than ours, but their time limit corresponds to about $130nm\,\mathrm{ms}$ on our hardware. After taking into account that factor, our method produces comparable results in about a quarter of the time of MRSILS, and consistently better results in half of the time.

Table 5.5: Comparison to the multi-restart ILS reported by Dong et al. (2013).

Size	This paper		MRSILS
n×m	$30nm$	$60nm$	$400nm^*$
50×5	0.10	0.07	0.09
50×10	0.13	0.09	0.17
50×20	0.18	0.17	0.21
100×5	0.44	0.39	0.45
100×10	0.58	0.41	0.53
100×20	0.61	0.59	0.55
Averages	0.34	0.29	0.34

*: about $130nm$ ms on our hardware.

We cannot directly compare to the recent particle swarm optimization heuristic of Zhang & Wu (2014), since neither the total completion times nor the upper bounds used to compute the relative deviations are available. They report an average relative deviation about 0.5% less than that of the particle swarm optimization of Tasgetiren et al. (2007) in instances with up to 100 jobs. Pan, Tasgetiren & Liang (2008) report an improvement of about 0.75% over the latter method for their heuristic, which in turn is dominated by PR of Pan & Ruiz (2012). Therefore, PR very likely is comparable to or dominates the particle swarm optimization of Zhang & Wu (2014).

In summary, the results show that our iterated local search is competitive with the currently best methods, and thus can serve as a fair baseline in the comparison to non-permutation schedules.

5.4.4 Quality of non-permutation schedules

In our second experiment we compare the quality of permutation and non-permutation schedules. Table 5.6 reports average relative deviations for all 12 instance groups for the ILS for permutation schedules with a time limit of $60nm\,\mathrm{ms}$, and for the IGA for non-permutation schedules with the same time limit. We also report the results for the IGA for a longer time limit of $30nm^2\,\mathrm{ms}$. The longer time limit has been chosen to study the convergence of the search, since the non-permutation problem is considerably more difficult to solve. All values are averages of 10 replications for 10 instances. Negative relative deviations indicate an improvement over the current best upper bound for permutation schedules.

The comparison shows that non-permutation schedules achieve a relative deviation which is in average about 0.44% less within the same time limit of $60nm\,\mathrm{ms}$. The

Table 5.6: Quality of permutation (PS) and non-permutation (NPS) schedules.

Size	PS	NPS	
n×m	$60nm$	$60nm$	$30nm^2$
20×5	0.00	-0.60	-0.62
20×10	0.00	-1.18	-1.28
20×20	0.00	-1.01	-1.19
50×5	0.17	0.08	0.06
50×10	0.27	-0.01	-0.10
50×20	0.36	-0.38	-0.48
100×5	0.52	0.50	0.43
100×10	0.69	0.47	0.43
100×20	0.88	0.39	0.22
200×10	0.57	0.45	0.38
200×20	0.41	0.06	-0.08
500×20	0.30	0.18	-0.01
Averages	0.35	-0.09	-0.19

improvement over permutation schedules is larger for smaller instances, and for instances with a larger number of machines.

Of the 120 instances, 114 non-permutation schedules have a smaller total completion time than the permutation schedules found with the same time limit of $60nm\,\mathrm{ms}$. The average total completion in 10 replications is better than the best upper bound for the permutation schedules in 49 instances, and overall 79 shorter non-permutation schedules have been found.

Non-permutation schedules improve about 1% over the presumably best possible (or optimal) permutation schedules for the instances with 20 jobs.

The longer time limit of $30nm^2\,\mathrm{ms}$ improves mostly the instances with a large number of machines, which are harder to optimize, and results in an overall improvement of 0.54%, that is 0.1% more. Of the 120 instances, 66 non-permutation schedules are shorter in average, and 88 instances permit a shorter schedule than the best known permutation schedule. Non-permutation schedules improve about 1.2% over the presumably best possible (or optimal) permutation schedules for the instances with 20 jobs.

In summary, non-permutation schedules notably improve over permutation schedules, and given the current quality of heuristics for permutation schedules, additional optimization time is better invested into finding non-permutation schedules.

5.4.5 Job reordering and buffer sizes

Next, we investigated the amount of job operations that change their processing order in the non-permutation schedules from our second experiment. To do this, we defined the job reordering index of an schedule $s = (\pi_1, \ldots, \pi_m)$ as the number of job inversions between adjacent machines normalized by the number of jobs and the number of adjacent machine pairs. This is defined mathematically as

$$\text{JRI}(s) = \frac{\sum_{i=1}^{m-1} \tau(\pi_i, \pi_{i+1})}{n(m-1)},$$

where we used the Kendall's τ distance between two permutations π and σ, that is defined as the number of element pairs that are in different order in the two permutations, or

$$\tau(\pi, \sigma) = |\{(i,j) : \pi^{-1}(i) < \pi^{-1}(j) \land \sigma^{-1}(i) > \sigma^{-1}(j)\}|,$$

where $i, j \in [n]$, and π^{-1} and σ^{-1} are the inverse permutations.

We observed that the average JRI is 3.3% and the maximum JRI is 6% among 2400 non-permutation schedules, with the exception of the schedules of the instance ta003 that present JRI between 6% and 9%. We also counted the number of inversions that each job has on each schedule, and we observed that 83.7% of the jobs change their position only once in a schedule, 12.5% changes twice, 3.2% changes three times, and the rest (0.6%) changes between four and sixteen times mainly in the instances of 200 jobs or more. This confirms the hypothesis that strategic operation reordering improves the total completion time in non-permutation flow shop schedules. Another observation is that 30% of the job reordering take place between the second and the third machine, and the 70% is equally disperse over the third machine. This may be taken into account for future research.

Finally, Table 5.7 compares the buffer requirements of permutation and non-permutation schedules. It reports the average buffer size $\bar{b}$ and the maximum buffer size B for all permutation and non-permutation schedules found in the first two experiments. The buffer size of a single schedule is defined as the largest buffer necessary between any two machines. As in the previous experiments, the values are averages over 10 replications. The results show that the average and maximum buffer sizes depend mainly on the number of jobs and machines of the instance, and do not vary significantly with the quality of the heuristic solution. In particular, non-permutation schedules do not lead to larger buffer sizes, and therefore can be implemented in practice without technological changes.

Table 5.7: Average ($\bar{b}$) and maximum (B) buffer sizes of permutation (PS) and non-permutation (NPS) schedules.

Size	PS				NPS			
	$30nm$		$60nm$		$60nm$		$30nm^2$	
n×m	$\bar{b}$	B	$\bar{b}$	B	$\bar{b}$	B	$\bar{b}$	B
20×5	2.0	2.0	2.0	2.0	2.0	2.1	2.0	2.0
20×10	2.2	2.2	2.2	2.2	2.2	2.3	2.1	2.4
20×20	2.7	2.7	2.7	2.7	2.4	2.8	2.4	2.9
50×5	2.8	3.3	2.7	3.2	2.7	3.4	2.8	3.3
50×10	3.1	3.8	3.1	3.9	3.0	3.5	3.1	3.8
50×20	3.9	5.2	4.0	4.9	3.8	4.6	3.8	4.8
100×5	3.9	4.6	3.9	4.6	3.9	4.7	3.7	4.4
100×10	4.4	5.6	4.3	5.0	4.3	5.2	4.4	5.3
100×20	5.7	6.9	5.6	6.5	5.5	6.3	5.6	6.8
200×10	5.2	6.3	5.3	6.2	5.2	6.4	5.0	6.1
200×20	6.5	7.9	6.4	7.6	6.4	7.7	6.4	7.7
500×20	9.3	10.4	9.1	10.1	9.2	10.3	8.9	9.9
Averages	4.3	5.1	4.3	4.9	4.2	4.9	4.2	5.0

5.5 Concluding remarks

In this chapter we have studied heuristics based on iterated local search for minimizing the total completion time in permutation and non-permutation flow shop schedules. We have proposed an ILS for permutation schedules and an IGA for non-permutation schedules. ILS produces permutation schedules that are about 0.3% better than state-of-the-art methods, and it found 6 new upper bounds for the instances of Taillard (1993). IGA produces non-permutation schedules that are 0.44% better than the permutation schedules produced by ILS in the same time, and improves a little more with a larger time limit. IGA found 88 non-permutation schedules of less total completion time than the current best upper bounds for permutation schedules, and they improve up to 1.2% over the presumably best possible (or optimal) permutation schedules for the instances with 20 jobs. Thus, additional computation time after $30nm\,\mathrm{ms}$ is better invested in finding non-permutation schedules.

Our main conclusion is that non-permutation schedules can be found within the same time than permutation schedules, and they are even better than the presumably best possible (or optimal) permutation schedules. We also confirmed that strategic operation reordering improves the total completion time in non-permutation flow shop schedules, and that non-permutation flow shop schedules can be implemented in practice without technological changes.

6 A FIRST APPROACH TO MINIMIZE THE MAKESPAN IN THE NON-PERMUTATION FLOW SHOP SCHEDULING PROBLEM

Section 2.2.1 has defined the permutation and non-permutation FSSP formally, Chapter 4 has explored the theoretical differences between the permutation and non-permutation FSSP, and Chapter 5 has studied the practical differences between the permutation and the non-permutation FSSP with the total completion time minimization criterion.

In this chapter, we propose a constructive heuristic and an iterated local search heuristic for the non-permutation flow shop scheduling problem with makespan minimization criterion. Both heuristics are based on the observation that optimal non-permutation schedules often result from a few local job inversions in a permutation structure. These heuristics are described in Section 6.1. The computational experiments in Section 6.2 compare our heuristics to the best heuristics in the literature for finding non-permutation and permutation flow shop schedules, and evaluate the reduction in makespan and buffer size that can be achieved by non-permutation schedules. Finally, we conclude in Section 9.4. This chapter corresponds to the publication of Benavides & Ritt (2016). The detailed computational results of this chapter are available online at <http://www.inf.ufrgs.br/algopt/npfs>.

6.1 Heuristics for the non-permutation FSSP

6.1.1 A constructive heuristic for the non-permutation FSSP

The best constructive heuristic for the permutation FSSP is the NEH heuristic proposed by Nawaz, Enscore & Ham (1983) with the acceleration technique of Taillard (1990). This method is described in Section 3.1.1. It builds a permutation schedule by repeatedly inserting the jobs in some order into a partial permutation schedule at

Algorithm 6.1 Constructive heuristic NFS for the non-permutation FSSP.

```
Input: The fraction $p$ of insertions without job passing, a job order $\rho$.
Output: A non-permutation schedule $S = \{\pi_1, \pi_2, \ldots, \pi_m\}$.
 1: function NFS( )
 2:     Let $S$ be the empty schedule
 3:     for $j = 1, \ldots, \lfloor pn \rfloor$ do
 4:         Insert job $\rho_j$ at the optimal position $k \in [j]$ into $S$
 5:     end for
 6:     for $j = \lfloor pn \rfloor + 1, \ldots, n$ do
 7:         for all positions $k \in [j]$ do
 8:             Evaluate insertion of $\rho_j$ at $k$ with anticipation after machine $2, \ldots, m-1$
 9:             Evaluate insertion of $\rho_j$ at $k$ with delay after machine $2, \ldots, m-1$
10:             Evaluate insertion of $\rho_j$ at $k$
11:         end for
12:         Apply the best insertion of $\rho_j$ into $S$
13:     end for
14:     return $S$
15: end function
```

the position which produces the smallest makespan. We extend this method to create non-permutation schedules by allowing job passing when inserting a job.

Algorithm 6.1 shows the proposed constructive heuristic NFS. For each job to be inserted, and for each candidate position, we evaluate insertions without job passing, with anticipation, or with delay after some intermediate machine. After the evaluations, the job is inserted at the position and with the anticipation or delay that minimizes the makespan. We explained the insertion of a job with anticipation and delay in Section 5.1.1. If there are several optimal positions, ties can be broken by any of the rules proposed in the literature, e.g. KK2 of Kalczynski & Kamburowski (2009). Any remaining ties are broken by giving preference to insertions with earliest anticipation, followed by an insertion without job passing, followed by insertions with earliest delay.

We have observed that allowing job passing is more beneficial for jobs inserted later. For this reason, we allow to specify a fraction p of the jobs, which will be inserted first without job passing.

The speedup technique of Taillard (1990) finds the best insertion of a job into a permutation schedule in time $O(nm)$. It cannot be applied directly to the non-permutation case, because some head and tail values become invalid during the evaluation. Then, the evaluation of one insertion with anticipation or delay is evaluated in time $O(n^2m)$. Additionally, we have to test nm instead of n possible insertions. Thus, the proposed constructive heuristic NFS has time complexity $O(n^3m^2)$.

6.1.2 Inserting jobs efficiently into non-permutation schedules

In this section we propose a speedup technique to reduce the complexity of the constructive heuristic NFS. We first propose a technique for the fast calculation of the invalid heads and tails to insert jobs without job passing into non-permutation schedules, and we finally generalize this technique to arbitrary insertions. In the following, if not mentioned otherwise, we use index $i \in [m]$ for machines, index $j \in [n]$ for jobs, and index $k \in [n]$ for positions of jobs.

Consider an insertion without job passing at position k into a non-permutation schedule. This will increase the completion times of jobs at positions $k' \geq k$ in the current schedule. Different from the permutation case, such a job may occur at a position $k'' < k$ on later machines. In this case, the head $e_{ik''}$ is invalid, and Taillard's optimization cannot be applied. Similarly, the tail of a job at position $k' \geq k$ on machine i is invalid, if it occurs at a position $k'' < k$ on a later machine. In both cases, a job which occurs after the inserted job on some machine i, but is anticipated to occur before it on some later machine $i' > i$, has an invalid tail on i and an invalid head on i'. Moreover, an invalid head or tail on some machine may lead to more invalid heads or tails on later or earlier machines. Figure 6.1 gives two examples of an insertion without job passing which leads to a different number of invalidated heads.

The set of invalid heads and tails can be found efficiently as follows. Let $\pi_1, \ldots, \pi_m$ be the current non-permutation schedule with n jobs. Thus, the kth job on machine i is π_{ik}, and we will write π_{ij}^{-1} for the position of job J_j on machine i. For a set of jobs $T \subseteq [n]$, let $l_i(T) = \min_{j \in T} \pi_{ij}^{-1}$ be the smallest index of some job in T on machine i, and $L_i(T) = \max_{j \in T} \pi_{ij}^{-1}$ the largest index of some job in T on machine i. Furthermore let $S(\pi, k) = \{\pi_k, \ldots, \pi_n\}$ be the suffix of permutation π starting at position k, and $P(\pi, k) = \{\pi_1, \ldots, \pi_{k-1}\}$ be the prefix of permutation π ending at position $k - 1$.

Pos.	1	2	3	4
M_1	4	3	2	**1**
M_2	4	3	2	**1**
M_3	4	**1**	**3**	**2**
M_4	**3**	**1**	**4**	**2**
M_5	**3**	**1**	**4**	**2**

Pos.	1	2	3	4
M_1	4	3	2	**1**
M_2	4	3	2	**1**
M_3	2	**1**	**3**	**4**
M_4	2	**1**	**3**	**4**
M_5	2	**1**	**3**	**4**

Figure 6.1: Two examples of an insertion without job passing at position 4 into non-permutation schedules. Permutations on each machine are shown with the jobs whose heads are invalidated in bold.

Let $l_{ik} = l_{i+1}(S(\pi_i, k))$ be the smallest index of some job in $S(\pi_i, k)$ on machine $i+1$. Values l_{ik} can be computed in time $O(nm)$ by

$$l_{i,n+1} = \infty; \qquad l_{ik} = \min\{\pi^{-1}_{i+1,\pi_{ik}}, l_{i,k+1}\}.$$

Then the first invalid head position μ_i on machine i is given by

$$\mu_1 = k; \qquad \mu_{i+1} = l_{i,\mu_i}. \tag{6.1}$$

Similarly, let $L_{ik} = L_{i-1}(P(\pi_i, k))$ be the largest index of some job in $P(\pi_i, k)$ on machine $i-1$. Values L_{ik} can be computed in time $O(nm)$ by

$$L_{i1} = -\infty; \qquad L_{ik} = \max\{\pi^{-1}_{i-1,\pi_{i,k-1}}, L_{i,k-1}\}.$$

Then the first valid tail position ν_i on machine i is given by

$$\nu_m = k; \qquad \nu_{i-1} = L_{i,\nu_i} + 1. \tag{6.2}$$

Given the information on the position of the first invalid head and the first valid tail, we can compute the makespan of an insertion without job passing of job J_j at position k as shown in Algorithm 6.2. For each machine i, the algorithm finds the first valid head e_{i,μ_i-1}, and then computes the completion times $C_{i,\pi_i(k')}$ for all subsequent jobs $\pi_i(k')$ up to the first valid tail q_{i,ν_i}, including the newly inserted job. This computation can also be understood as computing an extended relative head r_i.

Algorithm 6.2 Insertion into a non-permutation schedule.

Input: A non-permutation schedule $(\pi_1, \ldots, \pi_m)$, a job J_j and a position k for insertion.
Output: The new makespan after the insertion without job passing of J_j at position k.

```
 1: Let $C_{0j} := 0$ for all $j$.
 2: for $i \in [m]$ do
 3:     $r_i := e_{i,\mu_i-1}$
 4:     for $k' \in [\mu_i, k-1]$ do
 5:         $r_i := C_{i,\pi_i(k')} := \max\{r_i, C_{i-1,\pi_i(k')}\} + p_{i,\pi_i(k')}$
 6:     end for
 7:     $r_i := C_{ij} := \max\{r_i, C_{i-1,j}\} + p_{ij}$
 8:     for $k' \in [k, \nu_i - 1]$ do
 9:         $r_i := C_{i,\pi_i(k')} := \max\{r_i, C_{i-1,\pi_i(k')}\} + p_{i,\pi(k')}$
10:     end for
11: end for
12: $C_{\max} := \max_{i\in[m]} r_i + q_{i,\nu_i}$
```

Observe that for permutation schedules, we have $\mu_i = \nu_i = k$ and the procedure reduces to Taillard's insertion procedure. In general let $W = \max_{i\in[m]} \nu_i - \mu_i$ be the "width" of the invalid heads and tails. Then the complexity of an insertion is $O(mW)$, and all $\Theta(nm)$ insertions can be computed in time $O(nm^2W)$, since the values l_{ik} and L_{ik} have to be computed only once.

Finally, consider insertions into non-permutation schedules with anticipation or delay. More generally, when inserting a new job j at position k_i on machine i, we obtain the first invalid head position and the first valid tail position by considering the insertion position k_i in eqs. (6.1) and (6.2)

$$\mu_1 = k_1, \qquad \mu_{i+1} = \min\{l_{i,\mu_i}, k_{i+1}\}; \tag{6.3}$$

$$\nu_m = k_m, \qquad \nu_{i-1} = \max\{L_{i,\nu_i} + 1, k_{i-1}\}. \tag{6.4}$$

A NEH-like insertion procedure now has overall complexity $O(n^2m^2W)$, i.e. we have a speedup of n/W over the naïve insertion into non-permutation schedules, and a slowdown of mW compared to the permutation procedure. In the worst case $W = n$, but since the number of inversions of a non-permutation schedule is usually limited, W tends to be a small constant. Empirically, for the instances used in the experiments in Section 6.2, W correlates with the buffer size, and we find $W \approx 10$, in average.

6.1.3 A local search heuristic for the non-permutation FSSP

To reduce the makespan of a job shop schedule, the order of at least two operations in some block of the critical path must be inverted. The neighbourhood of Nowicki & Smutnicki (1996) limits the inversions to the first two operations of a block, except the first block, and the last two operations of a block, except the last block, because only those inversions may reduce the makespan directly.

Our local search extends the neighbourhood proposed by Nowicki & Smutnicki (1996) to permit local job passing in a flow shop schedule as follows. For each pair of candidate jobs J_1 and J_2 for an inversion on machine M_i according to the above neighbourhood, we analyze three kinds of neighbours obtained by inverting the order of the jobs J_1 and J_2 on machines $M_1, \ldots, M_i'$ for all $i' \geq i$, on machines $M_i'', \ldots, M_m$ for all $i'' \leq i$, and only on machines M_i. Note that this includes the complete inversion of jobs J_1 and J_2 on all machines $M_1, \ldots, M_m$. A neighbour is only considered if J_1 and J_2 are consecutive on the corresponding machines. The local search uses a best improvement strategy, i.e., it repeatedly passes to the best neighbour, until the solution is a local minimum. To speed up the evaluation of each neighbour, we estimate its makespan using the technique of Taillard (1990), and evaluate it completely only if the estimation is promising. If there are several critical paths, we choose the bottommost in the Gantt chart, i.e., when building the critical path from the last operation backwards and the current operation has several preceding critical operations, we choose the one on the same machine.

6.1.4 An iterated greedy algorithm for the non-permutation FSSP

The main principle of an iterated greedy algorithm is to repeatedly remove some random elements from the current solution, and insert them again into the partial solution using a greedy algorithm. An iterated greedy algorithm is similar to an iterated local search, but performs the perturbation by removing and reconstructing some elements.

The proposed iterated greedy algorithm for the FSSP starts with an initial solution which is the best solution obtained by applying the constructive heuristic NFS of Section 6.1.1 to the standard and reverse instance. It then repeatedly selects d out of the n jobs, removes them from the current schedule, and reconstructs a new, complete schedule, by reinserting them into the partial schedule. The reinsertion uses the same greedy strategy as the constructive heuristic NFS, inserting the job into the position which produces the smallest makespan, considering all insertions with anticipation, without job passing, and with delay.

After a new complete schedule has been constructed, we improve it with a local search, described below. The solution obtained after local search is accepted according to a Metropolis criterion, i.e., either if it improves the current solution or with probability exp$(-\Delta/T)$, for an increase of the makespan by $\Delta = C_{\max}(s') - C_{\max}(s)$. The temperature is set to a fixed value of $T = \alpha\overline{p}/10$ during the execution, for an average processing time of $\overline{p} = \sum_{j\in[n]}\sum_{i\in[m]} p_{ij}/nm$ and a parameter α.

6.2 Computational Results

6.2.1 Test instances and experimental methodology

We tested our algorithms on two sets of instances: 120 instances proposed by Taillard (1993) and 40 instances proposed by Demirkol, Mehta & Uzsoy (1998). and described in Section 3.4, which are the standard benchmark in the literature.

In the experiments we compare the solution quality of different algorithms within a fixed time limit. Heuristics for the PFSSP are usually compared with a time limit that is a small multiple of $10nm$ ms. Since the FSSP is considerably more difficult to solve, we adopt a standard time limit of $30nm^2$ ms for our search heuristic. This time limit is adjusted when necessary to make the computation times comparable to results from the literature.

We measure solution quality by the relative percentage deviation from the best known value $C^*_{\max}$ defined as RPD $= 100 \times (C_{\max} - C^*_{\max})/C^*_{\max}$.

Our algorithms have been implemented in implemented in C++, compiled with GNU C++ Compiler version 4.7.3 with maximum optimization, and run on a PC with an AMD FX-8150 processor running at 3.6 GHz, and with 32 GB of main memory, using only one core in each execution.

6.2.2 Parameter setting

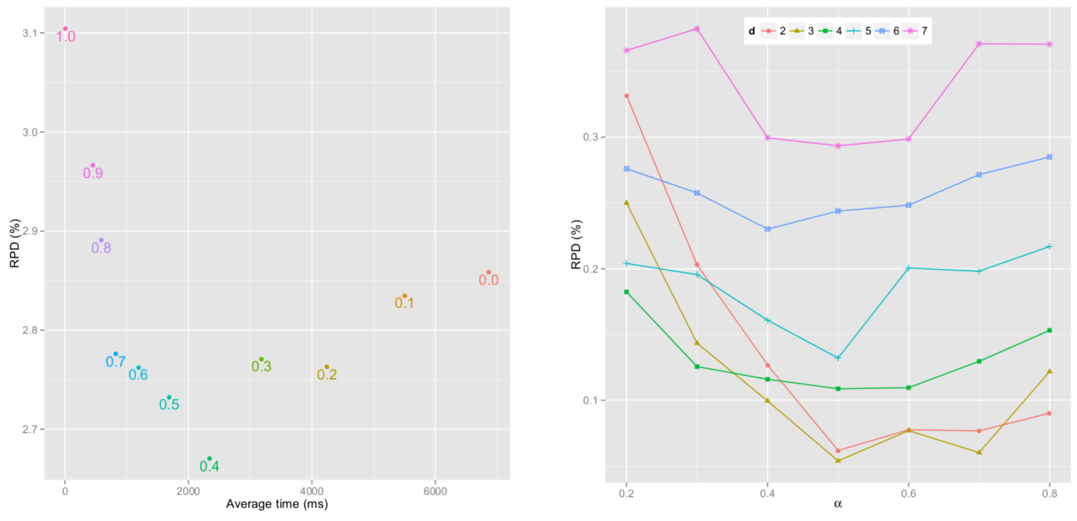
As explained before, it can be advantageous to insert a percentage p of the jobs regularly to produce a partial permutation schedule, and then insert the remaining jobs allowing job passing to produce a non-permutation schedule. We have tested our constructive heuristic using the insertion order and tie-breaking rule KK2 of Kalczynski & Kamburowski (2009) for $p \in \{0.0, 0.1, \ldots, 0.9, 1.0\}$ on the 120 instances of Taillard (1993). Observe that $p = 1.0$ corresponds to the NEHKK2 heuristic of Kalczynski & Kamburowski (2009) for the PFSSP. The results of these tests are summarized in Figure 6.2a, which plots the average execution time versus the average RPD from the best known values for the PFSSP for the different values of p. Based on these results, we chose $p = 0.4$ which produces the best values in an acceptable time.

To calibrate the IGA algorithm we first compared different combinations of the algorithmic components, and then tuned the parameters.

Table 6.1 shows the average RPD over the 120 instances of Taillard (1993) for four variants: the IGA starting from the identical permutation ("IGA"), the IGA starting from the solution obtained by our constructive heuristic ("NFS+IGA"), the IGA starting from the identical permutation, applying local search in each iteration ("IGA(LS)"), and the IGA starting from the solution obtained by our constructive heuristic and

Table 6.1: Comparison of variants of the proposed iterated greedy algorithm on the instances of Taillard (1993).

Variant	Avg.
IGA	0.33
NFS+IGA	0.31
IGA(LS)	0.22
NFS+IGA(LS)	0.19



(a) (b)

Figure 6.2: Calibration of (a) the percentage p of the jobs inserted without job passing and (b) of the parameters d and α.

applying local search in each iteration ("FS+IGA(LS)"). We can see that all algorithmic components contribute to the performance of the algorithm. The contribution of the initial solution, however, is, as expected, rather small, while adding the local search leads to an average RPD which is about 0.1% lower. The experiments which follow have been done with the best variant NFS+IGA(LS).

To calibrate the IGA we ran a full factorial experiment on all combinations of values $d \in \{2,\dots,7\}$ and $\alpha \in \{0.2,\dots,0.8\}$ with five replications for each parameter setting on a random instance of each of the 12 instance groups of Taillard (1993). The parameter ranges have been chosen based on the optimal values of $d = 4$ and $\alpha = 0.5$ determined by Ruiz & Stützle (2007) for their IGA for the PFSSP. The results are summarized in Figure 6.2b. Our results confirm the values found by Ruiz & Stützle (2007), with a slight advantage for d = 3 in the non-permutation case. We therefore set $d = 3$ and $\alpha = 0.5$ in our experiments.

6.2.3 Experiment 1: Effectiveness of the constructive heuristic

Our first experiment evaluates the effectiveness of the constructive heuristic. The currently best constructive heuristic for the FSSP has been proposed by Koulamas (1998). Ruiz & Maroto (2005) have compared 18 constructive heuristics for the permutation and non-permutation FSSP, including the heuristic of Koulamas (1998), and found the heuristic NEH of Nawaz, Enscore & Ham (1983) to perform best. Fernandez-Viagas & Framinan (2014) recently have proposed a tie-breaking rule which obtains (statistically) significantly better results than previous methods. In

Table 6.2: Comparison of constructive heuristics on the instances of Taillard (1993).

Grp.	Size	NEH	KK2	KM	FF	NFS
ta01	20×5	2.49	2.48	7.68	2.29	2.02
ta02	20×10	4.17	4.17	11.82	4.15	3.03
ta03	20×20	3.36	3.57	11.89	3.31	3.21
ta04	50×5	0.58	0.44	4.03	0.92	0.28
ta05	50×10	4.97	5.38	12.13	5.15	4.55
ta06	50×20	5.85	6.22	14.93	6.21	5.43
ta07	100×5	0.38	0.25	3.12	0.38	0.21
ta08	100×10	2.02	2.17	7.50	2.18	1.53
ta09	100×20	5.20	5.32	14.04	5.02	4.96
ta10	200×10	1.15	1.13	5.09	0.98	1.01
ta11	200×20	4.20	4.22	11.60	4.04	3.92
ta12	500×20	1.98	1.90	6.82	1.78	1.89
Averages		3.03	3.10	9.22	3.03	2.67

Table 6.2 we compare the solution quality of the heuristic of Koulamas (1998) (column "KM"), as reported by Ruiz & Maroto (2005), the heuristic of Fernandez-Viagas & Framinan (2014) (column "FF"), to our implementation of NEH and NEHKK2 (KALCZYNSKI; KAMBUROWSKI, 2009) (columns "NEH" and "KK2"), and our heuristic (column "NFS"). For the last three we report the best value from an application to the standard and the reverse instance.

The results show that NFS can improve the permutation schedules of the best constructive methods for the PFSSP by about 0.3%. The makespans of NFS are the shortest in all twelve instance groups, except two (ta10 and ta12) when compared to heuristic FF. Given the difficulty of finding better constructive heuristics for the PFSSP, this can be considered a reasonable improvement. Note, however, that the improvement over heuristic FF could be slightly smaller, since it has not been applied to the reverse instance. Our algorithm has a worst case time complexity of $O(n^3m^2)$ and thus it is no surprise that its average execution time of about 2.2 s is much slower than that of the other algorithms which run in about 10 ms. However, the execution time is still reasonably short.

6.2.4 Experiment 2: Effectiveness of the iterated greedy algorithm

In this section we compare our method to the state-of-the-art heuristics for the FSSP shown in Table 6.3. For each heuristic, the table provides the set of test instances and the computing environment used in the computational experiments.

Table 6.3: Overview of recent heuristic approaches for the FSSP.

Reference.	Instances	Environment
Lin & Ying (2009)	de	Pentium 4, 1.4 GHz
Yagmahan & Yenisey (2010)	ta001–ta028	Pentium M760, 2 GHz
Rossi & Lanzetta (2013b)	ta001–ta028, de	Pentium 4, 3 GHz
This paper	ta, de	AMD FX-8150, 3.6 GHz

Table 6.4: Comparison to Lin & Ying (2009) and Rossi & Lanzetta (2013b) on the instances of Demirkol, Mehta & Uzsoy (1998).

				Time LY/6			Time $30nm^2$ ms		
Grp.	Size	LY	RL	Min.	Avg.	Max.	Min.	Avg.	Max.
de01	20×15	0.00	4.45	-1.91	-1.52	-1.08	-2.11	-1.99	-1.82
de02	20×20	0.00	5.11	-2.54	-2.23	-1.87	-2.72	-2.66	-2.60
de03	30×15	0.00	7.73	-1.96	-1.45	-0.83	-2.18	-2.03	-1.83
de04	30×20	0.00	7.56	-2.76	-2.37	-1.88	-3.05	-2.88	-2.68
de05	40×15	0.00	8.59	-1.68	-1.26	-1.00	-2.10	-1.76	-1.50
de06	40×20	0.00	10.62	-2.00	-1.55	-1.12	-2.44	-2.19	-1.87
de07	50×15	0.00	9.27	-1.17	-0.71	-0.34	-1.69	-1.42	-1.08
de08	50×20	0.00	10.62	-1.83	-1.45	-0.91	-2.51	-2.17	-1.94
Averages		0.00	7.99	-1.98	-1.57	-1.13	-2.35	-2.14	-1.92

We first compare to the two best heuristics of Lin & Ying (2009) and Rossi & Lanzetta (2013b) which report results on the instances of Demirkol, Mehta & Uzsoy (1998). The machine of Lin & Ying (2009) is about 6 times slower than our machine, that of Rossi & Lanzetta (2013b) is about 3 times slower than our machine. The running times of Rossi & Lanzetta (2013b) (reported in Rossi & Lanzetta (2013a)) are, after normalization, a factor of about 1.7 larger than those of Lin & Ying (2009). For this reason we report results for our heuristic for two different time limits: the time limit as reported by Lin & Ying (2009), corrected by a factor of 6, which corresponds to a maximum execution time of 45 s on the largest instances, and our standard time limit of $30nm^2$ ms.

Table 6.4 reports for each group of instances the average RPD for Lin & Ying (2009) (column "LY") and Rossi & Lanzetta (2013b) (column "RL"), and the minimum, average, and maximum RPD of our method for both time limits. The values from Lin & Ying (2009) are the best of five replications, the values from Rossi & Lanzetta (2013b) are the best of ten replications, and our values are the best, average, and worst of ten replications.

Our method is able to find new best values for all 40 instances, with an average RPD of 1.57% below the current best known values, which have been obtained by the best of five runs of the method of Lin & Ying (2009). Neither the best nor the average of our values is directly comparable to Lin & Ying (2009), because of the different number of replications. However, when we select the five worst of our ten runs, the average makespan of all 40 instances is still shorter than the best known values, with an average RPD of 1.36% below their values. Compared to Rossi & Lanzetta (2013b), even the worst of our ten runs is about 8% better than their best result in ten runs.

We next compare to the heuristics of Yagmahan & Yenisey (2010) and Rossi & Lanzetta (2013b) which report results on the first 28 instances of Taillard (1993). We base our comparison on results reported by Rossi & Lanzetta (2013b) and report results for two time limits: a time limit of $30nm$ ms and our standard time limit of $30nm^2$ ms. The first time limit has been chosen such that the computation time is, after correction for different machine speeds, safely less than that of the other approaches, based on the values reported in Rossi & Lanzetta (2013a) and Yagmahan & Yenisey (2010).

Table 6.5 reports for each instance the best known value (column "BKV"), the RPD of the best in ten runs for Yagmahan & Yenisey (2010) (column "YY"), the best and average RPD over ten runs of Rossi & Lanzetta (2013b) (columns "RL"), and the minimum, average and maximum RPD of our method for the above time limits. We report in Table 6.6 the results for all 120 instances for our method, in average over each group of ten instances of the same size.

Our method finds 32 new best values, 13 of them in the first 28 instances. The worst makespan in ten runs obtained with the shorter time limit is in average about 5% better than the best in ten runs of the methods of Yagmahan & Yenisey (2010) and Rossi & Lanzetta (2013b). On all 120 instances we obtain an overall RPD of 0.19% for the shorter time limit, and -0.10% for the longer time limit.

6.2.5 Experiment 3: Comparison to permutation schedules

In our final experiment we want to assess the makespan obtainable by non-permutation schedules in comparison to permutation schedules and to compare the performance of our heuristic to that of the best heuristics for the PFSSP. We chose four state-of-the-art methods for this comparison: the hybrid algorithm NEGA$_{\text{VNS}}$ of Zobolas, Tarantilis & Ioannou (2009), the particle swarm optimization PSO of Tasgetiren et al. (2007), the hybrid genetic algorithm HGA RMA of Ruiz, Maroto & Alcaraz (2006), and the iterated greedy method of Fernandez-Viagas & Framinan (2014).

Table 6.5: Comparison to Yagmahan & Yenisey (2010) and Rossi & Lanzetta (2013b) on 28 instances of Taillard (1993).

				RL		Time $30nm$ ms			Time $30nm^2$ ms		
Name	Size	BKV	YY	Min.	Avg.	Min.	Avg.	Max.	Min.	Avg.	Max.
ta001	20×5	1278	1.49	0.94	1.18	0.00	0.00	0.00	0.00	0.00	0.00
ta002	20×5	1358	1.84	2.28	2.28	0.00	0.00	0.00	0.00	0.00	0.00
ta003	20×5	1073	12.12	2.52	3.68	0.00	0.00	0.00	0.00	0.00	0.00
ta004	20×5	1292	6.58	4.02	4.66	0.08	0.08	0.08	0.08	0.08	0.08
ta005	20×5	1231	6.50	1.54	2.25	0.32	0.32	0.32	0.00	0.19	0.32
ta006	20×5	1193	4.36	2.01	2.62	0.00	0.00	0.00	0.00	0.00	0.00
ta007	20×5	1234	5.59	1.94	2.07	0.16	0.38	0.41	0.16	0.36	0.41
ta008	20×5	1199	5.50	3.00	3.63	0.00	0.00	0.00	0.00	0.00	0.00
ta009	20×5	1210	7.69	3.97	5.40	0.00	0.00	0.00	0.00	0.00	0.00
ta010	20×5	1103	6.89	2.18	3.85	0.00	0.00	0.00	0.00	0.00	0.00
ta011	20×10	1560	7.76	8.53	10.87	-0.38	0.15	0.71	-0.38	-0.24	0.00
ta012	20×10	1644	6.39	8.58	9.46	0.00	0.09	0.30	0.00	0.01	0.06
ta013	20×10	1486	4.58	6.53	7.44	-1.08	-0.79	-0.13	-1.21	-1.20	-1.08
ta014	20×10	1368	8.92	6.14	8.06	-0.22	0.05	0.58	-0.37	-0.22	-0.15
ta015	20×10	1413	2.97	7.29	8.05	-1.27	-0.86	-0.35	-1.42	-1.30	-1.27
ta016	20×10	1369	14.24	5.55	7.25	0.00	0.02	0.22	0.00	0.00	0.00
ta017	20×10	1428	11.34	6.72	8.16	0.21	0.29	0.56	-0.07	-0.04	0.21
ta018	20×10	1527	4.45	8.06	8.96	0.00	0.00	0.00	0.00	0.00	0.00
ta019	20×10	1586	6.49	4.60	6.00	0.00	0.09	0.38	0.00	0.00	0.00
ta020	20×10	1559	10.26	7.12	7.61	0.00	0.09	0.45	0.00	0.00	0.00
ta021	20×20	2293	5.89	4.49	5.32	-2.31	-2.03	-1.88	-2.35	-2.34	-2.31
ta022	20×20	2092	9.03	6.36	7.05	-1.82	-1.32	-0.76	-1.86	-1.69	-1.39
ta023	20×20	2313	8.73	5.75	6.55	-2.98	-2.46	-1.51	-3.33	-3.20	-3.03
ta024	20×20	2223	3.42	5.53	6.20	-2.02	-1.75	-1.44	-2.16	-2.09	-2.02
ta025	20×20	2291	7.94	6.46	7.40	-2.01	-1.62	-1.00	-2.23	-2.00	-1.88
ta026	20×20	2221	5.31	4.95	5.65	-2.66	-2.35	-1.89	-2.93	-2.74	-2.66
ta027	20×20	2267	4.90	7.10	8.25	-1.94	-1.77	-1.63	-2.16	-2.10	-2.07
ta028	20×20	2183	10.77	6.32	7.45	-1.37	-0.93	-0.27	-1.37	-1.33	-0.96
Averages			6.86	5.02	5.98	-0.69	-0.51	-0.25	-0.77	-0.71	-0.63

Table 6.6: Complete results on the instances of Taillard (1993).

	Time $30nm$ ms			Time $30nm^2$ ms		
Size	Min.	Avg.	Max.	Min.	Avg.	Max.
20×5	0.06	0.08	0.08	0.02	0.06	0.08
20×10	-0.27	-0.09	0.27	-0.34	-0.30	-0.22
20×20	-2.14	-1.73	-1.22	-2.30	-2.18	-1.98
50×5	0.00	0.01	0.03	0.00	0.00	0.00
50×10	0.26	0.52	0.80	0.14	0.23	0.38
50×20	-0.28	0.15	0.65	-1.04	-0.74	-0.46
100×5	0.00	0.02	0.04	0.00	0.01	0.02
100×10	0.22	0.37	0.56	0.06	0.17	0.29
100×20	0.73	1.03	1.43	0.21	0.42	0.60
200×10	0.06	0.20	0.31	0.01	0.07	0.14
200×20	0.84	1.14	1.43	0.61	0.74	0.89
500×20	0.50	0.64	0.80	0.31	0.38	0.47
Averages	0.00	0.19	0.43	-0.19	-0.10	0.02

Table 6.7: Comparison to state-of-the-art heuristics from the literature for permutation schedules on the instances of Taillard (1993).

						Time $30nm$ ms			Time $30nm^2$ ms		
Size	Gap	HGA	PSO	NGV	FF	Min.	Avg.	Max.	Min.	Avg.	Max.
20×5	0.00	0.04	0.03	0.00	0.03	-0.35	-0.33	-0.32	-0.38	-0.34	-0.32
20×10	0.00	0.05	0.02	0.01	0.06	-1.57	-1.39	-1.03	-1.64	-1.60	-1.52
20×20	0.00	0.04	0.05	0.02	0.06	-2.42	-2.00	-1.49	-2.57	-2.45	-2.25
50×5	0.00	0.01	0.00	0.00	0.01	-0.17	-0.16	-0.14	-0.17	-0.16	-0.16
50×10	0.00	0.62	0.57	0.82	0.44	0.12	0.38	0.66	-0.01	0.08	0.23
50×20	1.58	1.03	1.36	1.08	1.05	-0.28	0.15	0.65	-1.04	-0.74	-0.46
100×5	0.00	0.01	0.00	0.00	0.01	-0.14	-0.12	-0.10	-0.14	-0.13	-0.12
100×10	0.00	0.19	0.18	0.14	0.15	0.06	0.22	0.40	-0.09	0.01	0.13
100×20	0.83	1.30	1.45	1.40	1.12	0.73	1.03	1.43	0.21	0.42	0.60
200×10	0.00	0.06	0.18	0.16	0.09	-0.01	0.13	0.24	-0.05	0.00	0.07
200×20	0.19	1.25	1.35	1.25	1.05	0.84	1.14	1.43	0.61	0.74	0.89
500×20	0.02	0.45		0.71	0.45	0.50	0.64	0.80	0.31	0.38	0.47
Averages	0.22	0.42	0.47	0.47	0.38	-0.22	-0.03	0.21	-0.41	-0.32	-0.20

Heuristics for the PFSSP are usually compared on the instances of Taillard (1993), so we limit this experiment to these instances. Table 6.7 shows the summarized results for all twelve instance groups of different size (column "Size" in $n \times m$). Column "Gap" reports the relative deviation of the best known values from the best lower bounds as reported in Taillard (2004). We obtained the average RPD over 10 replications for NEGA$_{\text{VNS}}$ and PSO from the study of Zobolas, Tarantilis & Ioannou (2009) (columns "NGV", "PSO"), and the average RPD over 5 replications from Fernandez-Viagas & Framinan (2014). The study of Zobolas, Tarantilis & Ioannou (2009) has taken care that the computation time of the methods are comparable, and uses a time limit of $100nm$ ms on a Pentium 4 running at 2.4 MHz. Since our machine is about a factor 3 faster, we ran our method with a time limit of $30nm$ ms, and report the minimum, average and maximum RPD from the best known values in percent over 10 replications (columns "Min.", "Avg." and "Max."). For validation we have implemented the method HGA-RMA of Ruiz, Maroto & Alcaraz (2006) and ran 10 replications with the same time limit of $30nm$ ms in our environment, and report the averages (column "HGA"). A comparison with the results reported by Zobolas, Tarantilis & Ioannou (2009) shows that the above assumption on the relative machine speeds are sound. From the results of Fernandez-Viagas & Framinan (2014) we chose the best results for a time limit of $30nm$ ms, since their machine is about 25% faster than ours. We further report the relative deviations of the results for our standard time limit of $30nm^2$ ms.

We can see that our heuristic is competitive with current heuristics for the PFSSP. It obtains a slightly negative average RPD compared to about 0.4% to 0.5% for the PFSSP heuristics. It can improve in particular the smaller instances and obtains negative relative deviations in five groups in average, and seven groups considering the best of ten runs. In all of these seven groups, except 50×20, the best known values for the PFSSP are optimal, and their average improvement is 0.7%. A comparison with the larger time scale shows that the minimum, average, and maximum RPD improve by another 0.3%. The overall average is now -0.32%, and in each group better than the average RPD of the methods for the PFSSP.

6.2.6 Amount of job reordering and buffer sizes

To validate our hypothesis that good non-permutation schedules exhibit only a few inversions of jobs, we have measured the number of operations that change their position in the non-permutation schedules found by our method. For all 1200 non-permutation schedules obtained in the previous experiment with a time limit of $30nm$ ms we determined the job reordering index of a schedule defined in Section 5.4.5 In average over all schedules the job reordering index was less than

Table 6.8: Comparison of buffer sizes for permutation and non-permutation solutions on the instances of Taillard (1993).

	Schedule type					
	Permutation			non-permutation		
Size	Min.	Avg.	Max.	Min.	Avg.	Max.
20×5	2.10	2.48	3.10	2.10	2.41	3.10
20×10	2.30	2.44	2.70	2.10	2.42	2.80
20×20	2.30	2.41	2.50	2.00	2.31	2.90
50×5	5.30	6.44	7.70	4.50	5.59	6.60
50×10	3.10	3.97	4.90	3.00	3.88	4.70
50×20	3.50	4.14	4.90	3.50	4.03	5.10
100×5	9.30	11.25	13.70	6.30	8.00	10.00
100×10	6.10	7.09	8.40	5.40	6.49	7.90
100×20	5.50	6.55	7.90	5.30	6.27	7.40
200×10	9.50	11.54	14.70	8.10	9.71	11.60
200×20	7.60	9.09	11.40	6.90	8.46	10.60
500×20	13.30	16.19	21.30	12.10	13.78	16.40
Averages	5.83	6.97	8.60	5.11	6.11	7.42

0.08, i.e., less than one inversion for every 10 jobs between adjacent machines, with a maximum of 0.325.

We finally compare the buffer sizes obtained by non-permutation schedules to those of permutation schedules. The non-permutation schedules have been obtained by our heuristic, the permutation schedules have been produced by our implementation of the IGA of Ruiz & Stützle (2007). We compare the minimum, average, and maximum buffer sizes on the instances of Taillard (1993), in 10 replicates obtained with a time limit of $30nm$ ms in Table 6.8. The results show that the buffer requirements of non-permutation schedules are consistently slightly smaller than those of the corresponding permutation schedules, independent of the reduction of the makespan.

In summary, allowing non-permutation schedules yields better or comparable makespans to those of methods for the PFSSP, in a comparable time, and with a reduced buffer size. If we invest a factor of 10 more time, non-permutation schedules with about 0.7% shorter makespans, can be achieved. This is a good improvement compared to that of recent PFSSP methods, and produces makespans below the lower bound for permutation schedules for all except the largest instances.

6.3 Concluding remarks

Tandon, Cummings & LeVanu (1991) and Liao, Liao & Tseng (2006) have compared improved non-permutation schedules to permutation schedules obtained by the same methods. Tandon, Cummings & LeVanu (1991) find 1%–3% shorter makespans, in average, while the average improvement found by Liao, Liao & Tseng (2006) is only 0.1%–0.5% We have compared non-permutation schedules on standard instances to best known permutation schedules. Our results confirm the order of magnitude of these findings and lie with an average improvement of 0.75% between them. For instances with known optimal values, we obtain average improvements of 0.7%, ranging from 0.06% to 2.5%. This shows that considering non-permutation schedules can be beneficial. We also have found that the reduced makespan in non-permutation schedules can reduce buffer requirements.

To obtain non-permutation schedules we have proposed a constructive heuristic, a generalization of Taillard's method for efficiently computing makespans, and an iterated greedy algorithm based on the insertion of jobs with local job passing. Overall, they obtain results better or comparable to those of current heuristics for the PFSSP, and, with a modest amount of additional time, strictly better results, with a makespan often below the best lower bounds. Both heuristics produce significantly better results than all current heuristics for the FSSP.

Our study suggests some future research. First, there have been extensive studies of insertion order and tie-breaking rules for NEH, and their impact during construction and iterated destruction and reconstruction has found to be different (KALCZYNSKI; KAMBUROWSKI, 2011; DONG; HUANG; CHEN, 2008; FERNANDEZ-VIAGAS; FRAMINAN, 2014). It remains open if the same differences apply to the non-permutation case or if better rules can be found. Second, it may be interesting to study a similar method for limited buffer space. Finally, Liao, Liao & Tseng (2006) have found larger improvements in solution quality for non-permutation schedules when minimizing total flowtime, tardiness or total tardiness, and Pugazhendhi et al. (2003) observed the same for instances with missing operations. Thus, it may be interesting to generalize our method to other objective functions.

7 NOVEL PERMUTATION REPRESENTATION AND HEURISTICS FOR THE NON-PERMUTATION FLOW SHOP SCHEDULING PROBLEM WITH MAKESPAN CRITERION

Section 2.2.1 has defined the permutation and non-permutation FSSP formally, Chapter 4 has explored the theoretical differences between the permutation and non-permutation FSSP, and Chapters 5 and 6 have studied the practical differences between the permutation and the non-permutation FSSP with total completion time minimization and makespan minimization. At this point of the thesis, it is clear that non-permutation schedules are better than permutation schedules for the FSSP, although additional computational efforts are required to create them.

The methods proposed in this chapter reduce that computational effort. We propose a new permutation representation for the non-permutation FSSP in Section 7.1. we propose three new heuristics for the non-permutation FSSP in Section 7.2: a constructive heuristic NEH$_{\text{BR}}$ with the same time complexity $O(n^2m)$ than NEH, a non-permutation insertion local search with the same time complexity $O(n^2m)$ (per neighbourhood) than the permutation insertion local search, and a best-improvement reduced-neighbourhood non-permutation (BRN) local search with a time complexity of $O(nm)$ (per neighbourhood). We also propose four iterated greedy algorithms for the permutation and non-permutation FSSP in Section 7.3. Section 7.4 shows our computational results and we conclude in Section 7.5. The methods and results presented in this chapter are being prepared as a journal paper for future publication.

7.1 A new permutation representation for non-permutation schedules using pseudo-jobs

Permutation flow shop schedules are usually represented by a single permutation $\pi = (\pi(1), \ldots, \pi(n))$ of n jobs. Section 2.4 presents representations for job shop schedules that may be used to represent non-permutation flow shop schedules. For example, non-permutation schedules may be represented by a sequence $S = (\pi_1(1), \dots, \pi_1(n), \pi_2(1), \dots, \pi_2(n), \dots, \pi_m(1) \dots, \pi_m(n))$ of m permutations, where each permutation π_i gives the processing order for machine M_i, by a permutation $\rho = (\rho(1), \rho(2), \dots, \rho(n \times m))$ of the n jobs which are repeated m times, or by a single permutation $\tau = (\tau(1), \tau(2), \dots, \tau(n \times m))$ of the $n \times m$ operations. These permutations may represent any non-permutation flow shop schedule, even non-permutation schedules with many differences between machines that are unlikely to be good solutions. Also, the acceleration technique of Taillard (1990) cannot be applied directly on these representations of non-permutation flow shop schedules.

In this section, we propose a new representation that uses a permutation of jobs to represent permutation schedules, and that splits a job into blocks of operations when it is necessary to represent non-permutation schedules. To this end, we define a *pseudo-job* $J_j[i, i']$ that represents a block of operations of job J_j from machine M_i to machine $M_{i'}$, and without operations before machine M_i and after machine $M_{i'}$. We use permutations of pseudo-jobs to represent non-permutation flow shop schedules. We still write J_j for a pseudo-job $J_j[1, m]$ that contains all the operations to simplify the notation. In the following we use index i to refer to a machine M_i, and index j to refer to a job pseudo-job J_j or to a pseudo-job $\pi(j)$ in the j-th position of π. The only requirement for this representation to be a valid schedule is that the pseudo-jobs with operations of the same job must appear in order and not consecutively. If two pseudo-jobs $J_j[i, i']$ and $J_j[i', i'']$ with operations of the same job became adjacent, they must rejoin into one pseudo-job $J_j[i, i'']$.

For example, consider the 6×6 FSSP instance given in Table 7.1. The permutation $\pi = (J_5, J_4, J_6, J_2, J_1, J_3)$ represents the permutation schedule of Figure 7.1. There are some gaps in this permutation schedule that may be reduced by changing the order of some jobs after intermediate machines. If we anticipate job J_3 before job J_1 on machines M_5 and M_6, and delay job J_6 after job J_4 on machines M_4, M_5, and M_6, we obtain the shorter non-permutation schedule of Figure 7.2. Now, jobs J_1 and J_6 are divided by jobs J_3 and J_4 respectively. To use a permutation representation for the non-permutation schedule, we must split these job into blocks of operations and represent them as pseudo-jobs. Thus, job J_1 is divided into two pseudo-jobs: $J_1[1, 4]$ that contains the operations for the machines M_1, M_2, M_3, and M_4; and $J_1[5, 6]$ that contains the operations for the machines M_5 and M_6. Likewise, job J_6 is divided into two pseudo-jobs: $J_6[1, 3]$ that contains the operations for the machines M_1, M_2, and M_3; and $J_6[4, 6]$ that contains the operations for the machines M_4, M_5, and M_6. Using those pseudo-jobs, the non-permutation schedule of Figure 7.2 is represented by the permutation $\pi' = (J_5, J_6[1, 3], J_4, J_6[4, 6], J_2, J_1[1, 4], J_3, J_1[5, 6])$.

Table 7.1: 6×6 instance of the FSSP.

Jobs	Operations					
	M_1	M_2	M_3	M_4	M_5	M_6
J_1	3	6	3	3	4	3
J_2	4	3	5	3	5	2
J_3	6	5	2	2	2	4
J_4	4	5	2	2	5	5
J_5	2	2	5	6	3	5
J_6	2	3	5	5	3	3

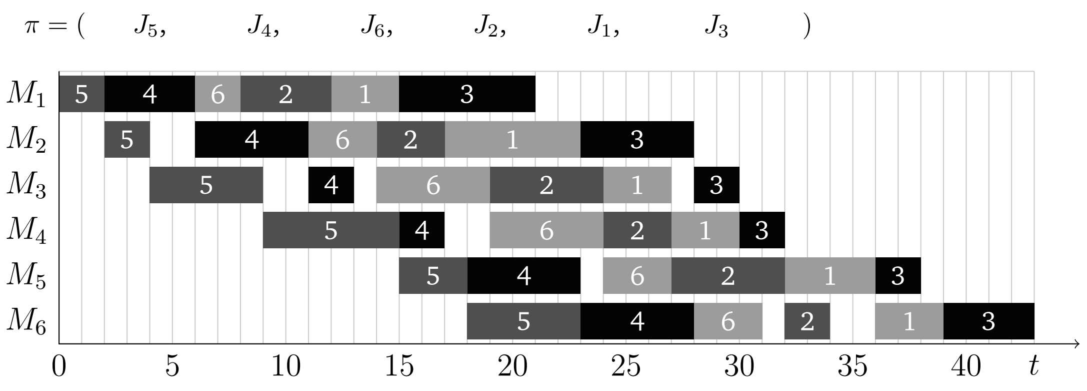


Figure 7.1: Permutation representation π and Gantt chart of a permutation schedule for the 6×6 FSSP instance given in Table 7.1.

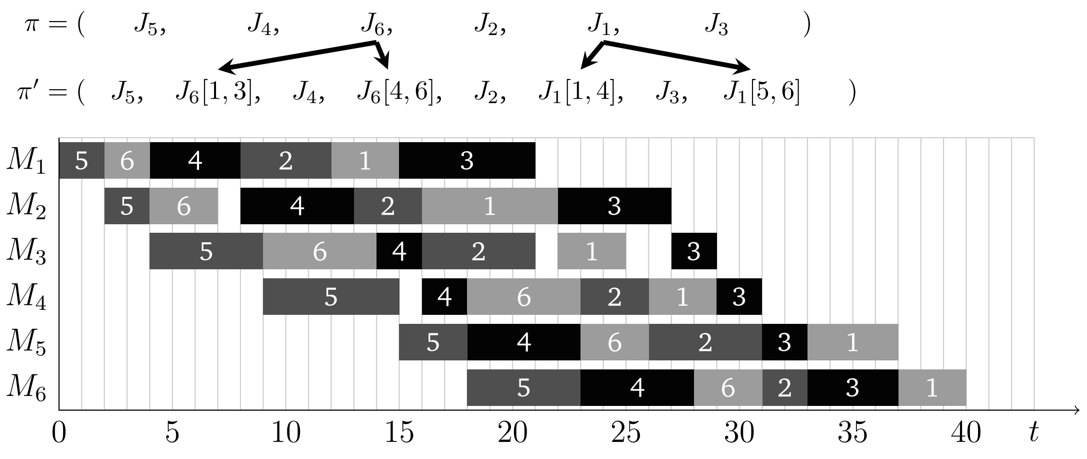


Figure 7.2: Permutation representation π' with job division and Gantt chart of a non-permutation schedule for the 6×6 FSSP instance given in Table 7.1.

7.2 Fast heuristics for the non-permutation FSSP

7.2.1 A constructive heuristic for the non-permutation FSSP with time complexity $O(n^2m)$

The best constructive heuristic for the permutation FSSP is the NEH heuristic proposed by Nawaz, Enscore & Ham (1983) with the acceleration technique of Taillard (1990). This method is described in Section 3.1.1. It builds a permutation schedule by iteratively inserting the jobs into a partial permutation schedule at the position which produces the smallest makespan. We extend this method to allow the insertion of jobs with an anticipation or a delay after some intermediate machine to create non-permutation schedules, and we extend the acceleration technique of Taillard (1990) to reduce the time complexity of the new constructive heuristic to the same of the NEH heuristic.

Algorithm 7.1 shows the proposed NEH$_{\text{BR}}$ constructive method to produce non-permutation schedules. NEH$_{\text{BR}}$ uses the same priority order π_o of NEH, i.e., non-increasing total processing time $\sum_{i\in[m]} p_{ij}$ for each job J_j. The division of the larger jobs during the first insertions with anticipation or delay may introduce many pseudo-jobs with large gaps in the schedule, and a later insertion of a smaller jobs may be unable to completely fill an existing large gap without delaying the other occupied machines, creating larger schedules. To avoid this, NEH$_{\text{BR}}$ inserts only a percentage p of the jobs considering anticipation or delay after an intermediate machine respectively (lines 11 to 27), after inserting the first larger jobs into straight positions like the NEH heuristic (lines 5 to 10).

NEH$_{\text{BR}}$ iteratively inserts the current job $\pi_o(j'')$ into a partial non-permutation schedule at the position which produces the smallest makespan until a schedule is complete. To determine the best position, it calculates the makespan values MC_j, $MC'_{i,j'}$, and $MC''_{i,j'}$ for inserting the current job into a straight position, with anticipation, and with delay after an intermediate machine respectively. NEH$_{\text{BR}}$ gives preference to straight insertions. Suppose that we are inserting job $J_{j'}$ into the partial non-permutation schedule $\pi = (\pi(1), \ldots, \pi(n')$. If the new makespan is MC_j, job $J_{j'}$ is inserted straight before position $j \in [|\pi|+1]$, producing the permutation $\pi' = (\pi(1), \ldots, \pi(j-1), J_{j'}, \pi(j), \ldots, \pi(n'))$. If the new makespan is MC''_{ij}, job $J_{j'}$ is inserted before position $j \in [|\pi|]$ with a delay after the i-th machine, i.e., job $J_{j'}$ is split into two pseudo-jobs: $J_{j'}[1, i]$ that is inserted before job $\pi(j)$ and $J_{j'}[i+1, m]$ that is inserted after job $\pi(j)$, producing the permutation $\pi' = (\pi(1), \ldots, \pi(j-1), J_{j'}[1, i], \pi(j), J_{j'}[i+1, m], \pi(j+1), \ldots, \pi(n')))$. The job in position j may represent a pseudo-job $\pi(j)[i', i'']$ that defines processing times

Algorithm 7.1 A constructive heuristic for the non-permutation FSSP.

```
Input: The percentage $p \in (0, \dots, 100)$ of jobs inserted considering job passing.
Output: A schedule $\pi$.
 1: function NEH$_{\text{BR}}(p)$
 2:     $\pi_o := (\pi_o(1), \dots, \pi_o(n))$ in non-increasing total processing time
 3:     $\pi := (\pi_o(1))$
 4:     $j'' := 2$
 5:     while $j'' \leq \lfloor p \times n/100 \rfloor$ do
 6:         Calculate $MC_j$ for the insertion positions $j \in [|\pi|+1]$ of job $\pi_o(j'')$ into $\pi$
 7:         $C_{\max} := \min_{j \in [|\pi|+1]}\{MC_j\}$
 8:         $\pi' := (\pi(1), \dots, \pi(j-1), \pi_o(j''), \pi(j), \dots, \pi(n'))$
 9:         $j'' := j''+1$
10:     end while
11:     while $j'' \leq n$ do
12:         Calculate $MC_j$ for the insertion of job $\pi_o(j'')$ into $\pi$ before position $j \in [|\pi|+1]$
13:         Calculate $MC'_{i,j'}$ for the insertion of job $\pi_o(j'')$ into $\pi$ after position $j' \in [|\pi|]$
                with an anticipation after the $i$-th machine, $i \in [2, m-2]$.
14:         Calculate $MC''_{i,j'}$ for the insertion of job $\pi_o(j'')$ into $\pi$ before position $j' \in [|\pi|]$
                with a delay after the $i$-th machine, $i \in [2, m-2]$.
15:         $C_{\max} := \min\{\min_{j \in [|\pi|+1]}\{MC_j\}, \min_{i \in [2,m-2], j' \in [|\pi|]}\{MC'_{i,j'}, MC''_{i,j'}\}\}$
16:         if $\exists j | MC_j = C_{\max}$ then
17:             $\pi' := (\pi(1), \dots, \pi(j-1), \pi_o(j''), \pi(j), \dots, \pi(n'))$
18:         else
19:             if $\exists i, j | MC'_{ij} = C_{\max}$ then
20:                 $\pi' := (\pi(1), \dots, \pi(j-1), \pi(j)[i', i], \pi_o(l), \pi(j)[i+1, i''], \pi(j+1), \dots, \pi(n')))$
21:             else    $(\exists i, j | MC''_{ij} = C_{\max})$
22:                 $\pi' := (\pi(1), \dots, \pi(j-1), \pi_o(l)[1, i], \pi(j), \pi_o(l)[i+1, m], \pi(j+1), \dots, \pi(n')))$
23:             end if
24:         end if
25:         $\pi := \pi'$
26:         $l := l + 1$
27:     end while
28:     return $\pi$
29: end function
```

only for the operations from machine $M_{i'}$ to machine $M_{i''}$ (where $1 \leq i' \leq i < i'' \leq m$), and the operations for other machines are missing. If the new makespan is MC'_{ij}, job $J_{j'}$ is inserted after position $j \in [|\pi|]$ with an anticipation after the i-th machine, i.e., pseudo-job $\pi(j)[i', i'']$ is split into two pseudo-jobs $\pi(j)[i', i]$ and $\pi(j)[i+1, i'']$, and job $J_{j'}$ is inserted in between, producing the permutation $\pi' = (\pi(1), \ldots, \pi(j-1), \pi(j)[i', i], J_{j'}, \pi(j)[i+1, i''], \pi(j+1), \ldots, \pi(n')))$.

NEH$_{\text{BR}}$ was designed without any complicated tie-breaking mechanism just as the original NEH heuristic. During the creation of the priority order π_o, both heuristics preserve the original order of the jobs with the same total processing time. They also select the first position when there is more than one insertion point that produces the smallest makespan.

The acceleration technique of Taillard (1990) reduces the time complexity of the NEH heuristic to $O(n^2 m)$. This acceleration technique might be used with the new representation of permutation of pseudo-job, but the number of elements $|\pi|$ may be greater than the number of inserted jobs and some pseudo-job may have missing operations due to the division of some jobs. Next, we redefine the acceleration technique of Taillard (1990) for pseudo-jobs to calculate the makespan MC_j for inserting the current job before a straight position j. The processing time of the operation of pseudo-job $\pi(j)$ on machine M_i is $p_{i,\pi(j)}$, we write $\nexists\ p_{i,\pi(j)}$ when the operation of pseudo-job $\pi(j)$ on machine M_i is missing.

The earliest completion time $e_{i,j}$ of pseudo-job $\pi(j)$ on machine M_i is defined as

$$e_{i,j} = \begin{cases} \max\{e_{i,j-1}, e_{i-1,j}\} + p_{i,\pi(j)}, & \text{if } \exists\ p_{i,\pi(j)}, \\ e_{i,j-1}, & \text{if } \nexists\ p_{i,\pi(j)}, \end{cases} \tag{7.1}$$

for $i \in [m]$ and $j \in [|\pi|]$, with $e_{0,j} = 0$ and $e_{i,0} = 0$. Similarly, the time $q_{i,j}$ between the end of processing and the latest starting time of job $\pi(j)$ on machine M_i is defined as

$$q_{i,j} = \begin{cases} \max\{q_{i,j+1}, q_{i+1,j}\} + p_{i,\pi(j)}, & \text{if } \exists\ p_{i,\pi(j)}, \\ q_{i,j+1}, & \text{if } \nexists\ p_{i,\pi(j)}, \end{cases} \tag{7.2}$$

for $i \in [m]$ and $j \in [|\pi|]$, with $q_{m+1,j} = 0$ and $q_{i,k+1} = 0$.

The earliest completion times of the jobs before the position $j \in [|\pi| + 1]$ will remain unchanged after the insertion of job $J_{j'}$ into π at that position, thus they are used to calculate the earliest relative completion time $f_{i,j}$ of job $J_{j'}$ on machine M_i (if it is inserted at position j) as

$$f_{i,j} = \max\{f_{i-1,j}, e_{i,j-1}\} + p_{i,\pi_o(l)}, \tag{7.3}$$

for $i \in [m]$, $j \in [|\pi|+1]$ with $f_{0,j} = 0$. Then, the makespan MC_j of the permutation schedule π' after the insertion of job $J_{j'}$ before position $j \in [|\pi|+1]$ is

$$MC_j = \max\nolimits_{i\in[m]}\{f_{i,j} + q_{i,j}\}. \tag{7.4}$$

At this point, we could insert job $J_{j'}$ into an straight position j of π that produces the best makespan $\min_{j\in[|\pi|+1]}\{MC_j\}$ like NEH heuristic, but we must evaluate the insertion of the job with anticipation or delay on some intermediate machine. Next, we extend the acceleration technique to calculate the makespan values $MC'_{i,j'}$ and $MC''_{i,j'}$ for inserting the current job with anticipation and with delay after an intermediate machine respectively.

The relative time $g_{i,j}$ between the end of processing and the latest starting time of job $J_{j'}$ (if it is inserted at position j) on machine M_i is defined as

$$g_{i,j} = \max\{g_{i+1,j}, q_{i,j}\} + p_{i,\pi_o(l)}, \tag{7.5}$$

for $i \in [m]$ and $j \in [|\pi|+1]$, with $g_{m+1,j} = 0$. Now we calculate the makespan $MC'_{i,j}$ for inserting job $J_{j'}$ after position j with an anticipation after machine M_i as

$$MC'_{i,j} = \begin{cases} \max\{f_{i,j+1} + g_{i+1,j}, & \\ \qquad \max\limits_{i'\in[i]}\{g_{i',j+1} + e_{i',j}\}, & \\ \qquad \max\limits_{i''\in[i+1,m]}\{f_{i'',j} + q_{i'',j}\}\}, & \text{if } \exists p_{i,\pi(j)} \wedge \exists p_{i+1,\pi(j)}, \\ \infty, & \text{if } \nexists p_{i,\pi(j)} \vee \nexists p_{i+1,\pi(j)}, \end{cases} \tag{7.6}$$

for $i \in [2, m-2]$ and $j \in [|\pi|]$.

We define the earliest completion time $e'_{i,j}$ of job $\pi(j)$ on machine M_i when scheduled right after job $J_{j'}$ as

$$e'_{i,j} = \begin{cases} \max\{e'_{i-1,j}, f_{i,j}\} + p_{i,\pi(j)}, & \text{if } \exists\ p_{i,\pi(j)}, \\ f_{i,j}, & \text{if } \nexists\ p_{i,\pi(j)}, \end{cases} \tag{7.7}$$

for $i \in [m]$ and $j \in [|\pi|]$, with $e'_{0,j} = 0$. Similarly, we define the time $q_{i,j}$ between the end of processing and the latest starting time of job $\pi(j)$ on machine M_i when scheduled right before job $J_{j'}$ as

$$q'_{i,j} = \begin{cases} \max\{q'_{i+1,j}, g_{i,j+1}\} + p_{i,\pi(j)}, & \text{if } \exists\ p_{i,\pi(j)}, \\ g_{i,j+1}, & \text{if } \nexists\ p_{i,\pi(j)}, \end{cases} \tag{7.8}$$

for $i \in [m]$ and $j \in [|\pi|]$, with $q'_{m+1,j} = 0$.

Now we calculate the makespan $MC''_{i,j}$ for inserting job $J_{j'}$ before position j with a delay after machine M_i as

$$MC''_{i,j} = \begin{cases} \max\{e'_{i,j} + q'_{i+1,j}, \\ \qquad \max\limits_{i' \in [i]}\{f_{i',j} + q_{i',j}\}, \\ \qquad \max\limits_{i'' \in [i+1,m]}\{g_{i'',j+1} + e_{i'',j}\}\}, & \text{if } \exists p_{i,\pi(j)} \wedge \exists p_{i+1,\pi(j)}, \\ \infty, & \text{if } \nexists p_{i,\pi(j)} \vee \nexists p_{i+1,\pi(j)}, \end{cases} \tag{7.9}$$

for $i \in [2, m-2]$ and $j \in [|\pi|]$,

Then, the new makespan is

$$C_{\max} = \min\{\min_{j \in [n]}\{MC_j\}, \min_{i \in [2,m-2], j \in [n]}\{MC'_{i,j}, MC''_{i,j}\}\}, \tag{7.10}$$

and job $J_{j'}$ is inserted into π to produce π' accordingly.

The proposed constructive heuristic NEH$_{\text{BR}}$ calculates three times the quantity of values that NEH calculates, and the number of pseudo-jobs n' in a non-permutation schedule is greater than the number of jobs n due to the division of jobs, thus each calculation has a time complexity of $O(n'm)$. Although a job may be divided in many pseudo-jobs, not all the jobs are divided, and each job insertion performs at most one division in two pseudo-jobs, thus the number of elements $n' = |\pi|$ in a non-permutation schedule π satisfies $n \leq n' < 2n$. Consequently, the complexity of the proposed NEH$_{\text{BR}}$ heuristic is also $O(n^2m)$, although it is expected to be at least three times more expensive than NEH in practice. For example, the maximum number of pseudo-jobs n' in our tests is observed for instances with 20 machines, and this number increases with a smaller pace than number of jobs n from $n' = 38$ for $n = 20$ to $n' = 580$ for $n = 500$.

Now, let us apply the proposed NEH$_{\text{BR}}$ heuristic to create a schedule for the 6×6 instance given in Table 7.1. NEH$_{\text{BR}}$ uses the priority order $\pi_o = (J_4, J_5, J_1, J_2, J_3, J_6)$ and creates the partial schedule $\pi = (J_5, J_4, J_2, J_1)$ in the first iterations in the same way as the NEH heuristic, because the minimum makespans are produced by straight insertions. Job J_3 is the next to be inserted. The makespan values MC_j, $MC'_{i,j}$, and $MC''_{i,j}$ in Table 7.2 would be the result after the insertion at position j straight, with anticipation, or with delay after the i-th machine, respectively. This time, the minimum makespan $C_{\max} = MC'_{4,4} = 38$ is shorter than the minimum value $MC_5 = 39$ for a straight insertion. Consequently, Job J_3 is inserted after job J_1 at position $j = 4$ with an anticipation after machine M_4 that splits job J_1 in two pseudo-jobs $J_1[1,4]$ and $J_1[5,6]$, producing the partial schedule $\pi'' = (J_5, J_4, J_2, J_1[1,4], J_3, J_1[5,6])$. After this insertion, the calculations differ from the original acceleration of Taillard. Table 7.3 shows the heads $e_{i,j}$ and tails $q_{i,j}$ for the partial schedule π'', and highlights in

Table 7.2: Makespan values for inserting job J_3 into the partial non-permutation schedule $\pi = (J_5, J_4, J_2, J_1)$ of the 6×6 instance given in Table 7.1.

MC	
j	MC_j
1	44
2	41
3	40
4	40
5	39

MC'			
j	$MC'_{2,j}$	$MC'_{3,j}$	$MC'_{4,j}$
1	46	43	41
2	41	40	40
3	43	40	40
4	40	39	**38**

MC''			
j	$MC'_{2,j}$	$MC'_{3,j}$	$MC'_{4,j}$
1	46	46	46
2	42	41	41
3	42	42	42
4	44	44	44

Table 7.3: Values $e_{i,j}$ and $q_{i,j}$ for schedule $\pi'' = (J_5, J_4, J_2, J_1[1,4], J_3, J_1[5,6])$ of the 6×6 instance given in Table 7.1.

$e_{i,j}$						
j	$e_{1,j}$	$e_{2,j}$	$e_{3,j}$	$e_{4,j}$	$e_{5,j}$	$e_{6,j}$
1	2	4	9	15	18	23
2	6	11	13	17	23	28
3	10	14	19	22	28	30
4	13	20	23	26	**28**	**30**
5	19	25	27	29	31	35
6	**19**	**25**	**27**	**29**	35	38

$q_{i,j}$						
j	$q_{1,j}$	$q_{2,j}$	$q_{3,j}$	$q_{4,j}$	$q_{5,j}$	$q_{6,j}$
1	38	35	33	28	22	19
2	36	32	24	21	19	14
3	31	27	22	17	14	9
4	27	24	17	14	**9**	**7**
5	24	18	13	11	9	7
6	**0**	**0**	**0**	**0**	7	3

Table 7.4: Makespan values for inserting job J_6 into the partial non-permutation schedule $\pi'' = (J_5, J_4, J_2, J_1[1,4], J_3, J_1[5,6])$ of the 6×6 instance given in Table 7.1.

M	
j	MC_j
1	43
2	42
3	41
4	43
5	46
6	48
7	44

MC'			
j	$MC'_{2,j}$	$MC'_{3,j}$	$MC'_{4,j}$
1	45	47	45
2	46	46	46
3	44	46	46
4	47	47	46
5	51	51	51
6	48	48	48

MC''			
j	$MC'_{2,j}$	$MC'_{3,j}$	$MC'_{4,j}$
1	45	48	46
2	42	**40**	44
3	46	46	45
4	49	48	45
5	50	47	47
6	44	44	48

bold the values that are repeated in successive machines due to divided pseudo-jobs. The last job to be inserted is J_6, and the makespan values that would result from its insertion are shown in Table 7.4. The final makespan $C_{\max} = MC''_{3,2} = 40$ is shorter than the minimum value $M_3 = 41$ for a straight insertion. Consequently, job J_6 is inserted before job J_4 at position $j = 2$ with a delay after machine M_3, i.e., job J_6 is divided in two pseudo-jobs: $J_6[1,3]$ that is inserted before job J_4 and $J_6[4,6]$ that is inserted after job J_4, producing the complete schedule $\pi' = (J_5, J_6[1,3], J_4, J_6[4,6], J_2, J_1[1,4], J_3, J_1[5,6])$ of Figure 7.2. The NEH heuristic would insert the last two jobs into straight positions, producing the schedule of Figure 7.1 with makespan $C_{\max} = 43$.

7.2.2 An insertion local search for the non-permutation FSSP with time complexity $O(n^2m)$ per neighbourhood

The most commonly used local search for the permutation FSSP explores the insertion neighbourhood (also called shift neighbourhood) that removes a job from the schedule and reinserts it into another position. This neighbourhood can use the acceleration of Taillard to calculate the best insertion position, examining the $(n-1)^2$ permutation insertion neighbours in a time of complexity $O(n^2m)$.

An insertion neighbourhood for the non-permutation FSSP removes a job from the schedule and reinserts it into another position: straight, with anticipation or with delay after an intermediate machine. The extended acceleration technique explained in Section 7.2.1 evaluates the $(n-1)^2(2m-5)$ non-permutation insertion neighbours in a time of complexity $O(n^2m)$.

The non-permutation insertion local search evaluates the reinsertion of jobs in a random cyclic order determined at the beginning of the search. In each iteration, it evaluates the reinsertion of the current job into a position that minimizes the makespan, straight, with anticipation or with delay, and the best schedule is updated with any improvement in the neighbourhood. The local search stops when there are no improvements found in the reinsertion of the n jobs.

7.2.3 A best-improvement reduced-neighbourhood non-permutation (BRN) local search for the FSSP

Algorithm 7.2 shows the proposed new local search for the non-permutation FSSP. We refer to this local search as BRN, standing for Best-improvement, Reduced-neighbourhood and Non-permutation. First, it determines the reduced neighbourhood R of pairs of adjacent jobs whose inversion may improve the

Algorithm 7.2 A best-improvement reduced-neighbourhood non-permutation local search for the FSSP.

```
Input: A non-permutation schedule $\pi'$ with makespan $C_{\max}(\pi')$.
Output: A local optimum schedule $\pi^*$ with makespan $C_{\max}(\pi^*) \leq C_{\max}(\pi')$.
 1: function BRN_LOCAL_SEARCH($\pi'$)
 2:   $\pi^* := \pi'$
 3:   $C_{\max}(\pi^*) := C_{\max}(\pi')$
 4:   repeat
 5:     $\pi := \pi'$
 6:     $C_{\max}(\pi) := C_{\max}(\pi')$
 7:     Compute $e_{i,j}$ and $q_{i,j}$ for all $i \in [m]$ and $j \in [|\pi|]$
 8:     $R := \{(\pi(j), \pi(j+1)) | \exists i,$
            $e_{i,j} + q_{i+1,j} = C_{\max}(\pi) \vee e_{i,j+1} + q_{i+1,j+1} = C_{\max}(\pi)\}$
 9:     $C_{\max}(\pi') := \infty$
10:     for each pair of adjacent jobs $(\pi(j), \pi(j+1)) \in R$ do
11:       Calculate $MC_j$, $MC'_{i,j}$ and $MC''_{i,j}$ for $i \in [2, m-2]$
12:       $C_j := \min\{C_{\max}(\pi'), MC_j, \min_{i \in [2,m-2]}\{MC'_{i,j}, MC''_{i,j}\}\}$
13:       $C_{\max}(\pi') := C_j$
14:     end for
15:     if $C_{\max}(\pi') < C_{\max}(\pi)$ then
16:       if $\exists j | MC_j = C_{\max}(\pi')$ then
17:         $\pi' := (\pi(1), \ldots, \pi(j-1), \pi(j+1), \pi(j), \pi(j+2), \ldots, \pi(n')))$
18:       else
19:         if $\exists i, j | MC'_{ij} = C_{\max}(\pi')$ then
20:           $\pi' := (\pi(1), \ldots, \pi(j-1), \pi(j+1)[1,i], \pi(j), \pi(j+1)[i+1,m], \pi(j+2), \ldots, \pi(n')))$
21:         else   $(\exists i, j | MC''_{ij} = C_{\max}(\pi'))$
22:           $\pi' := (\pi(1), \ldots, \pi(j-1), \pi(j)[1,i], \pi(j+1), \pi(j)[i+1,m], \pi(j+2), \ldots, \pi(n')))$
23:         end if
24:       end if
25:       if $C_{\max}(\pi') < C_{\max}(\pi^*)$ then
26:         $\pi^* := \pi'$
27:         $C_{\max}(\pi^*) := C_{\max}(\pi')$
28:       end if
29:     end if
30:   until $C_{\max}(\pi') \geq C_{\max}(\pi)$
31:   return $\pi^*$
32: end function
```

makespan. Then, it evaluates the inversion of the processing order of each pair of adjacent jobs $(\pi(j), \pi(j+1)) \in R$ completely, before, or after some intermediate machine. Finally, it applies the best inversion if it produces a shorter makespan. Those steps are repeated until the local search cannot reduce the makespan.

To determine the reduced neighbourhood R of the current schedule π, the BRN local search first calculates the earliest completion times $e_{i,j}$ and the time $q_{i,j}$ between the end of processing and the latest starting time of each job $\pi(j)$ on the i-th machine according to Equations (7.1) and (7.2). A critical operation of job $\pi(j)$ on the i-th machine satisfies $e_{i,j-1} + q_{i,j} = C_{\max}(\pi)$ if it follows another critical operation on the same machine, or $e_{i-1,j} + q_{i,j} = C_{\max}(\pi)$ if it follows another critical operation of the same job. Thus, the reduced neighbourhood R of the BRN local search only includes a pair of jobs $(\pi(j), \pi(j+1))$ if any of them has two consecutive critical operations on machines in positions $i \in [2, m-2]$ and $i+1$, this means $e_{i,j} + q_{i+1,j} = C_{\max}(\pi)$ or $e_{i,j+1} + q_{i+1,j+1} = C_{\max}(\pi)$. This is based on the neighbourhood proposed by Nowicki & Smutnicki (1996) for the job shop scheduling problem that limits the changes of processing order to the inversion of the first two and the last two operations of each block in the critical path, with exception of the first two and the last two operations of the schedule, because only those inversions may directly reduce the makespan of a job shop schedule.

The BRN local search performs the following calculations to evaluate the inversion of the processing order of a pair of consecutive jobs $(\pi(j), \pi(j+1)) \in R$. The completion time $e'_{i,j}$ of the operation of job $\pi(j+1)$ on the i-th machine if scheduled before $\pi(j)$ is defined as

$$e'_{i,j} = \begin{cases} \max\{e_{i,j-1}, e'_{i-1,j}\} + p_{i,\pi(j+1)}, & \text{if } \exists\, p_{i,\pi(j+1)} \\ e_{i,j-1}, & \text{if } \nexists\, p_{i,\pi(j+1)} \end{cases} \tag{7.11}$$

for $i \in [m]$, with $e'_{0,j} = 0$. The time $q'_{i,j}$ between the end of processing and the latest starting time of the operation of job $\pi(j)$ on the i-th machine if scheduled after $\pi(j+1)$ is defined as

$$q'_{i,j} = \begin{cases} \max\{q_{i,j+2}, q'_{i+1,j}\} + p_{i,\pi(j)}, & \text{if } \exists\, p_{i,\pi(j)}, \\ q_{i,j+2}, & \text{if } \nexists\, p_{i,\pi(j)}, \end{cases} \tag{7.12}$$

for $i \in [m]$, with $q'_{m+1,j} = 0$. Thus, the makespan MC_j of the permutation schedule π' that result of the exchange of the pair of consecutive jobs $(\pi(j), \pi(j+1))$ defined as

$$MC_j = \max_{i \in [m]}\{e'_{i,j} + q'_{i,j}\}. \tag{7.13}$$

The completion time $e''_{i,j}$ of the operation of job $\pi(j)$ on the i-th machine if scheduled after $\pi(j+1)$ is defined as

$$e''_{i,j} = \begin{cases} \max\{e'_{i,j}, e''_{i-1,j}\} + p_{i,\pi(j)}, & \text{if } \exists\ p_{i,\pi(j)}, \\ e'_{i,j-1}, & \text{if } \nexists\ p_{i,\pi(j)}, \end{cases} \tag{7.14}$$

for $i \in [m]$, with $e''_{0,j} = 0$. The time $q''_{i,j}$ between the end of processing and the latest starting time of the operation of job $\pi(j+1)$ on the i-th machine if scheduled before $\pi(j)$ is defined as

$$q''_{i,j} = \begin{cases} \max\{q'_{i,j}, q''_{i+1,j}\} + p_{i,\pi(j+1)}, & \text{if } \exists\ p_{i,\pi(j+1)} \\ q'_{i,j}, & \text{if } \nexists\ p_{i,\pi(j+1)} \end{cases} \tag{7.15}$$

for $i \in [m]$, with $q''_{m+1,j} = 0$.

The makespan $MC'_{i,j}$ for dividing the pseudo-job $\pi(j+1)[i', i'']$ after the i-th machine in two pseudo-jobs $\pi(j+1)[i', i]$ and $\pi(j+1)[i+1, i'']$ (where $1 \le i' \le i < i'' \le m$) and inserting job $\pi(j)$ between them is defined as

$$MC'_{i,j} = \begin{cases} \max\{e''_{i,j} + q_{i+1,j}, \\ \qquad \max\limits_{i' \in [i]}\{e'_{i',j} + q'_{i',j}\}, \\ \qquad \max\limits_{i'' \in [i+1,m]}\{e_{i'',j} + q_{i'',j+1}\}\}, & \text{if } \exists p_{i,\pi(j)} \wedge \exists p_{i+1,\pi(j)}, \\ \infty, & \text{if } \nexists p_{i,\pi(j)} \vee \nexists p_{i+1,\pi(j)}, \end{cases} \tag{7.16}$$

for $i \in [2, m-2]$, and the makespan $MC''_{i,j}$ for dividing the pseudo-job $\pi(j)[i', i'']$ after the i-th machine in two pseudo-jobs $\pi(j)[i', i]$ and $\pi(j)[i+1, i'']$ (where $1 \le i' \le i < i'' \le m$) and inserting job $\pi(j+1)$ between them is defined as

$$MC''_{i,j} = \begin{cases} \max\{e_{i,j+1} + q''_{i+1,j}, \\ \qquad \max\limits_{i' \in [i]}\{e_{i',j} + q_{i',j+1}\}, \\ \qquad \max\limits_{i'' \in [i+1,m]}\{e'_{i'',j} + q'_{i'',j}\}\}, & \text{if } \exists p_{i,\pi(j)} \wedge \exists p_{i+1,\pi(j)}, \\ \infty, & \text{if } \nexists p_{i,\pi(j)} \vee \nexists p_{i+1,\pi(j)}, \end{cases} \tag{7.17}$$

for $i \in [2, m-2]$,

After those calculations, the new makespan $C_{\max}(\pi')$ is defined as

$$C_{\max}(\pi') = \min_{j|(\pi(j),\pi(j+1)) \in R}\{MC_j, \min_{i \in [2,m-2]}\{MC'_{i,j}, MC''_{i,j}\}\}, \tag{7.18}$$

and the current schedule π is modified accordingly to create the new schedule π'.

Table 7.5: Makespan values for first iteration of the BRN local search on the schedule $\pi = (J_5, J_4, J_6, J_2, J_1, J_3)$ of the 6×6 instance given in Table 7.1.

MC		MC'				MC''			
j	MC_j	j	$MC'_{2,j}$	$MC'_{3,j}$	$MC'_{4,j}$	j	$MC'_{2,j}$	$MC'_{3,j}$	$MC'_{4,j}$
1	46	1	49	49	49	1	48	45	44
2	44	2	44	42	46	2	48	48	48
3	43	3	46	48	48	3	46	46	46
4	42	4	47	47	47	4	45	45	45
5	42	5	46	46	46	5	43	42	**41**

The calculations of the earliest completion times $e_{i,j}$ and the time $q_{i,j}$ between the end of processing and the latest starting time of each job $\pi(j)$ on the i-th machine have a time complexity of $O(nm)$. The other calculations are performed for each pair of adjacent jobs $(\pi(j), \pi(j+1))$ in the reduced neighbourhood R with a time complexity of $O(|R|m)$, where the size of the neighbourhood is limited by the number of jobs in the schedule $|R| < n$. Finally, with the extended acceleration technique, the BRN local search evaluates the $n(2m-5)$ neighbours in a time of complexity $O(nm)$.

Now, let us apply the BRN local search to the schedule $\pi = (J_5, J_4, J_6, J_2, J_1, J_3)$ of Figure 7.1. Neighbourhood R contains all pairs of adjacent jobs, because jobs J_4, J_6, J_2, and J_1 have consecutive critical operations in successive machines. The makespan values MC_j, $MC'_{i,j}$, and $MC''_{i,j}$ in Table 7.5 result from inverting the processing order of two consecutive jobs $(\pi(j), \pi(j+1)) \in R$ completely, before, or after the i-th machine, respectively. The new makespan $C_{\max}(\pi') = MC''_{4,5} = 41$ is shorter than the minimum value $M_4 = 42$ for a complete inversion. Consequently, job J_1 at position $j = 5$ is divided after machine M_4 into two pseudo-jobs: $J_1[1,4]$ that stays at the same position $j = 5$ and $J_1[5,6]$ that is inserted after job J_3, producing the new schedule $\pi' = (J_5, J_4, J_6, J_2, J_1[1,4], J_3, J_1[5,6])$. The next iteration of the BRN local search will produce the optimal schedule of Figure 7.2, and one last iteration will confirm that it is a local minimum to finish the search.

7.3 An iterated greedy algorithm for the non-permutation FSSP

Algorithm 7.3 shows a general iterated greedy algorithm. It starts by constructing an initial solution with a constructive heuristic. Then, it repeatedly removes d elements from the current solution, and reinserts them with a constructive heuristic to produce a perturbed solution. Optionally, a local search may improve each new solution. The new solution replaces the current solution according to an acceptance criterion. Finally, it returns the best solution found.

Algorithm 7.3 Iterated greedy algorithm.

```
Input: A number $d$ of elements to perturb, and temperature $T$ parameter.
Output: The best solution $s^*$ found during the search.
function IGA($d$, $T$)
    $s :=$ CONSTRUCTIVE_HEURISTIC( )
    $s :=$ LOCAL_SEARCH($s$)
    repeat
        $s^p :=$ REMOVE_ELEMENTS($s, d$)
        $s' :=$ CONSTRUCTIVE_HEURISTIC($s^p$)
        $s' :=$ LOCAL_SEARCH($s'$)
        if ACCEPTANCE_CRITERION($s, s', T$) then
            $s := s'$
            Update $s^*$ if necessary
        end if
    until some stopping criterion is satisfied
    return $s^*$
end function
```

The acceptance criterion is based on that proposed by Metropolis et al. (1953). The new solution s' replaces the current solution s if it is better, or with probability $P[\text{acceptance\_criterion}(s, s')] = \min\{e^{-(C_{\max}(s')-C_{\max}(s))/T}, 1\}$ for a *temperature* $T = \alpha\overline{p}/10$, where $\overline{p} = \sum_{j\in[n]}\sum_{i\in[m]} p_{ij}/nm$ is the average processing time of an operation and a parameter α.

Two parameters control the performance of the iterated greedy algorithm: the temperature factor α that controls the pliability of the acceptance criterion, and the number of jobs d to be reconstructed that controls the strength of the perturbation. We set the number of jobs to $d = 4$ and the temperature factor to $\alpha = 0.4$ in our implementations, following the calibration of Ruiz & Stützle (2007).

To create an iterated greedy algorithm for the non-permutation FSSP, we embed the constructive heuristic NEH$_{\text{BR}}$, the insertion local search, and the BRN local search that were proposed in the previous section. The NEH$_{\text{BR}}$ heuristic has a simple deterministic tie-breaking mechanism. Vasiljevic & Danilovic (2015) used random tie breaks to improve the results of NEH, with better results than complicated mechanisms. For this reason, our iterated greedy algorithm randomly breaks any tie found in the reconstruction and local search phases, after the first solution is complete.

We propose three variations of the iterated greedy algorithm that use the constructive heuristic NEH$_{\text{BR}}$ and differ in the used local search: IG$_{\text{i}}$ uses the proposed insertion local search, IG$_{\text{b}}$ uses the proposed BRN local search, and IG$_{\text{bi}}$ uses both the BRN local search and the insertion local search successively. We refer to our implementation of IG_RS$_{\text{LS}}$ for the permutation FSSP as IG$_{\text{p}}$, it also breaks ties randomly after the first solution is complete.

7.4 Computational Results

7.4.1 Experimental methodology

The proposed heuristics were implemented in C++, compiled with GNU C++ Compiler version 4.9.2 with optimization level 2 (-O2), and run on a PC with an AMD Opteron 6238 processor running at 2.9 GHz, and with 64 GB of main memory, using only one core in each execution. We have tested our algorithms on the 120 instances proposed by Taillard (1993) that were described in Section 3.4, and are the standard benchmark in the literature. We repeat each test once for the deterministic constructive heuristics and twenty times for the iterated greedy algorithms.

We present the quality of each result as the relative percentage deviation $100 \times (C_{\max} - C^*_{\max})/C^*_{\max}$ from the best known value $C^*_{\max}$, and as the average relative percentage deviations of the grouped instances and replications. The best known values are those reported by Taillard (2004) and listed in column "P" of Table 7.17. The best known values $C^*_{\max}$ may be improved by other researchers, and the comparison is impossible without the identification of the values used to calculate the relative percentage deviations. The NEH relative deviation in Table 7.6 or the average $C^*_{\max}$ in Tables 7.15 and 7.16 may confirm rapidly if other publications use the same best known values. We ran the iterated greedy algorithms for time $n(m/2)\rho$ ms with $\rho \in \{30, 60, 90\}$. The values of ρ were selected to match the running time of the compared methods.

7.4.2 Calibration of the constructive heuristic NEH$_{\text{BR}}$

First, we study the percentage p of jobs that the NEH$_{\text{BR}}$ heuristic shall insert using non-permutation insertions after the permutation insertion of the first $100-p$ percent of jobs. To do this, we vary the percentage $p \in \{0, 10, \ldots, 100\}$, where a percentage $p = 0$ represents a complete run of the original NEH heuristic for the permutation FSSP, and a percentage $p = 100$ represents a complete run of the NEH$_{\text{BR}}$ heuristic considering non-permutation insertions for every job. Both NEH and NEH$_{\text{BR}}$ heuristics are deterministic, thus we run them once on each instance. Table 7.6 shows the average (over 10 instances with the same size) of the relative percentage deviations for the NEH$_{\text{BR}}$ heuristic with a percentage p of non-permutation insertions. The best overall average result is 2.789 achieved by the NEH$_{\text{BR}}$ heuristic with a combination of 40 percent of permutation insertions followed by $p = 60$ percent of non-permutation insertions. Thus, we use this combination for the NEH$_{\text{BR}}$ to initialize the iterated greedy algorithms for the non-permutation FSSP.

Table 7.6: Average relative percentage deviations for the NEH and NEH$_{\text{BR}}$ heuristics.

Instances	NEH	Percentage p of jobs that consider non-permutation insertions									NEH$_{\text{BR}}$
	0	10	20	30	40	50	60	70	80	90	100
20×5	3.300	3.268	2.992	3.114	2.817	2.809	2.836	**2.796**	**2.796**	**2.796**	**2.796**
20×10	4.601	4.135	3.769	3.535	3.206	3.405	**2.637**	3.262	3.574	3.427	3.441
20×20	3.731	3.585	3.431	2.860	2.729	2.841	2.727	2.928	2.902	2.790	**2.349**
50×5	0.727	0.684	0.775	0.718	**0.631**	0.708	0.690	0.698	0.741	0.741	0.758
50×10	5.073	4.783	4.695	4.649	4.673	5.025	4.795	5.044	4.802	**4.080**	4.141
50×20	6.648	6.224	5.882	5.769	5.656	5.688	**5.200**	5.714	5.366	5.477	5.357
100×5	0.527	0.465	0.340	**0.327**	0.330	0.381	0.404	0.404	0.404	0.404	0.404
100×10	2.215	2.126	1.980	1.878	1.921	**1.620**	1.758	1.872	1.969	2.145	1.873
100×20	5.345	5.238	5.046	5.067	4.987	**4.811**	5.135	5.147	5.340	5.338	5.142
200×10	1.258	1.106	**1.040**	1.181	1.129	1.190	1.062	1.166	1.219	1.191	1.193
200×20	4.408	4.294	4.194	4.179	4.346	4.195	4.196	**4.054**	4.175	4.115	4.186
500×20	2.066	2.071	**2.005**	2.115	2.152	2.146	2.033	2.067	2.072	2.120	2.051
Averages	3.325	3.165	3.012	2.949	2.881	2.902	**2.789**	2.929	2.947	2.885	2.808

7.4.3 Comparison of the constructive heuristic NEH$_{\text{BR}}$ with other NEH variations

Next, we compare the NEH$_{\text{BR}}$ heuristic with different tie-breaking mechanisms for the NEH heuristic proposed in the literature. Table 7.7 shows the average relative percentage deviations for the original NEH heuristic and the proposed NEH$_{\text{BR}}$ heuristic, together with those reported by Fernandez-Viagas & Framinan (2014) for three tie-breaking mechanisms: NEH$_{\text{D}}$ proposed by Dong, Huang & Chen (2008),

Table 7.7: Average relative percentage deviations for the NEH$_{\text{BR}}$ heuristic compared to tie-breaking mechanisms.

Instances	NEH	NEH$_{\text{D}}$	NEH$_{\text{FF}}$	NEH$_{\text{D-FF}}$	NEH$_{\text{BR}}$
20×5	3.300	2.655	**2.293**	2.559	2.836
20×10	4.601	4.661	4.152	3.543	**2.637**
20×20	3.731	3.443	3.305	3.331	**2.727**
50×5	0.727	**0.497**	0.922	0.749	0.690
50×10	5.073	5.082	5.150	4.905	**4.795**
50×20	6.648	6.091	6.207	5.812	**5.200**
100×5	0.527	0.459	**0.378**	0.412	0.404
100×10	2.215	2.065	2.182	**1.719**	1.758
100×20	5.345	5.235	**5.021**	5.147	5.135
200×10	1.258	1.182	**0.984**	0.987	1.062
200×20	4.408	3.901	4.037	**3.885**	4.196
500×20	2.066	1.779	1.776	**1.713**	2.033
Averages	3.325	3.088	3.034	2.897	**2.789**

NEH$_{\text{FF}}$ proposed by Fernandez-Viagas & Framinan (2014), and NEH$_{\text{D-FF}}$ that uses the priority order of Dong, Huang & Chen (2008) and the tie-breaking mechanism of Fernandez-Viagas & Framinan (2014). The NEH$_{\text{FF}}$ and NEH$_{\text{D-FF}}$ heuristics have the best average results on the larger instances (with 100 jobs or more). Nevertheless, the proposed NEH$_{\text{BR}}$ heuristic finds the best overall results with a smaller average relative percentage deviation by 0.1. The insertion of a job to create a non-permutation schedule produces better results than calculating complicated tie breaks to create a permutation schedule.

Table 7.8 shows the average running times (in milliseconds) of the original NEH heuristic that only considers permutation insertions, the NEH$_{\text{BR}}$ heuristic with $p = 60$ percent of non-permutation insertions that shows the best results, and the NEH$_{\text{BR}}$ heuristic with $p = 100$ percent of non-permutation insertions that is the most expensive. It reports running times for instances of size 100$\times$10 or larger. The running times for smaller instances are less than one millisecond. The NEH$_{\text{BR}}$ heuristic with $p = 100$ expends approximately three times the running time of the NEH heuristic. This is consistent with the fact that both heuristics have the same time complexity $O(n^2m)$ and that NEH$_{\text{BR}}$ with $p = 100$ is a factor three more expensive than NEH. The NEH$_{\text{BR}}$ heuristic with $p = 60$ is 20% faster than with $p = 100$ and obtains the best results.

Table 7.8: Average running times (ms) for the NEH and NEH$_{\text{BR}}$ heuristics.

Instances	NEH	NEH$_{\text{BR}}$	NEH$_{\text{BR}}$
	$p = 0$	$p = 60$	$p = 100$
100$\times$10	0.8	1.7	2.1
100$\times$20	1.3	3.6	4.7
200$\times$10	3.2	6.9	8.3
200$\times$20	5.5	14.0	18.1
500$\times$20	34.3	83.8	106.9

The NEH$_{\text{BR}}$ heuristic was designed without any complicated tie-breaking mechanism just as the original NEH heuristic. The NFS heuristic from Section 6.1.1 proposes different tie-breaking mechanisms, and produces non-permutation schedules with an average relative percentage deviation of 2.67, but it has a time complexity of $O(n^3m^2)$. The NEHI heuristic proposed by Vasiljevic & Danilovic (2015) applies five different insertion tie breaks for each of five different priority orders, and produces permutation schedules with an average relative percentage deviation of 2.465, but runs random variations of NEH up to 26 times. It would be unfair to compare NEH$_{\text{BR}}$ with NFS and NEHI because of their higher running times. Besides, the same techniques may be implemented within NEH$_{\text{BR}}$ to improve its results.

Table 7.9: Average relative percentage deviations for the heuristics NEH and NEH_{BR} compared to the heuristics FRB and FRB_{BR}.

Instances	Permutation				Non-permutation			
	NEH	FRB2	FRB3	FRB5	NEH_{BR}	FRB2_{BR}	FRB3_{BR}	FRB5_{BR}
20×5	3.300	1.933	1.200	1.099	2.836	1.478	1.242	**0.929**
20×10	4.601	1.880	2.174	1.965	2.637	1.339	1.068	**1.059**
20×20	3.731	2.520	1.904	2.004	2.727	1.544	0.802	**0.481**
50×5	0.727	0.552	0.332	0.196	0.690	0.369	0.212	**-0.118**
50×10	5.073	3.341	3.006	2.476	4.795	2.598	1.838	**1.606**
50×20	6.648	4.251	3.534	3.417	5.200	3.663	2.956	**2.450**
100×5	0.527	0.255	0.186	0.115	0.404	0.137	0.102	**0.071**
100×10	2.215	1.170	0.947	0.902	1.758	1.077	0.739	**0.494**
100×20	5.345	3.413	3.242	2.578	5.135	3.070	2.511	**2.268**
200×10	1.258	0.757	0.539	0.345	1.062	0.608	0.323	**0.170**
200×20	4.408	2.640	2.321	1.880	4.196	2.476	1.962	**1.615**
500×20	2.066	1.197	1.091	0.774	2.033	1.101	0.837	**0.620**
Averages	3.325	1.992	1.706	1.479	2.789	1.622	1.216	**0.970**

7.4.4 Comparison of the constructive heuristic NEH_{BR} with the heuristics FRB

In Table 7.9 we compare the heuristics NEH and NEH_{BR} with our implementations of the heuristics FRB2, FRB3, and FRB5. This set of heuristics was proposed by Farahmand Rad, Ruiz & Boroojerdian (2009) and described in Section 3.1.2. We also implemented non-permutation versions FRB2_{BR}, FRB3_{BR}, and FRB5_{BR}, using our non-permutation insertion with anticipation and delay in the last 60% of the insertions, like in NEH_{BR}. To ensure that our implementations are deterministic, they use the same simple tie-breaking mechanisms of NEH_{BR}, and the insertion neighbourhood explores the reinsertion of the jobs in the order they were inserted.

The heuristics FRB2, FRB3, and FRB5 produce better solutions than NEH. The best average results among the permutation constructive heuristics are for FRB5. The small differences between the results of Farahmand Rad, Ruiz & Boroojerdian (2009) and our implementations may be attributed to different tie-breaking mechanisms. The heuristics FRB2_{BR}, FRB3_{BR}, and FRB5_{BR} produce better solutions than NEH_{BR}, but only FRB3_{BR}, and FRB5_{BR} produce better solutions than the constructive heuristics for permutation schedules. The best average results among all the constructive heuristics are for FRB5_{BR}, that produces better solutions than the best known permutation schedules in 13 of the instances, 6 of them have size 50×5, producing the only negative average in Table 7.9.

Table 7.10: Average running times (ms) for the FRB and FRB$_{\text{BR}}$ heuristics.

Instances	Permutation			Non-permutation		
	FRB2	FRB3	FRB5	FRB2$_{\text{BR}}$	FRB3$_{\text{BR}}$	FRB5$_{\text{BR}}$
100×10	13.9	53.7	78.4	26.8	122.8	203.2
100×20	51.0	92.0	169.1	104.8	251.1	601.4
200×10	57.7	431.9	549.7	107.6	961.2	1375.3
200×20	211.2	748.4	1441.3	434.4	1971.1	4709.7
500×20	1285.4	11344.4	19076.5	2498.3	27644.0	55778.7

The techniques proposed by Farahmand Rad, Ruiz & Boroojerdian (2009) to improve the results of NEH also apply and improve the results of NEH$_{\text{BR}}$, but they require larger times for their execution. Table 7.10 shows the average running times (in milliseconds) of of the heuristics FRB and FRB$_{\text{BR}}$. For the largest instances, the heuristics FRB2 and FRB2$_{\text{BR}}$ reguire a factor 30 more time than the corresponding NEH and NEH$_{\text{BR}}$. This factor is 240 for the heuristics FRB3 and FRB3$_{\text{BR}}$, and 500 for the heuristics FRB5 and FRB5$_{\text{BR}}$. This makes a direct comparison with NEH and NEH$_{\text{BR}}$ unfair. These larger processing times are still smaller than the shortest time limit of $n(m/2)30$ ms for our iterated greedy algorithms. The heuristic FRB3$_{\text{BR}}$ requires 18% and the heuristic FRB5$_{\text{BR}}$ requires 37% of that time limit for the largest instances. This allows the use of these heuristics as seeds for our iterated greedy algorithms.

7.4.5 Comparison of local search heuristics within the iterated greedy algorithms

Now we study how the use of the different proposed local searches influences the results and performance of the iterated greedy algorithm. Table 7.11 shows the average relative percentage deviations (over ten instances and twenty replications for each instance) for the three versions of our iterated greedy algorithm for the non-permutation FSSP proposed in Section 7.3 with time limits of $n(m/2)\rho$ ms, $\rho \in \{30, 60, 90\}$. IG$_{\text{b}}$ uses the BRN local search and IG$_{\text{i}}$ uses the insertion local search. IG$_{\text{b}}$ produces consistently the best results for all three time limits and almost all the instances. IG$_{\text{b}}$ produces the best average results for all three time limits, and IG$_{\text{i}}$ produces the worst average results. The use of both BRN and insertion local searches in IG$_{\text{bi}}$ shows slightly better results than using the insertion local search alone, but the use of the BRN local search alone is much better. Moreover, the average results of IG$_{\text{b}}$ in time $n(m/2)30$ ms are better than that of both IG$_{\text{i}}$ and IG$_{\text{bi}}$ in twice that time.

Table 7.11: Average relative percentage deviations for the proposed iterated greedy algorithms for the non-permutation FSSP.

Instances	Time $n(m/2)30$ ms			Time $n(m/2)60$ ms			Time $n(m/2)90$ ms		
	IG$_{\text{i}}$	IG$_{\text{bi}}$	IG$_{\text{b}}$	IG$_{\text{i}}$	IG$_{\text{bi}}$	IG$_{\text{b}}$	IG$_{\text{i}}$	IG$_{\text{bi}}$	IG$_{\text{b}}$
20×5	**-0.385**	-0.382	-0.359	-0.386	-0.387	**-0.389**	-0.387	**-0.391**	**-0.391**
20×10	-1.368	-1.423	**-1.428**	-1.424	-1.461	**-1.491**	-1.442	-1.495	**-1.551**
20×20	-1.990	-2.102	**-2.145**	-2.045	-2.195	**-2.234**	-2.081	-2.237	**-2.253**
50×5	**-0.165**	**-0.165**	**-0.165**	**-0.165**	**-0.165**	**-0.165**	**-0.165**	**-0.165**	**-0.165**
50×10	0.106	0.110	**0.012**	0.017	0.032	**-0.009**	-0.011	-0.013	**-0.027**
50×20	-0.073	-0.104	**-0.320**	-0.282	-0.322	**-0.551**	-0.413	-0.445	**-0.649**
100×5	**-0.119**	-0.117	-0.108	-0.119	-0.120	**-0.123**	-0.122	-0.122	**-0.123**
100×10	0.026	0.049	**0.013**	-0.018	-0.004	**-0.046**	-0.039	-0.026	**-0.063**
100×20	0.768	0.779	**0.438**	0.516	0.557	**0.307**	0.430	0.437	**0.232**
200×10	-0.022	-0.007	**-0.038**	-0.036	-0.033	**-0.051**	-0.044	-0.044	**-0.058**
200×20	1.025	1.078	**0.679**	0.849	0.880	**0.553**	0.752	0.779	**0.500**
500×20	0.490	0.518	**0.354**	0.417	0.430	**0.305**	0.381	0.383	**0.292**
Averages	-0.142	-0.147	**-0.256**	-0.223	-0.232	**-0.325**	-0.262	-0.278	**-0.355**

Table 7.12: Average relative percentage deviations for the iterated greedy algorithm IG$_{\text{b}}$ for the non-permutation FSSP with different constructive heuristics as seeds.

Instances	Time $n(m/2)30$ ms			Time $n(m/2)60$ ms			Time $n(m/2)90$ ms		
	NEH$_{\text{BR}}$	FRB3$_{\text{BR}}$	FRB5$_{\text{BR}}$	NEH$_{\text{BR}}$	FRB3$_{\text{BR}}$	FRB5$_{\text{BR}}$	NEH$_{\text{BR}}$	FRB3$_{\text{BR}}$	FRB5$_{\text{BR}}$
20×5	-0.359	-0.356	**-0.365**	-0.389	-0.371	**-0.391**	-0.391	-0.383	**-0.392**
20×10	**-1.428**	-1.424	-1.375	**-1.491**	-1.474	-1.451	**-1.551**	-1.489	-1.502
20×20	**-2.145**	-2.081	-2.100	**-2.234**	-2.152	-2.208	**-2.253**	-2.216	-2.233
50×5	**-0.165**	**-0.165**	**-0.165**	**-0.165**	**-0.165**	**-0.165**	**-0.165**	**-0.165**	**-0.165**
50×10	**0.012**	0.054	0.048	**-0.009**	0.015	0.003	**-0.027**	-0.006	-0.019
50×20	-0.320	-0.288	**-0.351**	**-0.551**	-0.524	-0.530	-0.649	-0.639	**-0.657**
Averages	**-0.734**	-0.710	-0.718	**-0.807**	-0.778	-0.790	**-0.839**	-0.816	-0.828
100×5	-0.108	-0.120	**-0.121**	**-0.123**	-0.121	**-0.123**	**-0.123**	**-0.123**	**-0.123**
100×10	0.013	**0.007**	0.012	**-0.046**	-0.035	-0.039	-0.063	**-0.070**	-0.065
100×20	0.438	0.396	**0.375**	0.307	0.236	**0.210**	0.232	0.140	**0.128**
200×10	**-0.038**	-0.026	-0.029	**-0.051**	-0.040	-0.043	**-0.058**	-0.052	-0.046
200×20	0.679	0.663	**0.575**	0.553	0.548	**0.482**	0.500	0.489	**0.433**
500×20	0.354	0.342	**0.300**	0.305	0.275	**0.247**	0.292	0.251	**0.235**
Averages	0.223	0.210	**0.185**	0.158	0.144	**0.123**	0.130	0.106	**0.094**
Total Avg.	-0.256	-0.250	**-0.266**	-0.325	-0.317	**-0.334**	-0.355	-0.355	**-0.367**

7.4.6 Comparison of constructive heuristics as seeds of the iterated greedy algorithms

Table 7.12 shows the average relative percentage deviations (over ten instances and twenty replications for each instance) for the iterated greedy algorithm IG_b for the non-permutation FSSP with three different constructive heuristics as seeds: the heuristics FRB3_BR and FRB5_BR that obtained the best results in Table 7.9, and the simpler heuristic NEH_BR. The seed FRB5_BR produces the best average results for all three time limits, but these results are not consistent for all the instances. The seed NEH_BR produces the best average results for instances with 20 or 50 jobs, while the seed FRB5_BR produces the best average results for instances with 100 jobs or more. The differences between the averages in Table 7.12 are ten times smaller than those in Table 7.11, thus, the performance of the IGs depend more on the use of different local searches than on the use of different seeds. We want to keep our methods as simple as possible, thus we use the heuristic NEH_BR as initial solution of our iterated greedy algorithms for comparisons in the rest of the chapter.

7.4.7 Performance of the iterated greedy algorithms

Table 7.13 shows the average number of iterations (in thousands, over ten instances and twenty replications for each instance) for the three versions of our iterated greedy algorithm for the non-permutation FSSP and for our IG_p for the permutation FSSP, all with the time limit of $n(m/2)90$ ms. IG_p performs three times the number of iterations of IG_i. This is expected, because the reconstuction and the local search phases of IG_p use the permutation insertion of the NEH heuristic, and those of IG_i use the non-permutation insertion of the NEH_BR heuristic that performs three times the number of calculations. IG_bi performs 6% more iterations than IG_i, this indicates that the BRN local search slightly shortens the search for the insertion local search. IG_b is fastest, performing from 6 to 132 times more iterations than IG_i.

The number of iterations for our iterated greedy algorithms decreases when the size of the instance increases, with the exception of IG_b algorithm that keeps an average of 180 thousands of iterations. The reconstruction phase of our iterated greedy algorithm uses a non-permutation insertion with a time complexity of $O(nm)$. The non-permutation insertion local search explores a neighbourhood in a time of complexity $O(n^2m)$. Thus, the performance of both IG_i and IG_bi are absorbed by the insertion local search that consumes between 88.1 and 99.5 percent of the running time. The BRN local search explores a neighbourhood in a time of complexity $O(nm)$. That is comparable to the reconstruction phase. Consequently, in IG_b algorithm, the BRN local search consumes only between 18 and 42 percent of the running time, and

Table 7.13: Average number of iterations (in thousands) for the proposed iterated greedy algorithms with stop criterion of $n(m/2)90$ ms.

Instances	$\mathrm{IG_p}$	$\mathrm{IG_i}$	$\mathrm{IG_{bi}}$	$\mathrm{IG_b}$
20×5	61.2	29.2	29.6	180.1
20×10	81.0	24.0	24.9	161.3
20×20	101.7	22.6	23.5	151.1
50×5	31.9	15.9	16.3	205.3
50×10	31.1	9.9	10.8	165.9
50×20	37.8	8.5	9.7	144.4
100×5	17.8	9.2	9.3	219.3
100×10	19.1	7.3	7.7	192.0
100×20	17.7	4.8	5.5	154.3
200×10	10.9	4.4	4.6	209.3
200×20	9.3	2.9	3.2	173.2
500×20	4.4	1.5	1.6	203.4
Averages	35.3	11.7	12.2	180.0

the reconstruction phase consumes the remaining time. The number of iterations of $\mathrm{IG_b}$ appears to be constant because the time complexity $O(nm)$ of the reconstruction phase and the BRN local search increase with the size of the instance at the same pace as its time limit that is a multiple of nm ms.

7.4.8 State-of-the-art methods for the permutation and non-permutation FSSP

Table 7.14 gives an overview of the state-of-the-art methods to be compared: the iterated greedy algorithm $\mathrm{IG+TB_{FF}}$ proposed by Fernandez-Viagas & Framinan (2014) that reimplements the iterated greedy algorithm with insertion local search $\mathrm{IG\_RS_{LS}}$ of Ruiz & Stützle (2007) and includes their tie-breaking mechanism $\mathrm{TB_{FF}}$, the

Table 7.14: Overview of the state-of-the-art methods for the FSSP.

Source	Algorithm	Prm.	Environment	$\%S$
Fernandez-Viagas & Framinan (2014)	$\mathrm{IG+TB_{FF}}$	P	Core i7-930 2.8GHz	-3
Vallada & Ruiz (2009)	CIG_{12i} (12 islands)	P	12 Core2 Duo 2.4 Ghz	-17
This chapter	$\mathrm{IG_p}$	P	Opteron 6238 2.9GHz	0
Sadjadi, Bouquard & Ziaee (2008)	ACO+NLS	N	Pentium 4 1.7GHz	-41
Rossi & Lanzetta (2013a)	NNP−ACS	N	Pentium 4 3GHz	3
Benavides & Ritt (2016)	NFS+IGA(LS)	N	AMD FX-8150 3.6GHz	24
This chapter	$\mathrm{IG_b}$	N	Opteron 6238 2.9GHz	0

cooperative iterated greedy algorithm CIG$_{12i}$ proposed by Vallada & Ruiz (2009) uses the islands model of parallel optimization to evolve and later share solutions among 12 parallel iterated greedy algorithms IG_RS$_{\text{LS}}$ of Ruiz & Stützle (2007), the ant colony optimization algorithm with a non-permutation local search ACO+NLS proposed by Sadjadi, Bouquard & Ziaee (2008), the ant colony system for native non-permutation flow shop NNP−ACS proposed by Rossi & Lanzetta (2014), our iterated greedy algorithm for non-permutation flow shop NFS+IGA(LS) described in the previous chapter (BENAVIDES; RITT, 2016), and our iterated greedy algorithms IG$_\text{p}$ and IG$_\text{b}$ for the permutation and non-permutation FSSP respectively. The column "Prm." describes if the method was proposed for the permutation ("P") or for the non-permutation ("N") FSSP. The table also shows the hardware environment used for the tests by the researchers with its percentual speed factor defined as $\%S = 100\% \times (sp - sp_0)/sp_0$, where sp is the speed of their hardware and sp_0 is our hardware. Positive values indicate a faster hardware and negative values indicate slower hardware.

7.4.9 Comparison of the methods for the permutation FSSP

Table 7.15 shows average relative percentage deviations: for the IG+TB$_{\text{FF}}$, with a time limit of $n(m/2)90$ ms, average over ten instances and five replications for each instance; for our iterated greedy algorithm IG$_\text{p}$, with time limits $n(m/2)\rho$ ms, $\rho \in \{30, 60, 90\}$, average over ten instances and twenty replications for each instance; and for the CIG$_{12i}$, with a time limit of $n(m/2)60$ ms, average over one replication and ten instances. Table 7.15 also shows the average of the best known values $C^*_{\max}$ of Taillard (2004) used to calculate the relative percentage deviations.

Vallada & Ruiz (2009) tested their CIG$_{12i}$ in twelve parallel processors that are 20% slower than ours, and compared it with the best of twelve independent runs of their iterated greedy algorithm. The best average results of our IG$_\text{p}$ in $n(m/2)90$ ms are at most 0.01 percent worse than that of the CIG$_{12i}$ in less than the double of their adjusted time (90 ms $< 2(2.4/2.9)60$ ms), but it would be unfair to compare the results of twelve processors against one. For this reason, we isolated for each instance the best result among the first twelve replications of our IG$_\text{p}$, and included their average relative percentage deviations over ten instances in Table 7.15. The "best of 12" results of our IG$_\text{p}$ reduce the relative percentage deviations for CIG$_{12i}$ from 0.22 to 0.195 in $n(m/2)30$ ms, and to 0.154 in $n(m/2)90$ ms.

Fernandez-Viagas & Framinan (2014) tested their IG+TB$_{\text{FF}}$ on a hardware with similar performance than ours, where it reaches an average relative percentage deviation of 0.350 in $n(m/2)90$ ms. Our IG$_\text{p}$ reaches a better average relative percentage deviation of 0.304 in a third of that time, and of 0.230 in that time.

Table 7.15: Average relative percentage deviations for the methods for permutation FSSP with time limit of $n(m/2)\tau$ ms.

Instances	Average	IG+TB$_{\text{FF}}$	IG$_{\text{p}}$			CIG$_{12i}$	IG$_{\text{p}}$ (best of 12)		
	$C^*_{\max}$	τ=90	τ=30	τ=60	τ=90	τ=60	τ=30	τ=60	τ=90
20×5	1221.9	0.041	0.025	0.010	0.002	0.00	0.000	0.000	0.000
20×10	1513.6	0.024	0.012	0.002	0.002	0.00	0.000	0.000	0.000
20×20	2235.0	0.035	0.013	0.009	0.007	0.00	0.000	0.000	0.000
50×5	2736.4	0.004	0.000	0.000	0.000	0.00	0.000	0.000	0.000
50×10	2983.4	0.438	0.371	0.328	0.304	0.30	0.301	0.259	0.259
50×20	3710.3	0.858	0.656	0.539	0.485	0.50	0.388	0.339	0.290
100×5	5244.5	0.001	0.000	0.000	0.000	0.00	0.000	0.000	0.000
100×10	5627.4	0.169	0.116	0.075	0.058	0.04	0.023	0.021	0.018
100×20	6298.1	1.096	0.938	0.762	0.684	0.65	0.581	0.493	0.445
200×10	10670.3	0.078	0.057	0.046	0.041	0.04	0.026	0.026	0.026
200×20	11256.3	1.026	0.984	0.852	0.786	0.78	0.684	0.627	0.544
500×20	26362.8	0.428	0.480	0.422	0.395	0.37	0.336	0.282	0.265
Averages	6655.0	0.350	0.304	0.254	0.230	0.22	0.195	0.171	0.154

Table 7.16: Average relative percentage deviations for the methods for non-permutation FSSP with time limit of $n(m/2)\tau$ ms.

Instances	Average	NNP−ACS	ACO+NLS	NFS+IGA(LS)		IG$_{\text{b}}$		
	$C^*_{\max}$	T=NR	τ=90	τ=60	τ=60m	τ=30	τ=60	τ=90
20×5	1221.9	1.90	-0.075	-0.33	-0.34	-0.359	-0.389	**-0.391**
20×10	1513.6	6.50	-0.009	-1.39	**-1.60**	-1.428	-1.491	-1.551
20×20	2235.0	6.90	-0.073	-2.00	**-2.45**	-2.145	-2.234	-2.253
50×5	2736.4	2.43	0.027	-0.16	-0.16	**-0.165**	**-0.165**	**-0.165**
50×10	2983.4	11.74	0.496	0.38	0.08	0.012	-0.009	**-0.027**
50×20	3710.3	14.51	1.120	0.15	**-0.74**	-0.320	-0.551	-0.649
100×5	5244.5	1.96	-0.007	-0.12	**-0.13**	-0.108	-0.123	-0.123
100×10	5627.4	7.93	0.363	0.22	0.01	0.013	-0.046	**-0.063**
100×20	6298.1	14.36	0.327	1.03	0.42	0.438	0.307	**0.232**
200×10	10670.3	6.33	NR	0.13	0.00	-0.038	-0.051	**-0.058**
200×20	11256.3	15.07	NR	1.14	0.74	0.679	0.553	**0.500**
500×20	26362.8	11.60	NR	0.64	0.38	0.354	0.305	**0.292**
Averages	6655.0	8.44		-0.03	-0.32	-0.256	-0.325	**-0.355**

NR: not reported.

We attribute the better results of IG_p to a better implementation of the algorithm. Both IG+TB_FF and our IG_p are reimplementations of the IG_RS_LS of Ruiz & Stützle (2007) with different tie-breaking mechanisms. A deeper research on tie-breaking mechanisms is necessary to confirm if the higher diversification caused by the random tie-breaking mechanism in our permutation reinsertion or the random visiting order of our local search produce better results than the tie-breaking mechanisms of Fernandez-Viagas & Framinan (2014). Nevertheless, we will be able to fairly compare in Section 7.4.11 the results of IG_p with those of the iterated greedy algorithms proposed for the non-permutation FSSP.

7.4.10 Comparison of the methods for the non-permutation FSSP

Table 7.16 shows the average relative percentage deviations: for the NNP−ACS, with processing time not reported, average over ten instances and the best of ten replications for each instance; for the ACO+NLS, with a time limit of $n(m/2)90$ ms for the ACO plus 10 s for the non-permutation local search, average over one replication on ten instances; for the NFS+IGA(LS), with time limits of $n(m/2)\rho$ ms, $\rho \in \{60, 60m\}$, average over ten instances and ten replications for each instance; and for our iterated greedy algorithm IG_b, with time limits $n(m/2)\rho$ ms, $\rho \in \{30, 60, 90\}$, average over ten instances and twenty replications for each instance. A negative relative percentage deviation indicates that the makespan of some non-permutation schedules are shorter than that of the best known permutation schedules. Table 7.16 also shows the average of the best known values $C^*_{\max}$ of Taillard (2004) used to calculate the relative percentage deviations.

Rossi & Lanzetta (2013a) did not report their processing times, and reported average relative deviations to best known non-permutation schedules. The average relative percentage deviations reported here were recalculated using those values as reference. All the average relative percentage deviations for NNP−ACS are positive. This means that the non-permutation schedules produced by NNP−ACS are worse than the best known permutation schedules.

The average results for ACO+NLS improve over the best known permutation schedules for instances with 20 jobs, and 100 jobs and 5 machines, and they are very close to the best known permutation schedules for the other instances. The ant colony algorithm of Sadjadi, Bouquard & Ziaee (2008) has a good performance, producing good permutation schedules for the non-permutation local search, but a short run of 10 s of the non-permutation local search is not enough to find better non-permutation schedules for large instances.

The average relative percentage deviation for NFS+IGA(LS) in $n(m/2)60$ ms is -0.03, and for $\mathrm{IG_b}$ is -0.325 in that time. The NFS+IGA(LS) needs m times that time to produce results close to our $\mathrm{IG_b}$. We attribute the better results of our $\mathrm{IG_b}$ to the acceleration technique used in the calculation of the insertion position, and to the faster BRN local search.

7.4.11 Comparison of the methods for the permutation and non-permutation FSSP

We can compare the relative percentage deviations for the permutation and non-permutation FSSP in Tables 7.16 and 7.15, because they were calculated with the same best known values. We focus on our iterated greedy algorithms $\mathrm{IG_p}$ and $\mathrm{IG_b}$ that show the best average results for the permutation and non-permutation FSSP, respectively.

The average relative percentage deviation for $\mathrm{IG_p}$ with time limit of $n(m/2)30$ ms is 0.304. $\mathrm{IG_p}$ finds the best known value for 62 of the instances within that time, mostly for instances with 20 jobs or 5 machines. The average relative percentage deviation for $\mathrm{IG_b}$ with that time limit is -0.256, i.e., shorter than that of the best known permutation schedules. In that time, $\mathrm{IG_b}$ finds the best known values for 13 of the instances, and finds non-permutation schedules that are better than the best known values for 67 of the instances, mostly for instances with up to 50 jobs or up to 10 machines.

$\mathrm{IG_p}$ it reduces the average relative percentage deviation to 0.254 in $n(m/2)60$ ms and to 0.230 in $n(m/2)90$ ms, but it fails to find more best known values after $n(m/2)30$ ms. $\mathrm{IG_b}$ also reduces the average relative percentage deviation to -0.325 in $n(m/2)60$ ms and to -0.355 in $n(m/2)90$ ms, and finds more solutions with makespan equal or better than the best known solutions with more time. $\mathrm{IG_b}$ finds the best known values for 15 of the instances, and finds non-permutation schedules that are better than the best known permutation schedules for 72 of the instances within $n(m/2)90$ ms. Table 7.17 shows the best non-permutation results of our $\mathrm{IG_b}$, together with the best known values for the permutation FSSP obtained from Taillard (2004), and for the non-permutation FSSP obtained from Vaessens (1996).

7.5 Concluding remarks

This chapter proposed a new permutation representation for non-permutation flow shop schedules and three new heuristics for the non-permutation FSSP: a constructive heuristic $\mathrm{NEH_{BR}}$, an insertion local search, and a best-improvement reduced-neighbourhood non-permutation (BRN) local search. The new permutation

Table 7.17: Best known solutions for permutation (P) from Taillard (2004) and non-permutation (NP) from Vaessens (1996), and best non-permutation from our results.

	P	NP	IG$_b$		P	NP	IG$_b$		P	NP	IG$_b$
20×5	**1278**	**1278**	**1278**	20×10	1582	1560	**1554**	20×20	2297	2293	**2237**
	1359	**1358**	**1358**		1659	1644	**1640**		2099	2092	**2061**
	1081	**1073**	**1073**		1496	1486	**1468**		2326	2313	**2243**
	1293	**1292**	1293		1377	1368	**1360**		2223	2223	**2175**
	1235	**1231**	**1231**		1419	1413	**1394**		2291	2291	**2250**
	1195	**1193**	**1193**		1397	**1369**	1371		2226	2221	**2160**
	1234	**1234**	**1234**		1484	**1428**	1432		2273	2267	**2223**
	1206	**1199**	**1199**		1538	**1527**	**1527**		2200	2183	**2159**
	1230	**1210**	**1210**		1593	**1586**	**1586**		2237	2227	**2184**
	1108	**1103**	**1103**		1591	**1559**	1566		2178	2178	**2122**
50×5	**2724**	**2724**	**2724**	50×10	**2991**	**2991**	3010	50×20	3850	3856	**3803**
	2834	**2834**	**2834**		**2867**	**2867**	2868		3704	3707	**3669**
	2621	**2612**	**2612**		2839	**2832**	2840		3640	3643	**3617**
	2751	**2751**	**2751**		3063	3063	**3061**		3723	3731	**3688**
	2863	**2853**	**2853**		2976	2976	**2966**		3611	3619	**3578**
	2829	**2825**	**2825**		3006	**2991**	**2991**		3681	3687	**3649**
	2725	**2716**	**2716**		**3093**	**3093**	**3093**		3704	3706	**3663**
	2683	**2683**	**2683**		3037	3026	**3025**		3691	3700	**3673**
	2552	**2545**	**2545**		2897	2887	**2882**		3743	3755	**3699**
	2782	**2776**	**2776**		**3065**	**3065**	3073		3756	3767	**3717**
100×5	**5493**	**5493**	**5493**	100×10	5770	**5759**	**5759**	100×20	**6202**	6228	6205
	5268	**5257**	**5257**		**5349**	**5349**	5350		**6183**	6210	6184
	5175	**5173**	5175		5676	5673	**5661**		**6271**	**6271**	6277
	5014	**4993**	4996		5781	**5759**	5779		6269	6269	**6250**
	5250	**5247**	**5247**		5467	**5455**	5457		**6314**	6319	6323
	5135	**5135**	**5135**		5303	**5293**	**5293**		6364	6403	**6345**
	5246	**5232**	**5232**		5595	5584	**5578**		**6268**	6292	6274
	5094	**5083**	5086		5617	5617	**5615**		6401	6423	**6391**
	5448	**5442**	**5442**		5871	**5852**	5858		**6275**	**6275**	6276
	5322	**5318**	**5318**		**5845**	**5845**	**5845**		**6434**	**6434**	6437
200×10	10862	**10857**	**10857**	200×20	**11195**	**11195**	11237	500×20	**26059**	**26059**	26139
	10480	10480	**10467**		**11203**	11223	11245		**26520**	**26520**	26615
	10922	**10922**	**10922**		**11281**	11337	11362		**26371**	**26371**	26434
	10889	**10862**	10871		**11275**	11299	11297		**26456**	**26456**	26532
	10524	10524	**10493**		**11259**	11260	11280		**26334**	**26334**	26375
	10329	**10329**	10331		**11176**	11189	11223		**26477**	**26477**	26502
	10854	**10836**	10854		**11360**	11386	11382		**26389**	**26389**	26412
	10730	**10727**	10728		**11334**	**11334**	11369		**26560**	**26560**	26615
	10438	**10419**	**10419**		**11192**	**11192**	11218		**26005**	**26005**	26061
	10675	**10675**	**10675**		**11288**	11313	11309		**26457**	**26457**	26510

representation for the non-permutation FSSP uses pseudo-jobs to represent blocks of operations of divided jobs. The constructive and local search heuristics are based on the new permutation representation for the non-permutation FSSP and on an extension of the acceleration technique of Taillard (1990). This chapter also proposed four iterated greedy algorithms that embed the proposed constructive and local search heuristics.

The proposed NEH$_{\text{BR}}$ constructive heuristic for the non-permutation FSSP shows the best overall results among the compared heuristics and tie-breaking mechanisms, and has the same time complexity of $O(n^2m)$ than the NEH heuristic. The proposed insertion local search for the non-permutation FSSP has the same time complexity of $O(n^2m)$ per neighbourhood than the insertion local search for the permutation FSSP. The proposed BRN local search for the non-permutation FSSP has a time complexity of $O(nm)$ per neighbourhood. The iterated greedy algorithm IG$_{\text{b}}$ uses the BRN local search and produces the best average results when compared to IG$_{\text{i}}$ and IG$_{\text{bi}}$ algorithms that use the insertion local search.

Although IG$_{\text{p}}$ has the best performance among the methods to solve the permutation FSSP, IG$_{\text{b}}$ finds better non-permutation schedules in the same time, which are shorter than the best known permutation schedules in many cases. Thus IG$_{\text{b}}$ produces the best average results compared to the state-of-the-art methods for the permutation and the non-permutation FSSP.

The NEH$_{\text{BR}}$ heuristic was designed without any complicated tie-breaking mechanisms just as the original NEH heuristic, and our iterated greedy algorithms break ties randomly after the first schedule is complete. Fernandez-Viagas & Framinan (2014) and Vasiljevic & Danilovic (2015) improve the results of the NEH and the iterated greedy algorithm for the permutation FSSP by applying different tie-breaking mechanisms. We wish to study in a near future how different tie-breaking mechanisms influence the results of the NEH$_{\text{BR}}$ heuristic and the iterated greedy algorithms for the non-permutation FSSP.

Part III

Heuristics for the heterogeneous workforce assignment and flow shop scheduling problem

8 SHOP SCHEDULING WITH A HETEROGENEOUS WORKFORCE

Section 1.1.2 has introduced and motivated the consideration of the heterogeneity of the workforce in the scheduling process, and Section 3.5 has presented examples in the literature that successfully introduce human diversity in the planning and scheduling processes of productive systems. This chapter introduces the combined problem of flow shop scheduling and worker assignment. Section 8.1 motivates treating these problems jointly with an example of the problem of finding a flow shop schedule when the workforce is heterogeneous. Section 8.2.1 defines the heterogeneous workforce assignment and flow shop scheduling problem (or Het-FSSP). Section 8.2.2 defines the heterogeneous workforce assignment and job shop scheduling problem (or Het-JSSP). Section 8.3 shows the complexity of these problems, and a new set of instances which models the situation of heterogeneous workers for these problems is defined in Section 8.4.

8.1 An example of a FSSP with heterogeneous workers

Let us begin with a simple example of the flow shop scheduling problem. Table 8.1 shows the processing times of four jobs that must be processed on four machines in a flow shop configuration. Figure 8.1 shows an optimal schedule for this instance that has a makespan of 15.

Now, let us explain a flow shop environment where the workers are heterogeneous. We represent this scenario in Table 8.2. In this scenario, the processing time of an operation performed on a machine will also depend on the worker assigned to operate it. We have four workers, which we want to assign to the four machines. The workers may have different performance for the same operations, and this is represented by different processing times. In this example, we chose processing times randomly in the interval $[p, 2p]$, for a processing time of p in the original flow shop instance

Table 8.1: 4×4 instance of the FSSP.

	Machine			
Job	M_1	M_2	M_3	M_4
J_1	3	1	1	3
J_2	3	1	3	3
J_3	1	2	1	2
J_4	1	3	3	1

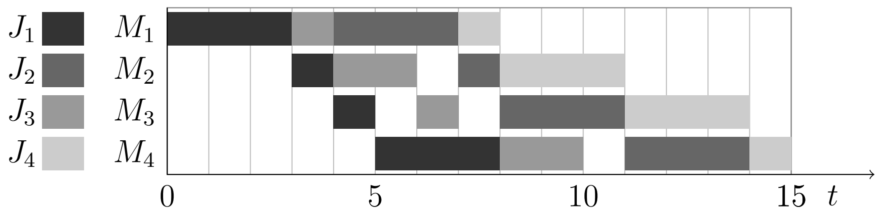


Figure 8.1: Optimal schedule for the 4×4 FSSP instance.

Table 8.2: An instance of the Het-FSSP.

	Worker w_1				Worker w_2				Worker w_3				Worker w_4			
Job	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4
J_1	5	2	2	4	∞	2	1	6	5	1	2	4	4	1	2	3
J_2	3	2	4	3	∞	2	3	5	5	1	6	6	4	1	6	4
J_3	1	4	2	3	∞	4	1	4	1	3	2	4	1	2	2	3
J_4	1	5	6	1	∞	6	3	2	1	5	5	2	1	3	4	1

given in Table 8.1. Additionally, some workers may be unable to operate a subset of the machines (for therapeutic or strategic reasons). In the example, worker w_2 is unable to operate machine M_1, and this is represented by processing times of ∞ for all operations on this machine.

With the selection of processing times in the interval $[p, 2p]$, we can expect the processing times and the makespan to increase in average by 50%. In our example, the increase in the average of the processing times (excluding incompatibilities) is 50.8%. Thus, we could expect to achieve a makespan of about 22.

If we evaluate all the 18 valid allocations of workers to machines fixing the schedule of Figure 8.1 that is optimal for the standard times of Table 8.1, we get an average makespan of 23.7, and a worst makespan of 27. The best makespan that can be obtained this way is 19, with the workers $w_4, w_3, w_2,$ and w_1 assigned to the machines in this order, as shown in Figure 8.2.

If we minimize the sum of the processing times for the assigned workers, the average result of all the 18 valid allocations is 48.3 and the worst is 56. The best sum is 38 with the workers $w_3, w_4, w_2,$ and w_1 assigned to the machines in this order. If we fix this worker assignment to later solve the scheduling problem, the best makespan that can be obtained is 18, as shown in Figure 8.3.

If we consider the heterogeneity of the workers jointly with the scheduling process, we can find an optimal solution with a makespan of 16. This optimal solution is shown in Figure 8.4. Note that this is a non-permutation schedule, since jobs J_1 and J_4 exchange their processing order on machine M_3. The best makespan that can be obtained with a permutation schedule is 17, and this schedule is shown in Figure 8.5. The workers $w_1, w_3, w_2,$ and w_4 are assigned to the machines in this order for both solutions, and the sum of the processing times is 39 for this assignment of workers. Neither the schedule nor the worker assignment are optimal if those problems are treated separately, the examples of Figures 8.2 and 8.3 produce worse makespans of 19 and 18 respectively.

The example shows that the problems of scheduling and assignment of workers are related, and that the consideration of both problems as one joint problem can improve over strategies that consider them separately. In this case the optimal solution of 16 is 11% less than the best assignment for the optimal original flow shop schedule, and 15% less than the best schedule for the worker assignment with the minimal sum of processing times. The example also shows that considering non-permutation schedules can improve the results of permutation schedules. The makespan of the optimal non-permutation schedule is 6% less than the best possible permutation schedule in this case.

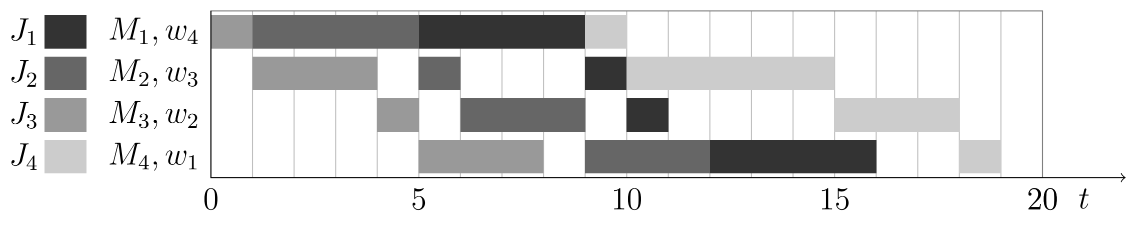


Figure 8.2: Best schedule for the 4×4 Het-FSSP instance obtained by assigning the workers while maintaining the optimal flow shop schedule.

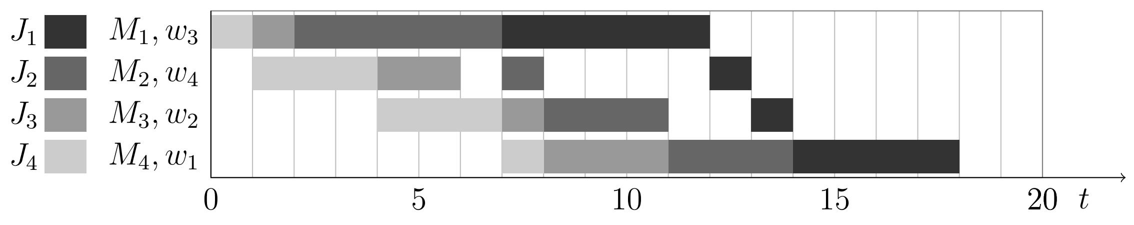


Figure 8.3: Best schedule for the 4×4 Het-FSSP instance obtained by solving a FSSP with a fixed assignment of workers that minimizes the sum of processing times.

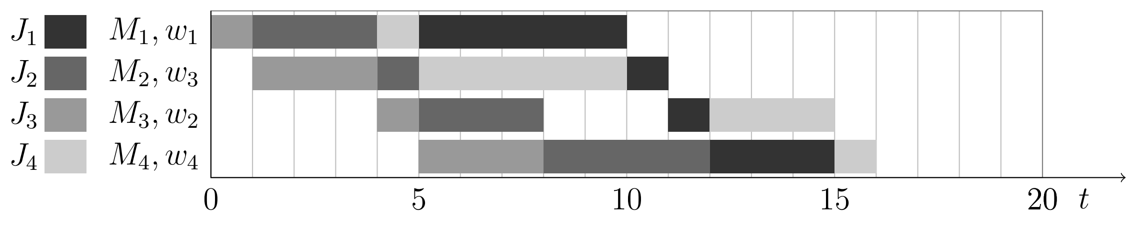


Figure 8.4: Optimal non-permutation schedule for the 4×4 Het-FSSP instance.

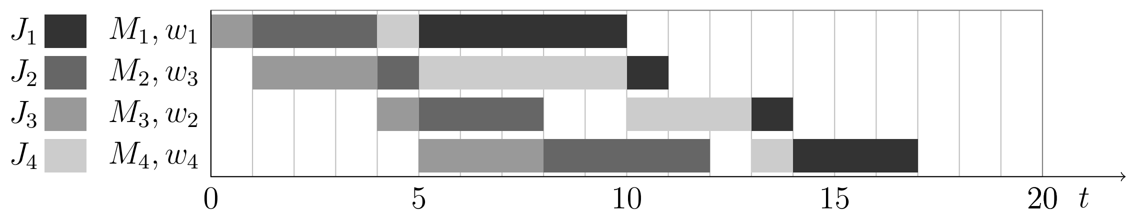


Figure 8.5: Best permutation schedule for the 4×4 Het-FSSP instance.

8.2 Mathematical formulations

8.2.1 A mathematical formulation of the Het-FSSP

This section presents a mathematical formulation of the Het-FSSP. We use index i for machines, j for jobs, and w for workers, and the notation $[n] = \{1, \ldots, n\}$. An integer linear program for the Het-FSSP can be formulated as

$$
\begin{array}{llrr}
\textbf{min.} & C_{\max}, & & (8.1)\\
\textbf{s.t.} & x_{mj} + p_{mj} \leq C_{\max}, & \forall j \in [n], & (8.2)\\
& x_{ij} + p_{ij} \leq x_{(i+1)j}, & \forall i \in [m-1], j \in [n], & (8.3)\\
& x_{ij} + p_{ij} \leq x_{ij'} + M(1 - y_{ijj'}), & \forall i \in [m], j \neq j' \in [n], & (8.4)\\
& y_{ijj'} + y_{ij'j} = 1, & \forall i \in [m], j \neq j' \in [n], & (8.5)\\
& p_{ij} = \sum_{w \in [m]} p_{ijw} z_{iw}, & \forall i \in [m], j \in [n], & (8.6)\\
& \sum_{w \in [m]} z_{iw} = 1, & \forall i \in [m], & (8.7)\\
& \sum_{i \in [m]} z_{iw} = 1, & \forall w \in [m], & (8.8)\\
& x_{ij} \geq 0, & \forall i \in [m], j \in [n], & (8.9)\\
& y_{ijj'} \in \{0, 1\}, & \forall i \in [m], j \neq j' \in [n], & (8.10)\\
& z_{iw} \in \{0, 1\}, & \forall i, w \in [m]. & (8.11)
\end{array}
$$

In this formulation, the auxiliary variables x_{ij} and p_{ij} represent the starting time and the processing time of the operation of job J_j that must be executed on machine M_i. Constraints (8.2)–(8.5) treat the flow shop subproblem for a fixed worker allocation: constraint (8.2) defines $C_{\max}$ as the latest completion time, and constraints (8.3) and (8.4) set the starting time of all operations according to their precedence constraints. The constant M in constraint (8.4) should be sufficiently large (e.g. $\sum_{i \in [m]} \sum_{j \in [n]} \max_{w \in [m]} p_{ijw}$). We introduce a binary decision variable $y_{ijj'}$ for each pair of operations of different jobs J_j and $J_{j'}$ that must be processed on the same machine M_i. The variable $y_{ijj'}$ equals 1 if job J_j precedes job $J_{j'}$ on machine M_i. Thus, constraint (8.5) enforces the precedence relations for the operations on the same machine. The processing time p_{ij} of an operation depends on the worker assigned to its machine, and is defined in constraint (8.6) based on the processing time p_{ijw} of the operation of job J_j on machine M_i when executed by worker w, and on a binary assignment variable z_{iw} which equals 1 if worker w is assigned to machine M_i.

Constraints (8.7) and (8.8) ensure that the assignment of workers to machines is bijective.

Note that this formulation is identical to that of the FSSP in Section 2.2.1, excluding constraints (8.6)–(8.8) and (8.11). Furthermore, any mathematical formulation for a shop scheduling problem that does not multiply the processing times p_{ij} with other variables can be extended to the heterogeneous case by adding constraints (8.6)–(8.8) and (8.11). For example, this applies to the formulations of Manne (1960) and Liao & You (1992) for the JSSP, and our formulations in Section 2.2.

8.2.2 A mathematical formulation of the Het-JSSP

This section presents a mathematical formulation of the Het-JSSP that extends the formulation for the JSSP of Section 2.2.2. We use index i for machines, j for jobs, and w for workers. We also use the precedence relation set $R_j = \{(i,i')|o_{ij} \prec o_{i'j}\}$ for the precedence of all pairs of consecutive operations that belong to job J_j. The precedence relation set R_j is the only difference with the mathematical formulation of the Het-FSSP. An integer linear program for the Het-JSSP can be formulated as

$$
\begin{aligned}
\textbf{min.}\quad & C_{\max}, && && (8.12)\\
\textbf{s.t.}\quad & x_{mj} + p_{mj} \le C_{\max}, && \forall j \in [n], && (8.13)\\
& x_{ij} + p_{ij} \le x_{i',j}, && \forall (i,i') \in R_j, j \in [n], && (8.14)\\
& x_{ij} + p_{ij} \le x_{ij'} + M(1 - y_{ijj'}), && \forall i \in [m], j \neq j' \in [n], && (8.15)\\
& y_{ijj'} + y_{ij'j} = 1, && \forall i \in [m], j \neq j' \in [n], && (8.16)\\
& p_{ij} = \sum_{w\in[m]} p_{ijw} z_{iw}, && \forall i \in [m], j \in [n], && (8.17)\\
& \sum_{w\in[m]} z_{iw} = 1, && \forall i \in [m], && (8.18)\\
& \sum_{i\in[m]} z_{iw} = 1, && \forall w \in [m], && (8.19)\\
& x_{ij} \ge 0, && \forall i \in [m], j \in [n], && (8.20)\\
& y_{ijj'} \in \{0,1\}, && \forall i \in [m], j \neq j' \in [n], && (8.21)\\
& z_{iw} \in \{0,1\}, && \forall i, w \in [m]. && (8.22)
\end{aligned}
$$

In this formulation, the auxiliary variables x_{ij} and p_{ij} represent the starting time and the processing time of the operation of job J_j that must be executed on machine M_i. Constraints (8.13)–(8.16) treat the job shop subproblem for a fixed worker allocation. Constraint (8.13) defines $C_{\max}$ as the latest completion time, and constraints (8.14) and (8.15) set the starting time of all operations according to their precedence constraints. The precedence between two operations of a job j that are executed consecutively on machines i and i' is fixed by constraint (8.14). The constant M in constraint (8.15) should be sufficiently large (e.g. $\sum_{i\in[m]}\sum_{j\in[n]}\max_{w\in[m]} p_{ijw}$). We introduce a binary decision variable $y_{ijj'}$ for each pair of operations of different jobs J_j and $J_{j'}$ that must be processed on the same machine M_i, and whose precedence is not fixed. If job J_j precedes job $J_{j'}$ on machine M_i, then variable $y_{ijj'}$ equals 1. Thus, constraint (8.16) enforces the precedence relations for the operations on the same machine. Constraint (8.17) defines the processing time of an operation depending on the worker assigned to its machine, based on the processing time p_{ijw} of the operation of job J_j on machine M_i when executed by worker w, and on a binary assignment variable z_{iw} which equals 1 if worker w is assigned to machine M_i. Constraints (8.18) and (8.19) ensure that the assignment of workers to machines is bijective.

8.3 Size of the search space

Chapter 4 mention that the number of possible solutions for non-permutation FSSPs are $n!^{(m-2)}$ for the makespan criterion, and $n!^{(m-1)}$ for the total completion time criterion. Those numbers are closer to the $n!^m$ possible solutions for the JSSP, than to the $n!$ possible solutions for the permutation FSSP. But the limitation to only permutation schedules also eliminates the possibility of reaching some optimal solutions.

The additional worker assignment problem increases the number of possible solutions of the shop scheduling problems by a factor of $m!$. Thus, the number of possible solutions of the Het-FSSP are $m!n!^{(m-2)}$ for the makespan criterion and $m!n!^{(m-1)}$ for the total completion time criterion. Those numbers are close to the $m!n!^m$ possible solutions of the Het-JSSP. The FSSP and the JSSP are NP-hard. The Het-FSSP and the Het-JSSP include the FSSP and the JSSP respectively as special cases. Thus, the Het-FSSP and the Het-JSSP are also NP-hard, and are more difficult than the FSSP and the JSSP. Even a permutation Het-FSSP is NP-hard, and it has $m!n!$ possible solutions, but we focus on the non-permutation Het-FSSP, because the best solutions are more beneficial to the shop floor.

8.4 New instances

We created instances for the Het-FSSP, based on the well-known FSSP instances proposed by Carlier (1978) and Taillard (1993). We also created instances for the

Het-JSSP based on the JSSP instances of Taillard (1993), and on the instance *ft10* of Fischer & Thompson (1963). The instances of Taillard (1993) are the most used for the FSSP and JSSP. The instances of Carlier (1978) and Fischer & Thompson (1963) were selected for their smaller size. To create the instances, we replicated the processing times p_{ij} of a FSSP (or JSSP) instance for our workers and, assuming that they are processing times of a standard worker, we modified these times in two ways: First, the processing times were randomized to simulate the different abilities of the workers. Second, a fixed percentage of incompatibilities is introduced to represent the case of workers that are unable to operate some machines (for example, worker w_2 on machine M_1 in the instance given in Table 8.2). This procedure is repeated m times to create m different workers. Based on experiences with existing SWDs, we chose to generate instances with 0%, 10% and 20% of incompatibilities per worker, and we randomly increased the standard times p in the interval $[p, 2p]$ or $[p, 5p]$ for all the operations.

We refer to the created instances by the name of the base instance, followed by an i for processing times in $[p, 2p]$ or an I for processing times in $[p, 5p]$, and the percentage of incompatibilities. For example, tai21i20 is a Het-JSSP with processing times in $[p, 2p]$ and 20% of incompatibilities, and ta021I00 is a Het-FSSP with processing times in $[p, 5p]$ and no incompatibilities. The Het-FSSP instances are available at <http://inf.ufrgs.br/algopt/hetFS>, and the Het-JSSP instances are available at <http://inf.ufrgs.br/~ajbenavides/hetJS>.

8.5 Concluding remarks

This chapter proposes the heterogeneous workforce assignment and flow shop scheduling problem (or Het-FSSP), an extension of the flow shop scheduling problem that considers the assignment problem of heterogeneous workers to the machines. The motivation to treat these problems jointly is presented by an example of the problem of finding a flow shop schedule when the workforce is heterogeneous. Mathematical models for the Het-FSSP and the Het-JSSP are given as extensions of the FSSP and the JSSP respectively, showing that these extensions can easily be applied to other models of shop scheduling problems. Finally, a new set of instances for the new combined problems is presented to be used for computational experiments in the next chapters.

9 A SCATTER SEARCH WITH PATH RELINKING FOR THE HET-FSSP

Chapter 8 has introduced the heterogeneous workforce assignment and flow shop scheduling problem (or Het-FSSP), together with its theoretical definition, search space size and new instances. In this chapter, we propose a scatter search with path relinking for solving the Het-FSSP. First, Section 9.1 introduces the basic concepts of scatter search and path relinking. Section 9.2 describes the proposed heuristic and its components. Section 9.3 presents computational experiments. Finally, we analyze and discuss the results and conclude in Section 9.4. This chapter corresponds to the publication of Benavides, Ritt & Miralles (2014a). The tested instances as well as the detailed computational results of this chapter can be obtained at <http://inf.ufrgs.br/algopt/hetFS>.

9.1 Scatter search and path relinking

Scatter search is a meta-heuristic that explores the search space by systematically selecting, combining and evolving a set of reference solutions. It was originally proposed by Glover (1977). A good overview of scatter search can be found in Laguna & Martí (2003). A basic scatter search procedure has five main elements:

1. A *diversification generation method* that produces diverse solutions for the initial pool.
2. An *improvement method* that is optionally applied to enhance each solution in the pool.
3. A *reference set update method* that builds and maintains a *reference set* of solutions. The solutions of the pool are included into the reference set for their good quality or their high diversity.

4. A *subset generation method* that forms subsets of reference solutions that will be combined.

5. A *solution combination method* that generates new solutions using the solutions in each subset. The new solutions form a new pool and the next iteration starts again with the improvement method.

Scatter search may use path relinking as a solution combination method. Path relinking is a strategy that explores trajectories between two solutions. One solution serves as the starting point of the trajectory, and the other as a *guiding solution*. The trajectory between them is explored by repeatedly applying to the current solution the best local modification that makes it more similar to the guiding solution. Formally, if $N(s)$ is the set of neighbours of the current solution s, and $d(s, s')$ is a distance measure between solution s and the guiding solution s', then the path relinking explores the directed neighbourhood $D(s) = \{s'' \in N(s) | d(s'', s') < d(s, s')\}$. Path relinking is called *forward* when the better of the two solutions is the starting solution, and *backward* otherwise. For a good review of scatter search with path relinking see Resende et al. (2010).

9.2 A scatter search with path relinking for the Het-FSSP

An overview of the scatter search procedure that we propose is shown in Algorithm 9.1. It maintains a reference set of high quality solutions as well as a reference set of diverse solutions. Pairs of solutions from these two sets will be combined using a positional combination method for the worker permutations, and a path relinking for the sequence of operations. A local search improves the new solutions. The scatter search stops after a fixed number of iterations or after a time limit is exceeded. Next, we explain the individual components of the proposed scatter search in detail.

9.2.1 Solution representation

A solution of the Het-FSSP consists of an assignment of workers to machines, and the sequence of the operations on each machine. A solution will be represented by $S = (\rho, \pi_1, \ldots, \pi_m)$, where ρ is a permutation of the workers which defines the assignment to the machines, and π_i is the permutation of the operations on machine M_i.

Algorithm 9.1 A heuristic based on scatter search for the Het-FSSP.

```
Input: Initial pool size $t$.
Output: Best found solution $S_{best}$.
 1: Create an initial pool $T$ of $t$ solutions.
 2: repeat
 3:     Update reference sets $RS_1$ and $RS_2$ with elements of $T$ (Algorithm 9.3)
 4:     $T \leftarrow \emptyset$
 5:     for $\forall S_1 \in RS_1$ do
 6:         for $\forall S_2 \in RS_2$ do
 7:             combine solutions $S_1$ and $S_2$ into solution $S_{new}$
 8:             apply local search to $S_{new}$
 9:             $T \leftarrow T \cup \{S_{new}\}$
10:             update $S_{best}$ with $S_{new}$ if necessary
11:         end for
12:     end for
13: until stopping criterion is satisfied
14: return $S_{best}$
```

9.2.2 Diversification generation method

Our diversification method is a randomized constructive heuristic shown in Algorithm 9.2. First, it generates a feasible random assignment of the workers to the machines, and a random permutation π_o of the jobs. Then, starting from an empty schedule, it iteratively inserts the next job from π_o at the position that minimizes the makespan of the partial schedule, until a permutation schedule is complete. The permutation schedule is replicated for every machine, and together with the worker assignment, they form a complete solution for the Het-FSSP. This method produces permutation schedules that form the initial pool. Note that, after the assignment of

Algorithm 9.2 Randomized constructive heuristic for Het-FSSP.

```
Input: The processing times $p_{ijw}$ for each job $J_j$ on each machine $M_m$
        when worker $w$ is assigned to that machine.
Output: A complete solution $S$.
 1: function RANDOMIZED_CONSTRUCTIVE_HEURISTIC( )
 2:     $\rho :=$ feasible random assignment of workers
 3:     $\pi_o := (\pi_o(1), \ldots, \pi_o(n))$ with a random order
 4:     $\pi := (\pi_o(1))$
 5:     for $\pi_o(j), j \in [2, n]$ do
 6:         evaluate all the insertion positions of job $\pi_o(j)$ into $\pi$
 7:         insert job $\pi_o(j)$ into $\pi$ at the position which minimizes $C_{\max}$
 8:     end for
 9:     $\pi_j := \pi$, $j \in [m]$
10:     return $S := (\rho, \pi_1, \ldots, \pi_m)$
11: end function
```

workers and the initial random priority order, the job insertion method is the same of the NEH heuristic from Algorithm 3.1.

9.2.3 Improvement method

Our improvement method is a local search that explores the reduced neighbourhood proposed by Nowicki & Smutnicki (1996) explained in Section 3.3.1. The local search iteratively chooses the best neighbour to replace the current solution, and it stops when there are no better solutions in the neighbourhood. The solution obtained after local search is inserted into the pool T to later update the reference sets.

9.2.4 Reference set update method

Our scatter search maintains two reference sets RS_1 and RS_2. Reference set RS_1 contains high quality solutions of a certain diversity, and reference set RS_2 contains more diverse solutions of lower quality. Diversity is guaranteed by maintaining a minimum distance between the solutions.

The distance of the solutions is defined as the sum of the Kendall-tau distances of the permutations of the operations for each machine, plus the number of workers that have been assigned to different machines. Formally, the Kendall-tau distance between two n-permutations π and π' is defined as

$$d(\pi, \pi') = \sum_{1 \le i < j \le n} [\pi(i) < \pi(j) \text{ and } \pi'(i) > \pi'(j)]. \tag{9.1}$$

Consequently, the distance between two solutions $S = (\rho, \pi_1, \ldots, \pi_m)$ and $S' = (\rho', \pi'_1, \ldots, \pi'_m)$ is defined as

$$d(S, S') = \sum_{1 \le k \le m} d(\pi_k, \pi'_k) + [\rho(k) \neq \rho'(k)]. \tag{9.2}$$

To guarantee diverse solutions in the reference sets, RS_1 only admits solutions with a distance of at least d_1 to all solutions in the set. Similarly, a solution may be added to RS_2 only if it has a distance of at least d_2 to the current solutions in $RS_1 \cup RS_2$. A higher diversity of solutions in RS_2 is achieved by setting $d_2 > d_1$.

The reference sets are continually rebuilt with solutions from the current solution pool T. The reference set update procedure is shown in Algorithm 9.3. It constructs the two reference sets from scratch by extracting solutions from the solution pool. Reference set RS_1 is constructed by adding the solutions from the pool T in order of non-increasing quality, when they satisfy the diversity criterion explained above.

Algorithm 9.3 Update reference sets.

```
Input: Solution pool $T$, reference set sizes $rs_1$, $rs_2$, minimum distances $d_1$, $d_2$.
Output: Reference sets $RS_1$, $RS_2$.
 1: $RS_1 \leftarrow \{$ best element of $T\}$; $T \leftarrow T \setminus RS_1$.
 2: while $|RS_1| < rs_1$ and $|T| > 0$ do
 3:     $S_c \leftarrow$ best element of $T$; $T \leftarrow T \setminus \{S_c\}$.
 4:     if $\min_{r \in RS_1} d(S_c, r) \geq d_1$ then
 5:         $RS_1 \leftarrow RS_1 \cup \{S_c\}$
 6:     end if
 7: end while
 8: $RS_2 \leftarrow \emptyset$.
 9: while $|RS_2| < rs_2$ and $|T| > 0$ do
10:     $S_c \leftarrow$ best element of $T$; $T \leftarrow T \setminus \{S_c\}$.
11:     if $\min_{r \in RS_1 \cup RS_2} d(S_c, r) \geq d_2$ then
12:         $RS_2 \leftarrow RS_2 \cup \{S_c\}$
13:     end if
14: end while
15: fill $RS_2$ with diverse generated solutions, if necessary
16: return $RS_1, RS_2$
```

A solution which has less distance than required is discarded. Reference set RS_2 is constructed in the same manner using the remaining solutions in the pool. If the solution pool is exhausted during the construction, the missing elements of reference set RS_2 are completed by new solutions produced with the diversification generation method. The new solutions are not required to satisfy the diversity criterion. This guarantees that the construction of the reference sets is always successful.

9.2.5 Subset selection and solution combination with path relinking

A new solution for the pool is created by combining solution subsets from the reference sets. Each subset has one solution of both reference sets RS_1 and RS_2. The two solutions are combined in two steps: The first step is to reassign workers, and the second step is to combine the schedules with path relinking while maintaining the new worker assignment.

We use a positional combination method to create a new worker assignment ρ'' from the two worker permutations ρ and ρ'. An example of the positional combination is given in Figure 9.1. First, a worker that has been assigned to the same machine in both permutations will also be assigned to this machine in the new permutation ρ''. Next, we attempt to assign to each free machine a worker assigned to the same machine randomly from either ρ or ρ'. If the chosen worker has not been assigned yet, he is assigned in ρ'', otherwise the machine remains free. The workers assigned in

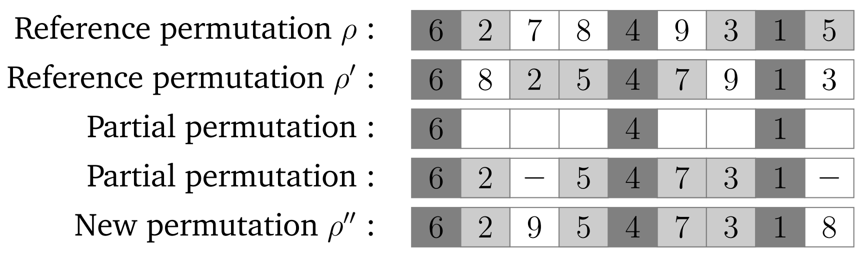

Reference permutation ρ : 6 2 7 8 4 9 3 1 5
Reference permutation ρ' : 6 8 2 5 4 7 9 1 3
Partial permutation : 6 4 1
Partial permutation : 6 2 – 5 4 7 3 1 –
New permutation ρ'' : 6 2 9 5 4 7 3 1 8

Figure 9.1: Example of positional combination with preservation of common elements. Common elements are highlighted in light grey. Elements in the same position randomly chosen from one of the two reference solutions in dark grey. Other elements are randomly assigned.

the resulting partial permutation are in the same positions than in at least one of the reference permutations. Finally, the yet unassigned workers are assigned randomly to the remaining free machines in ρ''.

Our path relinking explores the trajectory between two solutions. The solution from RS_1 serves as the starting point, and the solution from RS_2 is the guiding solution. Path relinking uses the new worker assignment to evaluate all the schedules. The neighbourhood of a solution consists of all solutions that can be obtained by swapping the order of two consecutive operations on the critical path, whose processing order is inverted in the guiding solution. In each step the current solution is substituted by its best neighbour. Path relinking may swap any two consecutive operations in the critical path, even when it does not necessarily improves the current solution, because it must get closer to the guiding solution. The exploration finishes when there are no more possible swaps in the critical path that get closer to the guiding solution. Finally, the best solution found on the explored trajectory is improved with a local search, and included into the pool as the newly combined solution.

9.3 Computational Experiments

9.3.1 Experimental methodology

The proposed scatter search was implemented in C++, and compiled with GNU C++ with optimization level 2 (-O2). All computational tests were executed on a PC with an AMD Opteron 6238 processor running at 2.9 GHz, and with 64 GB of main memory.

We use the following parameters for scatter search: an initial pool size of $t = 60$, minimum distances $d_1 = 10$ and $d_2 = 20$, and reference set sizes $rs_1 = 6$ and

$rs_2 = 4$. Those parameters were the best in preliminary tests on 12 Taillard instances $(1, 6, \ldots, 56)$ and all 6 combinations of incompatibilities and processing time variations. They have been obtained by varying each distance independently in $\{5, 10, \ldots, 30\}$ and each reference set size in $\{2, 4, \ldots, 10\}$.

We also tested forward and backward path relinking strategies. Both strategies produced very similar results. We chose backward path relinking, since it performed with an average relative deviation of 1.5% from the best known value slightly better than forward path relinking with a value of 1.6%.

Scatter search was given a time limit of ten minutes as a stopping criterion. Since scatter search makes random choices when initializing and updating the reference sets, we replicated each experiment 50 times and report averages. We compare the solutions obtained by scatter search to those obtained by solving the mathematical model of Section 8.2.1. The mathematical model has been solved with CPLEX 12.4 running with a single thread and a time limit of one hour.

For the comparison, we use the instances created for the Het-FSSP that were described in Section 8.4, based on the well-known flow shop instances proposed by Carlier (1978) and Taillard (1993).

9.3.2 Numerical results

Tables 9.1–9.4 present results for the CPLEX solver with the proposed model and for the scatter search. Tables 9.1 and 9.3 show results for Carlier instances, and Tables 9.2 and 9.4 for the Taillard instances of 20 or 50 jobs. Tables 9.1 and 9.2 present the results for instances with processing times in $[p, 2p]$, and Tables 9.3 and 9.4 for instances with processing times in $[p, 5p]$. All tables report the size of the instances $(n \times m)$, the best known upper bound ub, the relative deviation in percent of a solution with makespan $C_{\max}$ from the best known solution $\%R = 100\% \times (C_{\max} - ub)/ub$, the percentage relative gap ($gap\%$) between the lower and upper bounds found by CPLEX, the average time $\bar{t}$ that scatter search takes to reach the best value of each test, and the percentage of tests where scatter search reached the best known solution ($\%ub$). Tables 9.1 and 9.3 also present in the last two columns the results for solving the scheduling and assignment problems separately, we explain these results in the last paragraph of this section. The presented values are also presented as averages when grouped by instances or replications, e.g., $\overline{ub}$, $\overline{gap\%}$, $\overline{\%R}$.

CPLEX only solves optimally the instances $car7i00$, $car7i10$, and $car7i20$ in 333, 195 and 133 seconds, respectively. In the remaining instances CPLEX does not reach optimal solutions within an hour. The gap between the lower and the upper bound

Table 9.1: Results for the Carlier instances with processing times between in $[p, 2p]$.

			CPLEX		SS+PR			WA-SS	SS-WA
Instance	n×m	ub	$\%R$	$gap\%$	$\bar{t}$	$\overline{\%R}$	$\%_{ub}$	$\overline{\%R}$	$\overline{\%R}$
car1i00	11×5	9952	0.00	9.6	0.0	0.00	100	6.20	2.58
car1i10	11×5	9952	0.00	12.3	0.0	0.00	100	6.20	2.58
car1i20	11×5	9952	0.00	11.9	0.0	0.00	100	0.00	2.58
car2i00	13×4	10224	0.00	23.6	0.2	0.00	100	6.53	5.09
car2i10	13×4	10224	0.00	27.0	0.1	0.00	100	6.53	5.09
car2i20	13×4	10398	0.00	27.6	0.2	0.00	100	4.75	3.70
car3i00	12×5	10268	2.21	25.4	1.9	0.00	100	0.89	4.54
car3i10	12×5	10359	0.37	23.9	1.5	0.00	100	0.00	3.62
car3i20	12×5	10359	0.00	18.7	1.5	0.00	100	0.00	3.62
car4i00	14×4	11613	0.00	40.5	0.0	0.00	100	0.00	3.06
car4i10	14×4	11846	0.70	41.2	0.1	0.00	100	2.25	1.03
car4i20	14×4	11876	0.21	41.7	0.4	0.00	100	1.99	1.20
car5i00	10×6	10589	2.05	15.4	122.2	0.00	100	1.98	7.79
car5i10	10×6	10589	2.63	17.3	11.1	0.00	100	3.35	10.16
car5i20	10×6	10848	1.13	11.3	0.3	0.00	100	2.71	8.43
car6i00	8×9	11044	1.80	11.3	6.0	0.00	100	0.02	1.58
car6i10	8×9	11044	4.21	12.4	3.0	0.00	100	3.00	2.30
car6i20	8×9	11118	1.74	11.9	172.0	0.10	84	4.12	2.20
car7i00	7×7	8558	0.00	0.0	14.2	0.00	100	1.90	2.77
car7i10	7×7	8558	0.00	0.0	6.6	0.00	100	1.90	2.77
car7i20	7×7	8558	0.00	0.0	6.8	0.00	100	1.90	3.25
car8i00	8×8	11533	2.02	8.7	41.4	0.00	100	4.12	0.62
car8i10	8×8	11659	0.79	7.0	163.4	0.00	100	2.99	1.36
car8i20	8×8	11742	0.00	8.3	3.3	0.00	100	2.27	0.89
Averages			0.83	17.0	23.2	0.00	99	2.73	3.45

Table 9.2: Results for the Taillard instances with processing times in $[p, 2p]$.

			CPLEX		SS+PR		
Instance	n× m	$\overline{ub}$	$\overline{\%R}$	$\overline{gap\%}$	$\bar{t}$	$\overline{\%R}$	$\%_{ub}$
ta001i00-ta010i00	20× 5	1727.7	5.70	60.7	96.7	0.15	57
ta001i10-ta010i10	20× 5	1731.4	6.58	60.9	85.9	0.06	75
ta001i20-ta010i20	20× 5	1741	7.24	61.9	75.2	0.03	74
ta011i00-ta020i00	20×10	2086.3	16.49	61.1	310.1	0.65	8
ta011i10-ta020i10	20×10	2091	17.81	61.4	297.1	0.68	4
ta011i20-ta020i20	20×10	2101	17.56	60.9	314.4	0.88	6
ta021i00-ta030i00	20×20	3047.8	43.09	63.6	329.6	1.35	1
ta021i10-ta030i10	20×20	3039.4	43.20	63.6	322.0	1.40	2
ta021i20-ta030i20	20×20	3037.6	43.02	63.3	338.4	1.25	2
ta031i00-ta040i00	50× 5	3993.4	28.89	86.9	224.0	0.35	14
ta031i10-ta040i10	50× 5	3995.8	35.99	87.7	231.8	0.37	11
ta031i20-ta040i20	50× 5	4007.6	44.50	87.9	244.1	0.35	10
ta041i00-ta050i00	50×10	4352.8	203.73*	92.2*	351.9	1.87	1
ta041i10-ta050i10	50×10	4356	1175.55*	92.6*	353.5	2.00	0
ta041i20-ta050i20	50×10	4356.2	987.68*	92.9*	344.2	2.13	0
ta051i00-ta060i00	50×20	5567.6	–	–	361.6	2.10	1
ta051i10-ta060i10	50×20	5563.1	–	–	378.2	2.13	2
ta051i20-ta060i20	50×20	5565.2	–	–	374.0	2.14	2
Averages			178.47*	73.2*	279.6	1.11	15

* Average only over instances that found solutions.

Table 9.3: Results for the Carlier instances with processing times in $[p, 5p]$.

			CPLEX		SS+PR			WA-SS	SS-WA
Instance	n×m	ub	$\%R$	$gap\%$	$\bar{t}$	$\overline{\%R}$	$\%_{ub}$	$\overline{\%R}$	$\overline{\%R}$
car1I00	11×5	19508	0.00	21.7	0.0	0.00	100	0.00	1.60
car1I10	11×5	19831	0.00	22.4	0.0	0.00	100	1.26	6.31
car1I20	11×5	19831	0.00	11.1	0.0	0.00	100	1.26	6.77
car2I00	13×4	19876	0.01	27.7	1.0	0.00	100	6.98	5.59
car2I10	13×4	19876	1.35	26.4	0.6	0.00	100	6.98	5.59
car2I20	13×4	19876	0.00	30.2	0.4	0.00	100	6.98	5.59
car3I00	12×5	19498	0.29	23.7	140.2	0.10	2	8.70	7.15
car3I10	12×5	19498	0.29	25.1	158.5	0.07	2	8.70	7.15
car3I20	12×5	19498	0.29	23.1	201.9	0.03	90	8.70	12.38
car4I00	14×4	20381	0.00	29.0	0.0	0.00	100	0.00	8.45
car4I10	14×4	22270	0.00	32.9	0.2	0.00	100	1.03	9.27
car4I20	14×4	22270	6.21	26.0	0.2	0.00	100	1.03	9.27
car5I00	10×6	18693	4.46	19.0	0.5	0.00	100	0.00	7.68
car5I10	10×6	18693	0.12	14.6	0.4	0.00	100	0.00	7.68
car5I20	10×6	19468	4.71	22.2	0.9	0.00	100	0.00	10.74
car6I00	8×9	20070	4.14	18.6	34.7	0.00	100	12.87	6.81
car6I10	8×9	20070	2.82	15.4	14.8	0.00	100	12.87	6.81
car6I20	8×9	20070	0.34	11.2	4.3	0.00	100	5.35	13.88
car7I00	7×7	16423	0.00	11.8	0.8	0.00	100	3.28	5.85
car7I10	7×7	17067	0.00	6.6	4.3	0.00	100	2.60	7.72
car7I20	7×7	17257	0.16	9.5	0.9	0.00	100	1.65	7.87
car8I00	8×8	19159	3.84	19.3	1.3	0.00	100	8.27	7.46
car8I10	8×8	19159	0.92	11.7	1.7	0.00	100	8.27	7.46
car8I20	8×8	19791	2.67	16.1	216.2	1.17	18	0.00	10.75
Averages			1.36	19.8	32.7	0.06	88	4.45	7.74

Table 9.4: Results for the Taillard instances with processing times in $[p, 5p]$.

			CPLEX		SS+PR		
Instance	n× m	$\overline{ub}$	$\overline{\%R}$	$\overline{gap\%}$	$\bar{t}$	$\overline{\%R}$	$\%_{ub}$
ta001I00-ta010I00	20× 5	3296.6	6.36	62.1	136.9	0.29	49
ta001I10-ta010I10	20× 5	3311.7	7.49	62.2	133.2	0.18	56
ta001I20-ta010I20	20× 5	3343.7	6.64	61.1	122.7	0.22	41
ta011I00-ta020I00	20×10	3917.2	20.35	65.4	314.7	1.46	2
ta011I10-ta020I10	20×10	3944.3	21.09	65.3	316.3	1.38	2
ta011I20-ta020I20	20×10	3995	19.97	64.4	305.3	1.49	2
ta021I00-ta030I00	20×20	5742.8	123.55	76.6	317.1	2.48	2
ta021I10-ta030I10	20×20	5756.7	81.74	71.2	302.0	2.41	2
ta021I20-ta030I20	20×20	5773.7	64.91	70.5	314.4	2.14	2
ta031I00-ta040I00	50× 5	7853.4	47.41	89.2	274.7	0.52	20
ta031I10-ta040I10	50× 5	7860.2	46.40	89.0	281.9	0.47	17
ta031I20-ta040I20	50× 5	7875.9	47.91	88.0	279.7	0.65	29
ta041I00-ta050I00	50×10	8445.1	227.59*	93.3*	305.8	4.85	0
ta041I10-ta050I10	50×10	8449.3	181.23*	91.7*	297.5	5.00	0
ta041I20-ta050I20	50×10	8488.1	205.74*	92.7*	284.2	4.75	0
ta051I00-ta060I00	50×20	11120	–	–	302.6	4.18	1
ta051I10-ta060I10	50×20	11151.4	–	–	290.6	4.01	1
ta051I20-ta060I20	50×20	11152.5	–	–	270.5	4.12	0
Averages			73.89*	76.2*	269.5	2.26	13

* Average only over instances that found solutions.

ranges from 6 to 42 percent in the remaining Carlier instances that are not solved optimally. CPLEX finds solutions for all the Carlier instances, and finds the best known solution in 20 of the 48 instances.

For the Taillard instances, the gap increases with the size of the instance from 60 to 94 percent. CPLEX finds solutions for all the Taillard instances up to size 50×5, but only for 36 of the 60 instances of size 50×10, and it is not able to solve any of the Taillard instances greater than 50×20 within the time limit. Different from the FSSP, even the small Carlier instances of Het-FSSP cannot be solved optimally within one hour. This is expected since the number of possible solutions is a factor of $m!$ larger due to the additional worker assignment.

Scatter search in ten minutes reaches better solutions than CPLEX in one hour for all the tested instances. All best known values reported in the tables have been found by scatter search. In 43 of the 48 Carlier instances scatter search finds the same best makespan in all replications. For the remaining Carlier instances the average relative deviation from the best known value is less than 1.2%. All the best values were found in less than four minutes.

Scatter search also finds solutions for all the Taillard instances. Since the instances are substantially larger than the Carlier instances, the time to find the best value is larger, and varies from about one to six minutes with increasing size of the instance. Scatter search also exhibits more variation in the quality of the solutions for the Taillard instances: the average relative deviation reaches up to 5%, with larger gaps for increasing instance size, and the best known value is found in 20% of the replications.

To further evaluate the performance of the scatter search, we tested it on the first 60 homogeneous non-permutation FSSP instances of Taillard. Table 9.5 shows the best known value (BKV), the best solution found (BS), and the average relative percentage deviation $\overline{\%R}$ over the best known value of the non-permutation FSSP. These best known values of the non-permutation FSSP are the smaller than the best known values for the permutation FSSP from Taillard (2004), and than the results from Vaessens (1996). Our method finds 12 new best solutions (in bold) and has an average relative deviation of 1.2% in a time comparable to other approaches (LIN; YING, 2009; ROSSI; LANZETTA, 2013b). Given that the method was designed for a different problem, this indicates that it is reasonably effective in finding solutions of good quality.

The numerical results also indicate that the difficulty of solving the problem depends mainly on the instance size, and to a much lesser extent on the processing time variation and the percentage of incompatibilities. The average relative deviation from the best known value for instances with processing time in $[p, 2p]$, compared

Table 9.5: Results for the Taillard instances of the original non-permutation FSSP.

	BKV	BS	$\overline{\%R}$		BKV	BS	$\overline{\%R}$		BKV	BS	$\overline{\%R}$
	1278	1278	0.00		1560	1569	1.52		2293	**2255**	-0.84
	1358	1358	0.00		1644	1654	1.59		2092	**2070**	-0.40
	1073	1073	0.00		1486	**1476**	1.04		2313	**2294**	0.19
	1292	1293	0.08		1368	1375	1.30		2223	**2188**	-1.03
20×5	1231	1235	0.32	20×10	1413	**1391**	-0.04	20×20	2291	**2257**	-0.73
	1193	1193	0.13		1369	1369	1.70		2221	**2160**	-1.55
	1234	1234	0.31		1428	1436	1.60		2267	**2231**	-0.40
	1199	1199	0.00		1527	1532	0.96		2183	**2164**	0.11
	1210	1210	0.00		1586	1590	0.96		2227	**2201**	-0.55
	1103	1103	0.00		1559	1575	2.54		2178	**2128**	-1.07
	2724	2724	0.00		2991	3051	2.55		3850	3949	3.33
	2834	2838	0.26		2867	2923	2.64		3704	3795	3.42
	2612	2620	0.54		2832	2902	3.27		3640	3745	3.71
	2751	2751	0.06		3063	3085	1.35		3723	3801	3.32
50×5	2853	2854	0.13	50×10	2976	3016	2.14	50×20	3611	3690	3.42
	2825	2828	0.27		2991	3052	2.37		3681	3783	3.23
	2716	2719	0.68		3093	3133	1.87		3704	3816	3.73
	2683	2683	0.60		3026	3061	1.52		3691	3801	3.58
	2545	2550	0.75		2887	2933	2.34		3743	3818	3.21
	2776	2776	0.00		3065	3121	2.56		3756	3821	2.91

to instances with processing time in $[p, 5p]$, differs less than 3%, and less than 0.5% between instances with a different percentage of incompatibilities.

Finally, we were interested in assessing the benefit of solving the combined workforce assignment and scheduling problem, compared to simpler approaches. To this aim, we focused on the Carlier instances that can be solved optimally with the branch and bound method of Brucker, Jurisch & Sievers (1994) in short time. We computed two solutions: one (column "SS-WA") that first finds the best homogeneous non-permutation schedule, and then assigns the workers such that the makespan of this schedule is minimized, and another (column "WA-SS") that first finds the worker assignment that minimizes the total processing time of the jobs, and then determines the best non-permutation schedule. The makespan of the first solution is in average 2.7% (4.4%) longer than the best known values of the Het-FSSP for processing times chosen in the interval $[p, 2p]$ $([p, 5p])$. The average increase is 3.4% (7.7%) for the second solution. This comparison shows that, while the variations of the processing times due to the disabilities of the workers will obviously increase the makespan, the solution of the combined problem is able to compensate a part of it, making the production process more efficient than solving the subproblems separately. The improvement over other ad hoc methods for scheduling heterogeneous workers, e.g.

determining a flow shop schedule for the worst case processing times, is likely to be even worse.

9.4 Concluding remarks

This chapter proposed a scatter search with path relinking to solve the Het-FSSP. To solve it, one must find an optimal assignment of the workers to the workstations as well as an optimal schedule of the operations, whose processing times depend on the assignment of the workers. The Het-FSSP is considerably harder than the traditional FSSP, and therefore even small instances of its mathematical model cannot be solved by standard software. However, computational tests show that the problem can be solved satisfactorily by scatter search. The tests also indicate that its difficulty depends mainly on the size of the instance, and, to a much lesser extent, to the two variations levels of the processing times or the three levels of worker-machine incompatibilities that were introduced in our instances.

From a practical point of view, our proposal can be especially beneficial when all workers have different execution times, and when the optimal schedule varies depending on the available resources and the workplaces assignment, as it happens in sheltered work centers for disabled. The computational tests show that solving the combined workforce assignment and scheduling problem in such cases has benefits, compared to simpler methods that deal with these problems separately.

But this first approximation to worker heterogeneity within flow shop systems is also crucial for those ordinary companies willing to integrate just a percentage of disabled workers as part of their policies of Corporate Social Responsibility. Our main conclusion is that companies can contribute to integrate people with disabilities in their production systems without important losses in productivity, and a very interesting further research line will be to study in detail the correlation between the degree of workers integration and the supposed productivity decrease. Extending this study to other configurations like open shops and job shops can also be interesting future lines of research.

10 A MULTI-START LOCAL SEARCH FOR THE HET-JSSP

Chapter 8 has introduced the heterogeneous workforce assignment and job shop scheduling problem (or Het-JSSP), also showing its theoretical definition, search space size and new instances. In this chapter, we propose a multi-start local search heuristic for solving the Het-JSSP. First, Section 10.1 describes the multi-start local search heuristic. Section 10.2 presents computational experiments. Finally, we analyze and discuss the results and conclude in Section 10.3. This chapter corresponds to the publication of Benavides, Ritt & Miralles (2014b).

10.1 A multi-start local search for the Het-JSSP

Our multi-start local search heuristic for the Het-JSSP is shown in Algorithm 10.1. At each iteration of the multi-start local search, a construction heuristic generates a different initial solution, that is improved by a local search. The multi-start local search returns the best overall solution that it finds in all iterations. The multi-start local search stops after a fixed number of iterations or after a time limit is exceeded. Multi-start heuristics are efficient for problems where solutions are constructed easily, but their neighbourhoods are difficult to be explored by local search methods. Martí, Resende & Ribeiro (2013) give a detailed description of multi-start methods.

Algorithm 10.1 A multi-start local search for the Het-JSSP.

```
Output: Best solution found $S_{best}$
1: repeat
2:     Create a new solution $S_{new}$
3:     Apply a local search to $S_{new}$
4:     Update $S_{best}$ with $S_{new}$ if necessary
5: until stopping criterion is satisfied
6: return $S_{best}$
```

10.1.1 Construction heuristic

Our construction heuristic is shown in Algorithm 10.2. It first generates a random worker assignment and fixes the processing times of the assigned workers on the machines for the incumbent JSSP. After that, it creates a solution for that JSSP with the *shifting bottleneck procedure* (or SBP). The SBP was proposed by Adams, Balas & Zawack (1988) for the job shop scheduling problem. The SBP schedules iteratively the machine that is considered a bottleneck. To do this, it solves the one-machine sequencing problem with release and due dates that minimizes the maximum lateness for each unscheduled machine, and schedules the machine with the worst maximum lateness, i.e., the bottleneck machine. After each machine is scheduled, the procedure applies a reoptimization over all the previously scheduled machines. The reoptimization consists in trying to reschedule each previously scheduled machine one by one, and updates the partial schedule if there is a better one. This process is repeated until all machines are scheduled. The final schedule is then improved by a local search.

Algorithm 10.2 A construction heuristic for the Het-JSSP.

```
Output: Constructed solution
1: Assign workers randomly
2: repeat
3:     for all unscheduled machines do
4:         Solve the one-machine sequencing problem
5:     end for
6:     Schedule the bottleneck machine
7:     Reoptimize all the previously scheduled machines
8: until all machines are scheduled
9: return constructed solution
```

10.1.2 Local search heuristic

Our improvement method is a local search that explores the reduced neighbourhood proposed by Nowicki & Smutnicki (1996) explained in Section 3.3.1. The local search iteratively chooses the best neighbour to replace the current solution, and it stops when there are no better solutions in the neighbourhood. The best solution over all iterations is saved by the multi-start local search.

10.2 Computational Experiments

10.2.1 Experimental methodology

We compare the solutions obtained by the multi-start local search to those obtained by solving the mathematical model of Section 8.2.2. For the comparison, we use the instances created for the Het-JSSP that were described in Section 8.4, based on the well-known job shop instances of Taillard (1993) and on the instance *ft10* of Fischer & Thompson (1963).

The stopping criterion of the multi-start local search is one thousand iterations. This number of iterations was chosen for being large enough to reach a convergence of the method without running a long time. Our multi-start local search was implemented in C++, and compiled with GNU C++ 4.7.3 with optimization level 2 (-O2). We use the SBP implementation of Applegate & Cook (1991). The mathematical model has been solved with CPLEX 12.5 running with a single thread and a time limit of one hour. All computational tests were executed on a PC with an AMD Opteron 6238 processor running at 2.9 GHz and 64 GB of main memory.

10.2.2 Numerical results

Tables 10.1 and 10.2 present results for the CPLEX solver with the proposed model and for the multi-start local search. Table 10.1 presents results for instances with processing times in $[p, 2p]$, and Table 10.2 for instances with processing times in $[p, 5p]$. Each line of the tables reports averages over 10 instances, grouped by size and percentage of incompatibility.

All tables report the size of the instances ($n \times m$), the average of the best known upper bounds ($\overline{ub}$) obtained by this or by previous preliminary experiments, the average ($\overline{\%R}$) of the relative deviation $\%R = 100\% \times (C_{\max} - ub)/ub$ in percent of a solution with makespan $C_{\max}$ from the best known solution, and the average of the percentage relative gap ($\overline{gap\%}$) between the lower and upper bounds found by CPLEX. For the multi-start local search, we present the average time $\bar{t}$ in seconds, the average deviation for the best solution found ($\overline{\%R}$) and for all the one thousand independent solutions ($\overline{\%R_I}$) obtained by each iteration of construction and local search.

CPLEX was able to find solutions for all the instances, but it was not able to prove the optimality of any of them within the time limit of one hour. CPLEX finds the best known upper bound in 100 of the Taillard instances, 51 among the 60 instances of 15 jobs, 49 among the 120 instances of 20 jobs, and none of the 120 instances of 30 jobs. The relative deviation $\overline{\%R}$ increases with the size of the instance from 0.06%

Table 10.1: Average results for the Taillard instances with processing times in $[p, 2p]$.

			CPLEX		Multi-start		
Instances	n× m	$\overline{ub}$	$\overline{\%R}$	$\overline{gap\%}$	$\bar{t}$	$\overline{\%R}$	$\overline{\%R_I}$
tai01i00 - tai10i00	15×15	1783.2	0.26	30.22	61.9	2.22	12.34
tai01i10 - tai10i10	15×15	1794.8	0.07	30.25	61.3	1.70	11.61
tai01i20 - tai10i20	15×15	1797.8	0.76	30.58	62.0	1.80	11.33
tai11i00 - tai20i00	20×15	2095.1	2.77	42.37	112.8	0.18	8.82
tai11i10 - tai20i10	20×15	2088.9	3.79	42.18	112.5	0.19	9.25
tai11i20 - tai20i20	20×15	2096.2	3.80	41.93	114.4	0.00	8.96
tai21i00 - tai30i00	20×20	2494.1	1.65	38.15	256.9	0.64	8.05
tai21i10 - tai30i10	20×20	2496.6	2.20	38.25	253.8	0.40	7.97
tai21i20 - tai30i20	20×20	2494.5	2.18	37.93	254.6	0.12	8.07
tai31i00 - tai40i00	30×15	2769.9	16.80	60.35	298.5	0.00	7.78
tai31i10 - tai40i10	30×15	2771.7	16.16	59.92	299.6	0.00	7.65
tai31i20 - tai40i20	30×15	2786.0	17.14	60.29	298.6	0.00	7.14
tai41i00 - tai50i00	30×20	3119.3	26.81	59.39	649.1	0.00	7.84
tai41i10 - tai50i10	30×20	3128.7	21.42	57.69	639.2	0.00	7.46
tai41i20 - tai50i20	30×20	3125.6	20.12	56.94	633.5	0.00	7.56
Averages			9.06	45.76		0.48	8.79

Table 10.2: Average results for the Taillard instances with processing times in $[p, 5p]$.

			CPLEX		Multi-start		
Instances	n× m	$\overline{ub}$	$\overline{\%R}$	$\overline{gap\%}$	$\bar{t}$	$\overline{\%R}$	$\overline{\%R_I}$
tai01I00 - tai10I00	15×15	3407.9	0.19	35.83	62.7	5.86	20.65
tai01I10 - tai10I10	15×15	3396.6	0.41	35.22	62.4	6.33	21.35
tai01I20 - tai10I20	15×15	3448.8	0.06	34.91	62.2	4.75	19.47
tai11I00 - tai20I00	20×15	4028.4	1.27	46.99	115.0	1.72	15.49
tai11I10 - tai20I10	20×15	4029.7	1.56	46.61	117.0	1.66	15.45
tai11I20 - tai20I20	20×15	4065.5	2.48	46.59	116.7	1.03	14.48
tai21I00 - tai30I00	20×20	4766.4	1.59	42.53	261.8	3.31	15.13
tai21I10 - tai30I10	20×20	4802.1	0.70	41.74	270.2	2.95	14.37
tai21I20 - tai30I20	20×20	4822.4	0.48	41.30	269.7	2.74	13.86
tai31I00 - tai40I00	30×15	5445.2	15.38	63.47	302.8	0.00	10.83
tai31I10 - tai40I10	30×15	5428.1	16.42	63.40	303.9	0.00	10.94
tai31I20 - tai40I20	30×15	5407.4	15.06	62.32	304.5	0.00	11.31
tai41I00 - tai50I00	30×20	6186.2	23.13	62.09	711.6	0.00	10.06
tai41I10 - tai50I10	30×20	6208.6	19.69	60.83	682.1	0.00	9.70
tai41I20 - tai50I20	30×20	6178.2	21.66	60.91	711.9	0.00	10.39
Averages			8.00	49.65		2.02	14.23

to 27%, and the gap between the CPLEX lower and upper bound increases with the size of the instance from 30% to 61%. The average relative deviation from the best known value for instances with processing time in $[p, 2p]$, compared to instances with processing time in $[p, 5p]$, differs about 1%, and less than 0.8% between instances with a different percentage of incompatibilities. This indicates that the difficulty of solving the problem depends mainly on the instance size, and to a much lesser extent on the introduced variations on the processing times or the percentage of incompatibilities.

Even the small Het-JSSP instance based on *ft10* cannot be solved optimally by CPLEX within the time limit of one hour, returning a gap of 21% after that time. Considering that the optimal value for the original *ft10* JSSP instance is found in less than 10 minutes, and its optimality is proven in less than 15 minutes, this indicates that the Het-JSSP is considerably harder than the normal JSSP. This is expected since the Het-JSSP extends the JSSP with the additional worker assignment.

The multi-start local search reaches the best known values for 200 instances, all the 120 instances of 30 jobs, 71 among the 120 instances of 20 jobs, and 9 among the 90 instances of 15. The processing time of the multi-start local search increases from 1 to 12 minutes with the size of the instance. The average deviation of the best found solution $\overline{\%R}$ for the instances with processing times in $[p, 2p]$ (or $[p, 5p]$) decreases from 2.3% to 0% (or from 6.4% to 0%), and the average deviation of all the created solutions $\overline{\%R_I}$ decreases from 12.4% to 7.1% (or from 21.4% to 9.7%).

Our simple multi-start local search is effective enough to find solutions of good quality that are in average 7% better than the produced by the exact method, and in less than a fifth of the time. Furthermore, one iteration of construction and local search produces better results than CPLEX for the instances of 30 jobs in less than one second. The local search gives a relative improvement of less than 0.7% to the constructed solutions, but also the average time that it consumes is less than 4% of the total running time. This indicates that, although a local search is necessary to guarantee that the final solution is a local minimum, the constructive heuristic alone is thorough and creates solutions of good quality.

Finally, the largest variation between our upper bounds and the expected increased makespan obtained by multiplying the best known values of the original JSSP by a factor of 1.5 for instances with processing time in $[p, 2p]$ (and by a factor of 3 for instances with processing time in $[p, 5p]$) is less than 7%. Thus, the disabilities and different capabilities of the workers are compensated effectively without a great increase in the makespan.

10.3 Concluding remarks

In this chapter, we proposed a multi-start local search to solve the Het-JSSP. To solve it, one must find an optimal assignment of the workers to the workstations as well as an optimal schedule of the operations, whose processing times depend on the assignment of the workers. The Het-JSSP is considerably harder than a traditional job shop problem. The Het-JSSP mathematical model cannot be solved by standard software even for small instances of size 10×10. However, computational tests show that the problem can be solved satisfactorily by a multi-start local search.

This results confirm that the models and methods for the Het-JSSP and the Het-FSSP may be extended to other scheduling problems. Consequently, our proposal may be beneficial to the operations management area, when all (or some) workers perform their tasks with different execution times, and when both the optimal schedule and the optimal assignment of workers vary depending on each other.

CPLEX finds better solutions for many of the small instances, and our construction method is thorough. Therefore, there is still a need for other more suitable heuristics that may improve the results in comparable reasonable computational times. In future research, we may use a swifter construction method and invest better this time in other local search and diversification methods.

11 RELATED RESEARCH: INCLUDING WORKERS WITH DISABILITIES IN FLOW SHOPS

Chapters 8 and 9 have studied the heterogeneous workforce assignment and flow shop scheduling problem (or Het-FSSP) where there is a heterogeneous worker to be assigned to each workstation (or machine). A natural variation is the particular case of the inclusion of one or two workers with disabilities in one of the stations along with regular workers. This variation was studied jointly with Germano Carniel as the topic of his Master's degree research. This chapter briefly describes the results of his research from the publication of Carniel et al. (2015). The main contributions presented in this chapter are by Germano, and they are presented in this thesis for completeness.

11.1 Inserting a single worker with disabilities into a flow shop

Table 11.1 represents the situation where one worker with disabilities (WWD) must be integrated into a regular workforce in a flow shop. Besides the times that regular workers take to perform the operations, we have particular times for a WWD that may exceed the time of the regular workers. In the example, the times of the WWD were chosen randomly in the interval $[p, 2p]$, for a processing time of p of a regular worker. The WWD may also be unable to operate some of the machines. In the example, this is the case for machine M_4, and it is represented by processing times of ∞ on this machine. The problem of inserting a WWD into a flow shop is defined as follows: we have to assign the WWD to a machine she is able to operate, and find a valid schedule of the jobs, such that the makespan is minimized. Carniel et al. (2015) call the variant restricted to permutation schedules the *Permutation Flow Shop Insertion and Scheduling Problem* (PFSISP).

Figure 11.1 presents an optimal solution for the PFSISP instance in Table 11.1. This solution assigns the WWD to the machine M_3, which becomes a bottleneck, increasing

Table 11.1: An instance of the PFSISP.

	Regular Worker				WWD w			
Job	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4
J_1	1	2	2	1	2	4	2	∞
J_2	1	1	2	2	1	1	4	∞
J_3	2	1	1	2	4	2	1	∞
J_4	1	3	2	1	1	4	2	∞

w: Worker with disabilities.

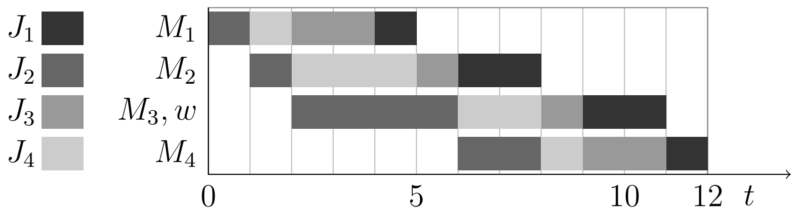


Figure 11.1: Optimal schedule for the PFSISP instance of makespan 12 which assign the WWD to machine M_3. The optimal regular schedule has makespan 11.

Table 11.2: An instance of the HPFSISP.

	Regular Worker				WWD w_1				WWD w_2			
Job	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4
J_1	1	2	2	1	2	4	2	∞	2	3	3	∞
J_2	1	1	2	2	1	1	4	∞	2	2	3	∞
J_3	2	1	1	2	4	2	1	∞	4	1	1	∞
J_4	1	3	2	1	1	4	2	∞	2	5	4	∞

w: Worker with disabilities.

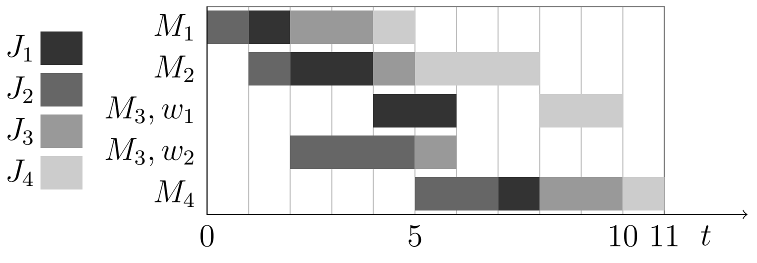


Figure 11.2: Optimal schedule for the HPFSISP instance of makespan 11.

the makespan to 12, while the optimal schedule considering only regular workers has a makespan of 11.

11.2 Inserting two workers into a hybrid flow shop

When assigning a single, slow worker to a machine in a flow shop, she will likely be a bottleneck and increase the makespan. An alternative is to assign two WWDs to a single stage with two parallel machines into a hybrid flow shop. This allows to integrate more WWDs into the production line and has the potential to compensate for their increased execution times. Carniel et al. (2015) refer to this variant as the *Hybrid Permutation Flow Shop Insertion and Scheduling Problem* (HPFSISP). Table 11.2 represents this situation, consisting of a set of processing times for the regular workers, and two sets of processing times for WWDs. Figure 11.2 presents a solution for the HPFSISP. A solution is given by an assignment of the two workers to some stage, and a processing order of the jobs. In the permutation version, the jobs are required to obey the processing order on each of the parallel machines. Note that the parallel stage was able to compensate for the longer processing times of the WWDs and the schedule has makespan 11, as in the regular case.

11.3 A pooled iterated local search for the PFSISP

Carniel et al. (2015) propose a pooled iterated local search for the PFSISP. It initially creates a pool with m candidate solutions, each of which assigns the WWDs to one of the m stages and applies the NEH heuristic to obtain an initial solution. Then, it improves every candidate solution in the pool with an IGA for a fixed time t, to later eliminate the worst solution from the pool. These steps are repeated until there is only one solution left in the pool. The total running time is therefore $\binom{m}{2}t$, and the k-th best solution receives a time $(m+1-k)t$. This ensures that more promising solutions receive more time to keep improving.

11.4 Solving the two-machine subproblem

A solution of the HPFSISP assigns two workers to a stage with two parallel machines, and must additionally solve the subproblem of finding an optimal schedule for this stage. This subproblem can be formulated as a head-body-tail problem on two unrelated machines as follows. For a permutation π of jobs, and an assignment of the WWDs to stage k, define heads r_j as the earliest completion time of job j in the previous stage $k-1$, and tails q_j as the shortest time from the start of job j on the next stage $k+1$ to the completion of the last operation. Then, the problem consists in finding starting times $S_j \geq r_j$ for the jobs $j \in [n]$ on the two machines that minimize $C_{\max} = \max_j(S_j + p_j + q_j)$. Since the order of the permutation flow shop π is imposed on all machines, the problem is reduced to find an optimal assignment of the jobs to the parallel machines. This problem is NP-hard, but can be solved heuristically by a greedy assignment or exactly by dynamic programming in time $O(n\overline{C})$ for some upper bound $\overline{C}$ on the makespan.

11.5 Results and remarks

Carniel et al. (2015) also proposed mathematical models for the PFSISP and the HPFSISP that were tested with CPLEX 12.5 with a time limit of one hour. The PFSISP was also solved with the LOMPEN branch-and-bound algorithm of Companys & Mateo (2007) by assigning the WWD to each stage separately with a time limit of two hours. The proposed heuristics and the mathematical models were compared using new instances based on the benchmarks of Carlier (1978) and Taillard (1993).

The heuristics find close to optimal solutions in a short time for both the PFSISP and the HPFSISP. The different heuristics strategies have only a small impact on solution quality. Carniel et al. (2015) observed that inserting a single worker introduces a large overhead. In contrast, the disabilities of two workers can be concealed when they are assigned to two parallel machines in one stage of the flow shop.

From a practical point of view, when the WWD is about or less than 50% slower than the regular workers, she can be included into a flow shop with a small overhead, and the assignment of two WWDs to a parallel stage with two machines can even reduce the makespan. In summary, the insertions of WWDs are feasible with small impact in the production efficiency. This suggests that companies can contribute to integrate persons with disabilities in their production systems with moderate or without losses in productivity.

Part IV

Concluding remarks and future research

12 CONCLUDING REMARKS AND FUTURE RESEARCH

In this thesis, we described the permutation and the non-permutation flow shop scheduling problem (FSSP), and the theoretical benefit of considering non-permutation schedules, together with other related shop scheduling concepts and models. We explained the assignment problem of heterogeneous workers, and the reasons for considering it with the shop scheduling as a combined problem. We mathematically defined the heterogeneous workforce assignment and flow shop scheduling problem (or Het-FSSP) and the heterogeneous workforce assignment and job shop scheduling problem (or Het-JSSP), we explained their complexity, and presented a new set of instances. We proposed and implemented different heuristic methods to solve the problems, such as iterated local search and iterated greedy algorithms for the permutation and non-permutation FSSP with makespan and total completion time criteria, a scatter search with path relinking for the Het-FSSP, and a multi-start local search for the Het-JSSP. We showed the results obtained by various experiments executed during the research.

The experiments studied different issues such as the quality of the permutation schedules obtained by our methods compared to state-of-the-art methods for the permutation FSSP, the quality of the non-permutation schedules obtained by our methods compared to state-of-the-art methods for the permutation and non-permutation FSSP, the technical differences between the permutation and non-permutation schedules obtained by our methods, the quality of the solutions obtained by our methods for the Het-FSSP and Het-JSSP, the influence of the heterogeneity of workers in the schedules for the Het-FSSP and Het-JSSP.

The two major concluding remarks of this thesis are presented next. First, the consideration of non-permutation schedules to solve the FSSP with the proposed methods is possible, using the same time and effort that state-of-the-art methods use to solve the permutation FSSP, and producing non-permutation schedules with better quality than permutation and non-permutation schedules produced by state-of-the-art methods. Second, the integration of the problem of heterogeneous workers assignment into the shop scheduling problems is possible and advantageous. The Het-FSSP and Het-JSSP can be solved satisfactorily by the proposed methods, and the produced solutions compensate the disabilities and different capabilities of the workers with insignificant or no losses in the productivity objectives.

Next, ideas and suggestions for possible trends for future research are presented, starting with concrete future research and ending with more general ideas.

Bi-objective permutation flow shop scheduling

In this thesis we have presented separated methods for the permutation FSSP with the makespan and the total completion time criteria that are comparable to or better than the state-of-the-art methods for the permutation FSSP. A natural extension of this work is to consider the bi-objective permutation FSSP with makespan and total completion time criteria.

We are already working in this idea. We implemented a hybrid iterated Pareto local search that hybridizes the principles of iterated local search and Pareto local search. It uses a population P to maintain the Pareto frontier or Pareto local optimum set. It starts with a population P that contains two initial solutions π_0 and π_1 that are created by good heuristics for each objective function, such as short runs of our iterated greedy algorithms for makespan and total completion time. Then, the algorithm repeatedly selects an unvisited solution $\pi \in P$ and a normalized weight vector $\vec{\lambda} = (\lambda, 1 - \lambda)$ with $0 \leq \lambda \leq 1$, and iteratively applies scalarized versions of a perturbation scheme and a Pareto local search, such as those of our iterated greedy algorithm described in Section 5.1, with a modification to verify the inclusion of each visited neighbour into the Pareto frontier P. These versions minimize a scalarized single-objective function $f_{\vec{\lambda}}(\pi) = \lambda C_{\max}(\pi) + (1 - \lambda)C_{\text{sum}}(\pi)$. Preliminary tests were executed with instances with 100 jobs and 20 machines of Taillard (1993), that are considered the hardest for both makespan and total completion time criteria. The results are promising when compared to the results of Dubois-Lacoste, López-Ibáñez & Stützle (2011) and Minella, Ruiz & Ciavotta (2011).

Bi-objective non-permutation flow shop scheduling

In this thesis we have presented separated methods for the non-permutation FSSP with the makespan and the total completion time criteria that are based on strategic job reordering, and that dominate the state-of-the-art methods for both the permutation and the non-permutation FSSP. Another natural next step is to extend these results to the bi-objective non-permutation FSSP, including methods that take into account strategic operation reordering in permutation schedules to create better non-permutation schedules.

Tie-breaking mechanisms for non-permutation FSSP

The proposed methods for the non-permutation FSSP used tie-breaking mechanisms that were originally proposed for permutation FSSP, or used random-breaking mechanisms. We plan to study in a near future how different tie-breaking mechanisms influence the results of the constructive heuristics and the iterated greedy algorithms for the non-permutation FSSP with the makespan and the total completion time criteria.

Heterogeneous workers in flow shop scheduling with makespan and total completion time criteria

In this thesis we have presented the heterogeneous workforce assignment and flow shop scheduling problem (or Het-FSSP) where there is a heterogeneous worker to be assigned to each workstation (or machine) and a flow shop schedule must be found with the objective of minimizing the makespan. We also have presented methods for the FSSP with total completion time criterion. The next step is to extend these results to the Het-FSSP with total completion time criterion and to the bi-objective Het-FSSP.

Further research lines

An unexplored research area is the use of exact algorithms such as branch-and-bound to find optimal solutions for the non-permutation flow shop using the results of this research to design dominance rules, lower and upper bounds that easily discard major reorders of jobs in subsequent machines.

Our research may also be expanded by taking into consideration:

- other optimization criteria such as minimizing flowtime, earliness, lateness or tardiness;
- other shop scheduling models such as job shop, open shop and even more general partial order shop models;
- other models for heterogeneous workers, such as categorized (e.g. newbie, regular, expert) or even with proportional models for related execution times;

- other models for insertion of heterogeneous workers in a homogeneous work environment, such as a small percentage of workers and the use of parallel workstations for them, following the example of Chapter 11;
- other specific models for insertion of heterogeneous workers in the regular industry, such as working cells (or groups) in a stage or when workers control subsets of parallel machines in a stage.

REFERENCES

ADAMS, J.; BALAS, E.; ZAWACK, D. The shifting bottleneck procedure for job shop scheduling. *Management science*, INFORMS, v. 34, n. 3, p. 391–401, 1988.

AHMADIZAR, F. A new ant colony algorithm for makespan minimization in permutation flow shops. *Computers & Industrial Engineering*, Elsevier, v. 63, n. 2, p. 355–361, 2012.

AIEX, R. M.; BINATO, S.; RESENDE, M. G. Parallel grasp with path-relinking for job shop scheduling. *Parallel Computing*, Elsevier, v. 29, n. 4, p. 393–430, 2003.

AKHSHABI, M.; TAVAKKOLI-MOGHADDAM, R.; RAHNAMAY-ROODPOSHTI, F. A hybrid particle swarm optimization algorithm for a no-wait flow shop scheduling problem with the total flow time. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 70, n. 5-8, p. 1181–1188, 2014.

AMIRGHASEMI, M.; ZAMANI, R. An effective asexual genetic algorithm for solving the job shop scheduling problem. *Computers & Industrial Engineering*, Elsevier, v. 83, p. 123–138, 2015.

APPLEGATE, D.; COOK, W. A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, INFORMS, v. 3, n. 2, p. 149–156, 1991.

BALAPRAKASH, P.; BIRATTARI, M.; STÜTZLE, T. Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In: BARTZ-BEIELSTEIN, T. et al. (Ed.). *Hybrid Metaheuristics*. Dortmund, Germany: Springer, 2007, (Lecture Notes in Computer Science, v. 4771). p. 108–122.

BANHARNSAKUN, A.; SIRINAOVAKUL, B.; ACHALAKUL, T. Job shop scheduling with the Best-so-far ABC. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 25, n. 3, p. 583–593, 2012.

BARTHOLDI, J. J.; EISENSTEIN, D. D. A production line that balances itself. *Operations Research*, INFORMS, v. 44, n. 1, p. 21–34, 1996.

BEASLEY, J. *ORLib–Operations Research Library*. 2005. Available from Internet: <http://people.brunel.ac.uk/~mastjjb/jeb/info.html>.

BENAVIDES, A. J.; RITT, M. Iterated local search heuristics for minimizing total completion time in permutation and non-permutation flow shops. In: *Twenty-Fifth International Conference on Automated Planning and Scheduling*. Jerusalem, Israel: AAAI Publications, 2015. p. 34–41.

BENAVIDES, A. J.; RITT, M. Two simple and effective heuristics for minimizing the makespan in non-permutation flow shops. *Computers & Operations Research*, Elsevier, v. 66, p. 160–169, 2016.

BENAVIDES, A. J.; RITT, M.; MIRALLES, C. Flow shop scheduling with heterogeneous workers. *European Journal of Operational Research*, Elsevier, v. 237, n. 2, p. 713–720, 2014.

BENAVIDES, A. J.; RITT, M.; MIRALLES, C. Heterogeneous workforce in job shop scheduling. In: *VIII ALIO/EURO Workshop on Applied Combinatorial Optimization*. Montevideo, Uruguay: ALIO/EURO, 2014.

BEZERRA, L. C. T.; LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. Deconstructing multi-objective evolutionary algorithms: An iterative analysis on the permutation flow-shop problem. In: PARDALOS, P. M. et al. (Ed.). *Learning and Intelligent Optimization*. Gainesville, Florida: Springer, 2014, (Lecture Notes in Computer Science, v. 8426). p. 157–172.

BINATO, S. et al. A GRASP for job shop scheduling. In: RIBEIRO, C. C.; P, H. (Ed.). *Essays and Surveys in Metaheuristics*. New York: Springer, 2002, (Operations Research/Computer Science Interfaces Series, v. 15). p. 59–79.

BŁAŻEWICZ, J. et al. *Handbook on scheduling: from theory to applications*. New York: Springer, 2007. (International Handbook on Information Systems).

BLUM, C.; MIRALLES, C. On solving the assembly line worker assignment and balancing problem via beam search. *Computers & Operations Research*, v. 38, n. 1, p. 328 – 339, 2011.

BORBA, L.; RITT, M. A heuristic and a branch-and-bound algorithm for the assembly line worker assignment and balancing problem. *Computers & Operations Research*, Elsevier, v. 45, p. 87–96, 2014.

BRUCKER, P. *Scheduling algorithms*. 4th. ed. New York: Springer, 2004.

BRUCKER, P.; HEITMANN, S.; HURINK, J. Flow-shop problems with intermediate buffers. *OR Spectrum*, Springer, v. 25, n. 4, p. 549–574, 2003.

BRUCKER, P.; JURISCH, B.; SIEVERS, B. A branch and bound algorithm for the job-shop scheduling problem. *Discrete applied mathematics*, Elsevier, v. 49, n. 1, p. 107–127, 1994.

CARLIER, J. Ordonnancements à contraintes disjonctives. *Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, EDP Sciences, v. 12, n. 4, p. 333–350, 1978.

CARNIEL, G. C. et al. Inclusion of workers with disabilities in flow shop scheduling problems. In: *11th IEEE International Conference on Automation Science and Engineering*. Gothenburg, Sweden: IEEE, 2015. (to be published).

CHANG, P.-C. et al. A block-based evolutionary algorithm for flow-shop scheduling problem. *Applied Soft Computing*, Elsevier, v. 13, n. 12, p. 4536–4547, 2013.

CHANG, P.-C. et al. A block mining and re-combination enhanced genetic algorithm for the permutation flowshop scheduling problem. *International Journal of Production Economics*, Elsevier, v. 141, n. 1, p. 45–55, 2013.

CHAVES, A. A.; LORENA, L. A. N.; MIRALLES, C. Hybrid metaheuristic for the assembly line worker assignment and balancing problem. In: BLESA, M. J. et al. (Ed.). *Hybrid Metaheuristics*. Berlin: Springer, 2009, (Lecture Notes in Computer Science, v. 5818). p. 1–14. ISBN 978-3-642-04917-0.

CHAVES, A. A.; MIRALLES, C.; LORENA, L. A. N. Clustering search approach for the assembly line worker assignment and balancing problem. In: *Proceedings of the 37th international conference on computers and industrial engineering*. Alexandria, Egypt: Elsevier, 2007. p. 1469–1478.

CHEN, M.-H.; CHANG, P.-C.; LIN, C.-H. A self-evolving artificial immune system II with T-cell and B-cell for permutation flow-shop problem. *Journal of Intelligent Manufacturing*, Springer, v. 25, n. 6, p. 1257–1270, 2014.

CHEN, Y.-M. et al. Extended artificial chromosomes genetic algorithm for permutation flowshop scheduling problems. *Computers & Industrial Engineering*, Elsevier, v. 62, n. 2, p. 536–545, 2012.

CHIANG, T.-C.; FU, L.-C. An improved multiobjective memetic algorithm for permutation flow shop scheduling. In: IEEE. *Congress on Evolutionary Computation (CEC)*. Barcelona, 2010. p. 1–8.

CHONG, C. S. et al. A bee colony optimization algorithm to job shop scheduling. In: *Proceedings of the 38th Conference on Winter Simulation*. Monterey, California: Winter Simulation Conference, 2006. (WSC '06), p. 1954–1961.

COELLO, C. A. C.; RIVERA, D. C.; CORTÉS, N. C. Use of an artificial immune system for job shop scheduling. In: TIMMIS, J.; BENTLEY, P. J.; HART, E. (Ed.). *Artificial Immune Systems*. Edinburgh, UK: Springer, 2003, (Lecture Notes in Computer Science, v. 2787). p. 1–10.

COMPANYS, R.; MATEO, M. Different behaviour of a double branch-and-bound algorithm on Fm|prmu|Cmax and Fm|block|Cmax problems. *Computers & Operations Research*, Elsevier, v. 34, n. 4, p. 938–953, 2007.

CONWAY, R. W.; MAXWELL, W. L.; MILLER, L. W. *Theory of scheduling*. Massachusetts: Addison-Wesley, 1967.

COROMINAS, A.; PASTOR, R.; PLANS, J. Balancing assembly line with skilled and unskilled workers. *Omega*, Elsevier, v. 36, n. 6, p. 1126–1132, 2008.

CZAPIŃSKI, M. Parallel simulated annealing with genetic enhancement for flowshop problem with Csum. *Computers & Industrial Engineering*, Elsevier, v. 59, n. 4, p. 778–785, 2010.

DANIELS, R. L.; MAZZOLA, J. B.; SHI, D. Flow shop scheduling with partial resource flexibility. *Management Science*, INFORMS, v. 50, n. 5, p. 658–669, 2004.

DASGUPTA, P.; DAS, S. A discrete inter-species cuckoo search for flowshop scheduling problems. *Computers & Operations Research*, Elsevier, v. 60, p. 111–120, 2015.

DE BRUECKER, P. et al. Workforce planning incorporating skills: state of the art. *European Journal of Operational Research*, Elsevier, v. 243, n. 1, p. 1–16, 2015.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 6, n. 2, p. 182–197, 2002.

DEMIRKOL, E.; MEHTA, S.; UZSOY, R. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, Elsevier, v. 109, n. 1, p. 137–141, 1998.

DONG, X. et al. A multi-restart iterated local search algorithm for the permutation flow shop problem minimizing total flow time. *Computers & Operations Research*, Elsevier, v. 40, n. 2, p. 627–632, 2013.

DONG, X.; HUANG, H.; CHEN, P. An improved NEH-based heuristic for the permutation flowshop problem. *Computers & Operations Research*, Elsevier, v. 35, n. 12, p. 3962–3968, 2008.

DONG, X.; HUANG, H.; CHEN, P. An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *Computers & Operations Research*, Elsevier, v. 36, n. 5, p. 1664–1669, 2009.

DONG, X. et al. Self-adaptive perturbation and multi-neighborhood search for iterated local search on the permutation flow shop problem. *Computers & Industrial Engineering*, Elsevier, 2015.

DUAN, J.-h. et al. A speed-up method for calculating total flowtime in permutation flow shop scheduling problem. In: IEEE. *25th Chinese Control and Decision Conference (CCDC)*. Guiyang, China, 2013. p. 2755–2758.

DUBOIS-LACOSTE, J.; LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. *Computers & Operations Research*, Elsevier, v. 38, n. 8, p. 1219–1236, 2011.

EKŞIOĞLU, B.; EKŞIOĞLU, S. D.; JAIN, P. A tabu search algorithm for the flowshop scheduling problem with changing neighborhoods. *Computers & Industrial Engineering*, Elsevier, v. 54, n. 1, p. 1–11, 2008.

EMMONS, H.; VAIRAKTARAKIS, G. *Flow shop scheduling: theoretical results, algorithms, and applications*. New York: Springer, 2012. (International Handbook on Information Systems, v. 182).

EREL, E.; SABUNCUOGLU, I.; SEKERCI, H. Stochastic assembly line balancing using beam search. *International Journal of Production Research*, Taylor & Francis, v. 43, n. 7, p. 1411–1426, 2005.

ESQUIVEL, S. et al. Enhanced evolutionary algorithms for single and multiobjective optimization in the job shop scheduling problem. *Knowledge-Based Systems*, Elsevier, v. 15, n. 1, p. 13–25, 2002.

FARAHMAND RAD, S.; RUIZ, R.; BOROOJERDIAN, N. New high performing heuristics for minimizing makespan in permutation flowshops. *Omega*, Elsevier, v. 37, n. 2, p. 331–345, 2009.

FÄRBER, G.; COVES MORENO, A. M. Performance study of a genetic algorithm for sequencing in mixed model non-permutation flowshops using constrained buffers. In: *Computational Science and Its Applications-ICCSA 2006*. Glasgow, UK: Springer, 2006, (Lecture Notes in Computer Science, v. 3982). p. 638–648.

FÄRBER, G.; SALHI, S.; COVES MORENO, A. M. Semi-dynamic demand in a non-permutation flowshop with constrained resequencing buffers. In: *Large-Scale Scientific Computing*. Sozopol, Bulgaria: Springer, 2008, (Lecture Notes in Computer Science, v. 4818). p. 536–544.

FERNANDEZ-VIAGAS, V.; FRAMINAN, J. M. On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers & Operations Research*, Elsevier, v. 45, p. 60–67, 2014.

FERNANDEZ-VIAGAS, V.; FRAMINAN, J. M. A new set of high-performing heuristics to minimise flowtime in permutation flowshops. *Computers & Operations Research*, Elsevier, v. 53, p. 68–80, 2015.

FIRAT, M.; HURKENS, C. A. J. An improved MIP-based approach for a multi-skill workforce scheduling problem. *Journal of Scheduling*, Springer, v. 15, n. 3, p. 363–380, 2012.

FISCHER, H.; THOMPSON, G. L. Probabilistic learning combinations of local job-shop scheduling rules. In: MUTH, J. F.; THOMPSON, G. L. (Ed.). *Industrial Scheduling*. Englewood Cliffs, New Jersey: Prentice Hall, 1963, (International series in management). p. 225–251.

FRAMINAN, J. M.; GUPTA, J. N.; LEISTEN, R. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, Nature Publishing Group, v. 55, n. 12, p. 1243–1255, 2004.

FRAMINAN, J. M.; LEISTEN, R. A multi-objective iterated greedy search for flowshop scheduling with makespan and flowtime criteria. *OR Spectrum*, Springer, v. 30, n. 4, p. 787–804, 2008.

FRAMINAN, J. M.; LEISTEN, R.; RUIZ GARCÍA, R. *Manufacturing Scheduling Systems: An Integrated View on Models, Methods and Tools*. London: Springer, 2014.

GAREY, M. R.; JOHNSON, D. S. *Computers and intractability: a guide to the theory of NP-completeness*. San Francisco: W. H. Freeman, 1979.

GAREY, M. R.; JOHNSON, D. S.; SETHI, R. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, INFORMS, v. 1, n. 2, p. 117–129, 1976.

GE, H. et al. A particle swarm optimization-based algorithm for job-shop scheduling problems. *International Journal of Computational Methods*, World Scientific, v. 2, n. 03, p. 419–430, 2005.

GEL, E. S.; HOPP, W. J.; VAN OYEN, M. P. Factors affecting opportunity of worksharing as a dynamic line balancing mechanism. *IIE Transactions*, Taylor & Francis, v. 34, n. 10, p. 847–863, 2002.

GHARBI, A.; LABIDI, M.; LOULY, M. A. The nonpermutation flowshop scheduling problem: Adjustment and bounding procedures. *Journal of Applied Mathematics*, Hindawi Publishing Corporation, v. 2014, 2014.

GLOVER, F. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, Blackwell Publishing Ltd, v. 8, n. 1, p. 156–166, 1977.

GONÇALVES, J. F.; RESENDE, M. G. An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. *International Transactions in Operational Research*, Wiley Online Library, v. 21, n. 2, p. 215–246, 2014.

GRABOWSKI, J.; WODECKI, M. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computers & Operations Research*, Elsevier, v. 31, n. 11, p. 1891–1909, 2004.

GRAHAM, R. et al. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, Elsevier, v. 5, p. 287–326, 1979.

GUPTA, J. N.; STAFFORD, E. F. Flowshop scheduling research after five decades. *European Journal of Operational Research*, Elsevier, v. 169, n. 3, p. 699–711, 2006.

HAQ, A. N. et al. A scatter search approach for general flowshop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 31, n. 7-8, p. 731–736, 2007.

HEINONEN, J.; PETTERSSON, F. Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. *Applied Mathematics and Computation*, Elsevier, v. 187, n. 2, p. 989–998, 2007.

HELLER, J. Some numerical experiments for an M×J flow shop and its decision-theoretical aspects. *Operations Research*, INFORMS, v. 8, n. 2, p. 178–184, 1960.

HOOGEVEEN, H. Multicriteria scheduling. *European Journal of Operational Research*, Elsevier, v. 167, n. 3, p. 592–623, 2005.

HOPP, W. J.; TEKIN, E.; VAN OYEN, M. P. Benefits of skill chaining in serial production lines with cross-trained workers. *Management Science*, INFORMS, v. 50, n. 1, p. 83–98, 2004.

HSU, C.-Y.; CHANG, P.-C.; CHEN, M.-H. A linkage mining in block-based evolutionary algorithm for permutation flowshop scheduling problem. *Computers & Industrial Engineering*, Elsevier, v. 83, p. 159–171, 2015.

HUANG, K.-L.; LIAO, C.-J. Ant colony optimization combined with taboo search for the job shop scheduling problem. *Computers & Operations Research*, Elsevier, v. 35, n. 4, p. 1030–1046, 2008.

ILO. *Inclusion of persons with disabilities*. International Labour Organization, 2012. Available from Internet: <http://www.ilo.org/skills/areas/inclusion-of-persons-with-disabilities/lang--en/index.htm>.

JACKSON, J. R. An extension of Johnson's results on job IDT scheduling. *Naval Research Logistics Quarterly*, Wiley Online Library, v. 3, n. 3, p. 201–203, 1956.

JAIN, A. S.; MEERAN, S. *A multi-level hybrid framework for the deterministic job-shop scheduling problem*. Tese (Doutorado) — University Of Dundee, Scotland, UK, 1998.

JAIN, A. S.; MEERAN, S. A multi-level hybrid framework applied to the general flow-shop scheduling problem. *Computers & Operations Research*, Elsevier, v. 29, n. 13, p. 1873–1901, 2002.

JARBOUI, B.; EDDALY, M.; SIARRY, P. An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems. *Computers & Operations Research*, Elsevier, v. 36, n. 9, p. 2638–2646, 2009.

JOHNSON, S. M. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, Wiley Online Library, v. 1, n. 1, p. 61–68, 1954.

KALCZYNSKI, P. J.; KAMBUROWSKI, J. An improved NEH heuristic to minimize makespan in permutation flow shops. *Computers & Operations Research*, Elsevier, v. 35, n. 9, p. 3001–3008, 2008.

KALCZYNSKI, P. J.; KAMBUROWSKI, J. An empirical analysis of the optimality rate of flow shop heuristics. *European Journal of Operational Research*, Elsevier, v. 198, n. 1, p. 93–101, 2009.

KALCZYNSKI, P. J.; KAMBUROWSKI, J. On recent modifications and extensions of the NEH heuristic for flow shop sequencing. *Foundations of Computing and Decision Sciences*, v. 36, n. 1, p. 17, 2011.

KOULAMAS, C. A new constructive heuristic for the flowshop scheduling problem. *European Journal of Operational Research*, Elsevier, v. 105, n. 1, p. 66–71, 1998.

KRONE, M. J.; STEIGLITZ, K. Heuristic-programming solution of a flowshop-scheduling problem. *Operations Research*, INFORMS, v. 22, n. 3, p. 629–638, 1974.

LAGUNA, M.; MARTÍ, R. *Scatter Search: Methodology and Implementations in C*. New York: Springer, 2003. (Operations Research/Computer Science Interfaces Series, v. 24).

LI, H.; WOMER, K. Scheduling projects with multi-skilled personnel by a hybrid MILP/CP benders decomposition algorithm. *Journal of Scheduling*, Springer, v. 12, n. 3, p. 281–298, 2009.

LIAO, C. J.; LIAO, L. M.; TSENG, C. T. A performance evaluation of permutation vs. non-permutation schedules in a flowshop. *International Journal of Production Research*, Taylor & Francis, v. 44, n. 20, p. 4297–4309, 2006.

LIAO, C.-J.; YOU, C.-T. An improved formulation for the job-shop scheduling problem. *The Journal of the Operational Research Society*, Palgrave Macmillan Journals, v. 43, n. 11, p. 1047–1054, 1992.

LIAO, L.-M.; HUANG, C.-J. Tabu search for non-permutation flowshop scheduling problem with minimizing total tardiness. *Applied Mathematics and Computation*, Elsevier, v. 217, n. 2, p. 557–567, 2010.

LIN, S.-W.; YING, K.-C. Applying a hybrid simulated annealing and tabu search approach to non-permutation flowshop scheduling problems. *International Journal of Production Research*, Taylor & Francis, v. 47, n. 5, p. 1411–1424, 2009.

LIN, S.-W.; YING, K.-C. Minimizing makespan and total flowtime in permutation flowshops by a bi-objective multi-start simulated-annealing algorithm. *Computers & Operations Research*, Elsevier, v. 40, n. 6, p. 1625–1647, 2013.

LIN, S.-W.; YING, K.-C.; LEE, Z.-J. Metaheuristics for scheduling a non-permutation flowline manufacturing cell with sequence dependent family setup times. *Computers & Operations Research*, Elsevier, v. 36, n. 4, p. 1110–1121, 2009.

LIU, F. et al. Discrete differential evolution algorithm for the job shop scheduling problem. In: ACM. *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*. New York: ACM, 2009. (GEC '09), p. 879–882.

LIU, J.; REEVES, C. R. Constructive and composite heuristic solutions to the $P//\sum Ci$ scheduling problem. *European Journal of Operational Research*, Elsevier, v. 132, n. 2, p. 439–452, 2001.

LIU, Y.-F.; LIU, S.-Y. A hybrid discrete artificial bee colony algorithm for permutation flowshop scheduling problem. *Applied Soft Computing*, Elsevier, v. 13, n. 3, p. 1459–1463, 2013.

LÓPEZ-IBÁÑEZ, M. et al. *The irace package, iterated race for automatic algorithm configuration*. Belgium, 2011. (IRIDIA – Technical Report Series, TR/IRIDIA/2011-004).

MANNE, A. S. On the job-shop scheduling problem. *Operations Research*, INFORMS, v. 8, n. 2, p. pp. 219–223, 1960.

MANSOOR, E. Assembly line balancing-a heuristic algorithm for variable operator performance levels. *Journal of Industrial Engineering*, v. 19, n. 12, p. 618, 1968.

MARICHELVAM, M. An improved hybrid cuckoo search (IHCS) metaheuristics algorithm for permutation flow shop scheduling problems. *International Journal of Bio-Inspired Computation*, Inderscience Publishers Ltd, v. 4, n. 4, p. 200–205, 2012.

MARTÍ, R.; RESENDE, M. G.; RIBEIRO, C. C. Multi-start methods for combinatorial optimization. *European Journal of Operational Research*, Elsevier, v. 226, n. 1, p. 1–8, 2013.

MEHRAVARAN, Y.; LOGENDRAN, R. Non-permutation flowshop scheduling in a supply chain with sequence-dependent setup times. *International Journal of Production Economics*, Elsevier, v. 135, n. 2, p. 953–963, 2012.

MEHRAVARAN, Y.; LOGENDRAN, R. Non-permutation flowshop scheduling with dual resources. *Expert Systems with Applications*, Elsevier, v. 40, n. 13, p. 5061–5076, 2013.

METLICKA, M. et al. GPU accelerated NEH algorithm. In: IEEE. *Symposium on Computational Intelligence in Production and Logistics Systems (CIPLS)*. Orlando, Florida, 2014. p. 114–119.

METROPOLIS, N. et al. Equation of state calculations by fast computing machines. *The journal of chemical physics*, AIP Publishing, v. 21, n. 6, p. 1087–1092, 1953.

MINELLA, G.; RUIZ, R.; CIAVOTTA, M. A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing*, INFORMS, v. 20, n. 3, p. 451–471, 2008.

MINELLA, G.; RUIZ, R.; CIAVOTTA, M. Restarted iterated pareto greedy algorithm for multi-objective flowshop scheduling problems. *Computers & Operations Research*, Elsevier, v. 38, n. 11, p. 1521–1533, 2011.

MIRALLES, C. et al. Advantages of assembly lines in sheltered work centres for disabled: A case study. *International Journal of Production Economics*, Elsevier, v. 110, n. 1, p. 187–197, 2007.

MIRALLES, C. et al. Branch and bound procedures for solving the assembly line worker assignment and balancing problem: Application to sheltered work centres for disabled. *Discrete Applied Mathematics*, v. 156, n. 3, p. 352 – 367, 2008.

MIRALLES, C. et al. Operations research/management science tools for integrating people with disabilities into employment: A study on Valencia's sheltered work centres for disabled. *International Transactions in Operational Research*, Blackwell Publishing Ltd, v. 17, n. 4, p. 457–473, 2010.

MOKOTOFF, E. Multi-objective simulated annealing for permutation flow shop problems. In: CHAKRABORTY, U. K. (Ed.). *Computational Intelligence in Flow Shop and Job Shop Scheduling*. Berlin: Springer, 2009, (Studies in Computational Intelligence, v. 230). p. 101–150.

MOREIRA, M. C. O.; COSTA, A. M. A minimalist yet efficient tabu search algorithm for balancing assembly lines with disabled workers. In: *Anais do XLI Simpósio Brasileiro de Pesquisa Operacional*. Porto Seguro, Brasil: SBPO, 2009. p. 660–671.

MOREIRA, M. C. O. et al. Simple heuristics for the assembly line worker assignment and balancing problem. *Journal of heuristics*, Springer, v. 18, n. 3, p. 505–524, 2012.

MUTLU, O.; POLAT, O.; SUPCILLER, A. A. An iterative genetic algorithm for the assembly line worker assignment and balancing problem of type-II. *Computers & Operations Research*, v. 40, n. 1, p. 418 – 426, 2013.

NAGANO, M.; MOCCELLIN, J. et al. A high quality solution constructive heuristic for flow shop sequencing. *Journal of the Operational Research Society*, Palgrave Macmillan, v. 53, n. 12, p. 1374–1379, 2002.

NAGANO, M. S.; RUIZ, R.; LORENA, L. A. N. A constructive genetic algorithm for permutation flowshop scheduling. *Computers & Industrial Engineering*, Elsevier, v. 55, n. 1, p. 195–207, 2008.

NAGARAJAN, V.; SVIRIDENKO, M. Tight bounds for permutation flow shop scheduling. *Mathematics of Operations Research*, INFORMS, v. 34, n. 2, p. 417–427, 2009.

NASIRI, M. M.; KIANFAR, F. A guided tabu search/path relinking algorithm for the job shop problem. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 58, n. 9-12, p. 1105–1113, 2012.

NAWAZ, M.; ENSCORE, E. E.; HAM, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, Elsevier, v. 11, n. 1, p. 91–95, 1983.

NOWICKI, E.; SMUTNICKI, C. A fast taboo search algorithm for the job shop problem. *Management Science*, INFORMS, v. 42, n. 6, p. 797–813, 1996.

NOWICKI, E.; SMUTNICKI, C. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, Springer, v. 8, n. 2, p. 145–159, 2005.

NOWICKI, E.; SMUTNICKI, C. Some aspects of scatter search in the flow-shop problem. *European Journal of Operational Research*, Elsevier, v. 169, n. 2, p. 654–666, 2006.

ODDI, A. *Demirkol job shop scheduling instances*. 2005. Available from Internet: <http://pst.istc.cnr.it/~angelo/OUdata/>.

ONWUBOLU, G.; DAVENDRA, D. Scheduling flow shops using differential evolution algorithm. *European Journal of Operational Research*, Elsevier, v. 171, n. 2, p. 674–692, 2006.

OSMAN, I.; POTTS, C. Simulated annealing for permutation flow-shop scheduling. *Omega*, Elsevier, v. 17, n. 6, p. 551–557, 1989.

PAN, Q.-K.; RUIZ, R. Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research*, Elsevier, v. 222, n. 1, p. 31–43, 2012.

PAN, Q.-K.; RUIZ, R. A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers & Operations Research*, Elsevier, v. 40, n. 1, p. 117–128, 2013.

PAN, Q.-K.; TASGETIREN, M. F.; LIANG, Y.-C. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers & Industrial Engineering*, Elsevier, v. 55, n. 4, p. 795–816, 2008.

PAN, Q.-K. et al. A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. *Omega*, Elsevier, v. 45, p. 42–56, 2014.

PARDALOS, P. M.; SHYLO, O. V. An algorithm for the job shop scheduling problem based on global equilibrium search techniques. *Computational Management Science*, Springer, v. 3, n. 4, p. 331–348, 2006.

PARDALOS, P. M.; SHYLO, O. V.; VAZACOPOULOS, A. Solving job shop scheduling problems utilizing the properties of backbone and “big valley”. *Computational Optimization and Applications*, Springer, v. 47, n. 1, p. 61–76, 2010.

PÉREZ, E.; POSADA, M.; HERRERA, F. Analysis of new niching genetic algorithms for finding multiple solutions in the job shop scheduling. *Journal of Intelligent manufacturing*, Springer, v. 23, n. 3, p. 341–356, 2012.

PIN, Z.; XIAO-PING, L.; HONG-FANG, Z. An ant colony algorithm for job shop scheduling problem. In: IEEE. *Fifth World Congress on Intelligent Control and Automation, 2004. WCICA 2004.* Hangzhou, China, 2004. v. 4, p. 2899–2903.

PINEDO, M. L. *Scheduling: theory, algorithms, and systems*. 4th. ed. New York: Springer, 2012.

PONGCHAIRERKS, P.; KACHITVICHYANUKUL, V. A two-level particle swarm optimisation algorithm on job-shop scheduling problems. *International Journal of Operational Research*, Inderscience Publishers, v. 4, n. 4, p. 390–411, 2009.

PONSICH, A.; COELLO, C. A. C. Testing the permutation space based geometric differential evolution on the job-shop scheduling problem. In: SCHAEFER, R. et al. (Ed.). *Parallel Problem Solving from Nature, PPSN XI*. Kraków, Poland: Springer, 2010. (Lecture Notes in Computer Science, v. 6239), p. 250–259.

PONSICH, A.; COELLO, C. A. C. A hybrid differential evolution—tabu search algorithm for the solution of job-shop scheduling problems. *Applied Soft Computing*, Elsevier, v. 13, n. 1, p. 462–474, 2013.

POTTS, C. N.; SHMOYS, D. B.; WILLIAMSON, D. P. Permutation vs. non-permutation flow shop schedules. *Operations Research Letters*, Elsevier, v. 10, n. 5, p. 281–284, 1991.

POTTS, C. N.; STRUSEVICH, V. A. Fifty years of scheduling: a survey of milestones. *Journal of the Operational Research Society*, Nature Publishing Group, v. 60, p. S41–S68, 2009.

PUGAZHENDHI, S. et al. Performance enhancement by using non-permutation schedules in flowline-based manufacturing systems. *Computers & Industrial Engineering*, Elsevier, v. 44, n. 1, p. 133–157, 2003.

PUGAZHENDHI, S. et al. Generating non-permutation schedules in flowline-based manufacturing sytems with sequence-dependent setup times of jobs: a heuristic approach. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 23, n. 1-2, p. 64–78, 2004.

PUGAZHENDHI, S. et al. Relative performance evaluation of permutation and non-permutation schedules in flowline-based manufacturing systems with flowtime objective. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 23, n. 11-12, p. 820–830, 2004.

RAJENDRAN, C.; ZIEGLER, H. A performance analysis of dispatching rules and a heuristic in static flowshops with missing operations of jobs. *European Journal of Operational Research*, Elsevier, v. 131, n. 3, p. 622–634, 2001.

RAJENDRAN, C.; ZIEGLER, H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, Elsevier, v. 155, n. 2, p. 426–438, 2004.

RAJENDRAN, C.; ZIEGLER, H. A multi-objective ant-colony algorithm for permutation flowshop scheduling to minimize the makespan and total flowtime of jobs. In: CHAKRABORTY, U. (Ed.). *Computational Intelligence in Flow Shop and Job Shop Scheduling*. Berlin: Springer, 2009, (Studies in Computational Intelligence, v. 230). p. 53–99.

RAMAN, N. Minimum tardiness scheduling in flow shops: construction and evaluation of alternative solution approaches. *Journal of Operations Management*, Elsevier, v. 12, n. 2, p. 131–151, 1995.

REEVES, C. R. A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, Elsevier, v. 22, n. 1, p. 5–13, 1995.

RESENDE, M. G. et al. Scatter search and path-relinking: Fundamentals, advances, and applications. In: GENDREAU, M.; POTVIN, J.-Y. (Ed.). *Handbook of Metaheuristics*. 2nd. ed. New York: Springer, 2010, (International Series in Operations Research & Management Science, v. 146). p. 87–107.

REZA HEJAZI, S.; SAGHAFIAN, S. Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research*, Taylor & Francis, v. 43, n. 14, p. 2895–2929, 2005.

RIBAS, I.; COMPANYS, R.; TORT-MARTORELL, X. Comparing three-step heuristics for the permutation flow shop problem. *Computers & Operations Research*, Elsevier, v. 37, n. 12, p. 2062–2070, 2010.

RONCONI, D. P. A note on constructive heuristics for the flowshop problem with blocking. *International Journal of Production Economics*, Elsevier, v. 87, n. 1, p. 39–48, 2004.

ROSSI, A.; LANZETTA, M. Nonpermutation flow line scheduling by ant colony optimization. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Cambridge Univ Press, v. 27, n. 04, p. 349–357, 2013.

ROSSI, A.; LANZETTA, M. Scheduling flow lines with buffers by ant colony digraph. *Expert Systems with Applications*, Elsevier, v. 40, n. 9, p. 3328–3340, 2013.

ROSSI, A.; LANZETTA, M. Native metaheuristics for non-permutation flowshop scheduling. *Journal of Intelligent Manufacturing*, Springer, v. 25, n. 6, p. 1221–1233, 2014.

RUIZ, R.; MAROTO, C. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, Elsevier, v. 165, n. 2, p. 479–494, 2005.

RUIZ, R.; MAROTO, C.; ALCARAZ, J. Two new robust genetic algorithms for the flowshop scheduling problem. *Omega*, Elsevier, v. 34, n. 5, p. 461–476, 2006.

RUIZ, R.; STÜTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, Elsevier, v. 177, n. 3, p. 2033–2049, 2007.

SADJADI, S.; BOUQUARD, J.-L.; ZIAEE, M. An ant colony algorithm for the flowshop scheduling problem. *Journal of Applied Sciences*, v. 8, n. 21, p. 3938–3944, 2008.

SANTUCCI, V.; BAIOLETTI, M.; MILANI, A. A differential evolution algorithm for the permutation flowshop scheduling problem with total flow time criterion. In: BARTZ-BEIELSTEIN, T. et al. (Ed.). *Parallel Problem Solving from Nature – PPSN XIII*. Ljubljana, Slovenia: Springer, 2014, (Lecture Notes in Computer Science, v. 8672). p. 161–170.

SEVKLI, M.; AYDIN, M. E. A variable neighbourhood search algorithm for job shop scheduling problems. In: GOTTLIEB, J.; RAIDL, G. (Ed.). *Evolutionary Computation in Combinatorial Optimization*. Budapest, Hungary: Springer, 2006, (Lecture Notes in Computer Science, v. 3906). p. 261–271.

SHYLO, O. *Demirkol job shop scheduling instances: Best known lower and upper bounds*. 2005. Available from Internet: <http://optimizizer.com/DMU.php>.

SINGH, S.; SINGH, K. Cuckoo search optimization for job shop scheduling problem. In: DAS, K. N. et al. (Ed.). *Proceedings of Fourth International Conference on Soft Computing for Problem Solving*. Silchar, India: Springer, 2015. (Advances in Intelligent Systems and Computing, v. 335), p. 99–111.

SUN, Y. et al. Multi-objective optimization algorithms for flow shop scheduling problem: a review and prospects. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 55, n. 5-8, p. 723–739, 2011.

SWAMINATHAN, R. et al. Impact of permutation enforcement when minimizing total weighted tardiness in dynamic flowshops with uncertain processing times. *Computers & Operations Research*, Elsevier, v. 34, n. 10, p. 3055–3068, 2007.

TAILLARD, E. Some efficient heuristic methods for the flow shop sequencing problem. *European journal of Operational research*, Elsevier, v. 47, n. 1, p. 65–74, 1990.

TAILLARD, E. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, Elsevier, v. 64, n. 2, p. 278–285, 1993.

TAILLARD, E. 2004. Best known lower and upper bounds of the PFSSP for Taillard's instances. Available from Internet: <http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/flowshop.dir/best_lb_up.txt>.

TANDON, M.; CUMMINGS, P.; LEVANU, M. Flowshop sequencing with non-permutation schedules. *Computers & Chemical Engineering*, Elsevier, v. 15, n. 8, p. 601–607, 1991.

TASGETIREN, M. F. et al. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, Elsevier, v. 177, n. 3, p. 1930–1947, 2007.

TASGETIREN, M. F. et al. A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. *Information Sciences*, Elsevier, v. 181, n. 16, p. 3459–3475, 2011.

T'KINDT, V.; BILLAUT, J.-C. *Multicriteria scheduling: theory, models and algorithms*. 2nd. ed. Berlin: Springer, 2006.

TSENG, C.-T.; LIAO, C.-J.; LIAO, T.-X. A note on two-stage hybrid flowshop scheduling with missing operations. *Computers & Industrial Engineering*, Elsevier, v. 54, n. 3, p. 695–704, 2008.

TSENG, L.-Y.; LIN, Y.-T. A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, Elsevier, v. 198, n. 1, p. 84–92, 2009.

TSENG, L.-Y.; LIN, Y.-T. A genetic local search algorithm for minimizing total flowtime in the permutation flowshop scheduling problem. *International Journal of Production Economics*, Elsevier, v. 127, n. 1, p. 121–128, 2010.

TYAGI, N.; VARSHNEY, R.; CHANRAMOULI, A. Six decades of flowshop scheduling research. *International Journal of Scientific & Engineering Research*, v. 4, n. 9, 2013.

VAESSENS, R. J. 1996. Best known lower and upper bounds of the permutation and non-permutation FSSP for Taillard's instances, mirror by M. Lanzetta. Available from Internet: <http://www2.ing.unipi.it/lanzetta/aco/vaessens.txt>.

VAHEDI-NOURI, B.; FATTAHI, P.; RAMEZANIAN, R. Hybrid firefly-simulated annealing algorithm for the flow shop problem with learning effects and flexible maintenance activities. *International Journal of Production Research*, Taylor & Francis, v. 51, n. 12, p. 3501–3515, 2013.

VAHEDI-NOURI, B.; FATTAHI, P.; RAMEZANIAN, R. Minimizing total flow time for the non-permutation flow shop scheduling problem with learning effects and availability constraints. *Journal of Manufacturing Systems*, Elsevier, v. 32, n. 1, p. 167–173, 2013.

VAHEDI-NOURI, B. et al. A general flow shop scheduling problem with consideration of position-based learning effect and multiple availability constraints. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 73, n. 5-8, p. 601–611, 2014.

VALLADA, E.; RUIZ, R. Cooperative metaheuristics for the permutation flowshop scheduling problem. *European Journal of Operational Research*, Elsevier, v. 193, n. 2, p. 365–376, 2009.

VALLADA, E.; RUIZ, R.; FRAMINAN, J. M. New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, Elsevier, v. 240, n. 3, p. 666–677, 2015.

VARADHARAJAN, T.; RAJENDRAN, C. A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. *European Journal of Operational Research*, Elsevier, v. 167, n. 3, p. 772–795, 2005.

VASILJEVIC, D.; DANILOVIC, M. Handling ties in heuristics for the permutation flow shop scheduling problem. *Journal of Manufacturing Systems*, Elsevier, v. 35, p. 1–9, 2015.

VÁZQUEZ, M.; WHITLEY, D. A comparison of genetic algorithms for the static job shop scheduling problem. In: SCHOENAUER, M. et al. (Ed.). *Parallel Problem Solving from Nature, PPSN VI*. Paris, France: Springer, 2000. (Lecture Notes in Computer Science, v. 1917), p. 303–312.

VILÀ, M.; PEREIRA, J. A branch-and-bound algorithm for assembly line worker assignment and balancing problems. *Computers & Operations Research*, v. 44, p. 105 – 114, 2014.

WANG, J.-J.; ZHANG, B.-H. Permutation flowshop problems with bi-criterion makespan and total completion time objective and position-weighted learning effects. *Computers & Operations Research*, Elsevier, v. 58, p. 24–31, 2015.

WATSON, J.-P.; HOWE, A. E.; WHITLEY, L. D. Deconstructing Nowicki and Smutnicki's i-TSAB tabu search algorithm for the job-shop scheduling problem. *Computers & Operations Research*, Elsevier, v. 33, n. 9, p. 2623–2644, 2006.

WENQI, H.; AIHUA, Y. An improved shifting bottleneck procedure for the job shop scheduling problem. *Computers & Operations Research*, Elsevier, v. 31, n. 12, p. 2093–2110, 2004.

XIE, Z. et al. An effective hybrid teaching-learning-based optimization algorithm for permutation flow shop scheduling problem. *Advances in Engineering Software*, Elsevier, v. 77, p. 35–47, 2014.

XU, X.; XU, Z.; GU, X. An asynchronous genetic local search algorithm for the permutation flowshop scheduling problem with total flowtime minimization. *Expert Systems with Applications*, Elsevier, v. 38, n. 7, p. 7970–7979, 2011.

YAGMAHAN, B.; YENISEY, M. M. A multi-objective ant colony system algorithm for flow shop scheduling problem. *Expert Systems with Applications*, Elsevier, v. 37, n. 2, p. 1361–1368, 2010.

YAMADA, T.; NAKANO, R. Job-shop scheduling. In: _____. *Genetic algorithms in engineering systems*. London: The Institution of Electrical Engineers, 1997. (Control Engineering Series, v. 55), cap. 7, p. 134–160.

YANDRA; TAMURA, H. A new multiobjective genetic algorithm with heterogeneous population for solving flowshop scheduling problems. *International Journal of Computer Integrated Manufacturing*, Taylor & Francis, v. 20, n. 5, p. 465–477, 2007.

YENISEY, M. M.; YAGMAHAN, B. Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. *Omega*, Elsevier, v. 45, p. 119–135, 2014.

YING, K.-C. Solving non-permutation flowshop scheduling problems by an effective iterated greedy heuristic. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 38, n. 3-4, p. 348–354, 2008.

YING, K.-C. et al. Permutation and non-permutation schedules for the flowline manufacturing cell with sequence dependent family setups. *International Journal of Production Research*, Taylor & Francis, v. 48, n. 8, p. 2169–2184, 2010.

YING, K.-C.; LIN, S.-W. Multi-heuristic desirability ant colony system heuristic for non-permutation flowshop scheduling problems. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 33, n. 7-8, p. 793–802, 2007.

ZHANG, C. Y. et al. A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & Operations Research*, Elsevier, v. 35, n. 1, p. 282–294, 2008.

ZHANG, J. et al. Implementation of an ant colony optimization technique for job shop scheduling problem. *Transactions of the Institute of Measurement and Control*, SAGE Publications, v. 28, n. 1, p. 93–108, 2006.

ZHANG, L.; WU, J. A PSO-based hybrid metaheuristic for permutation flowshop scheduling problems. *The Scientific World Journal*, Hindawi Publishing Corporation, v. 2014, 2014.

ZHANG, R.; SONG, S.; WU, C. A two-stage hybrid particle swarm optimization algorithm for the stochastic job shop scheduling problem. *Knowledge-Based Systems*, Elsevier, v. 27, p. 393–406, 2012.

ZHANG, Y.; LI, X.; WANG, Q. Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. *European Journal of Operational Research*, Elsevier, v. 196, n. 3, p. 869–876, 2009.

ZHANG, Z. et al. Flow shop scheduling with reinforcement learning. *Asia-Pacific Journal of Operational Research*, World Scientific, v. 30, n. 05, p. 1350014–1–1350014–25, 2013.

ZHAO, F. et al. A hybrid differential evolution and estimation of distribution algorithm based on neighbourhood search for job shop scheduling problems. *International Journal of Production Research*, Taylor & Francis, n. ahead-of-print, p. 1–22, 2015.

ZHENG, T.; YAMASHIRO, M. A novel quantum differential evolutionary algorithm for non-permutation flow shop scheduling problems. In: IEEE. *Electrical Engineering Computing Science and Automatic Control (CCE), 2010 7th International Conference on*. [S.l.], 2010. p. 357–362.

ZOBOLAS, G.; TARANTILIS, C. D.; IOANNOU, G. Exact, heuristic and meta-heuristic algorithms for solving shop scheduling problems. In: *Metaheuristics for Scheduling in Industrial and Manufacturing Applications*. [S.l.]: Springer, 2008. p. 1–40.

ZOBOLAS, G.; TARANTILIS, C. D.; IOANNOU, G. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research*, Elsevier, v. 36, n. 4, p. 1249–1267, 2009.

APPENDIX A HEURÍSTICAS PARA ESCALONAMENTO EM FLOW SHOPS: CONSIDERANDO ESCALONAMENTOS NÃO-PERMUTACIONAIS E TRABALHADORES HETEROGÊNEOS

O problema de escalonamento de tarefas num *flow shop* (ou *flow shop scheduling problem*, FSSP) é um modelo de sistemas de produção muito comum que é bem estudado na literatura. No entanto, quase toda a literatura foca-se em escalonamentos permutacionais, como se mostra no Capítulo 3, desconsiderando soluções ótimas e quase ótimas que são escalonamentos não-permutacionais, como mostra o exemplo da Seção 4. Além disso, a prática comum padroniza os tempos de processamento de cada operação, mesmo que estes tempos variem dependendo das diferentes capacidades dos operadores das máquinas. A diferença entre as capacidades e deficiências dos trabalhadores em centros de emprego para deficientes (CEDs) é significativa, e consequentemente, esta diversidade deve ser considerada no processo de escalonamento. A seção 8.1 mostra com um exemplo claro a necessidade e os benefícios de tratar os problemas de escalonamento das tarefas e de designação de trabalhadores conjuntamente.

O foco principal da nossa pesquisa é propor métodos para resolver o problema de escalonamento de tarefas em flow shops, considerando escalonamentos não-permutacionais e trabalhadores heterogêneos, com o fim de minimizar o tempo de processamento das tarefas.

O Capítulo 1 apresenta a tese, descrevendo a motivação da nossa pesquisa, nossos principais objetivos e contribuições, e as limitações no escopo da pesquisa.

O Capítulo 2 descreve os problemas de escalonamento em shops, junto com as características e formulações deles. Como o problema de escalonamento em flow shops pode ser considerado um caso especial do problema de escalonamento em job shops (ou *job shop scheduling problem*, JSSP), o capítulo também descreve outros conceitos teóricos relevantes ao JSSP. Finalmente, o capítulo descreve as principais diferenças teóricas entre a variante permutacional e a variante não-permutacional do FSSP.

O Capítulo 3 apresenta uma breve revisão da literatura no FSSP permutacional e não-permutacional. Também revisa a literatura relevante ao JSSP, e à inclussão da diversidade dos trabalhadores no processo de planificação da produção.

O Capítulo 8 apresenta o problema combinado de designação de trabalhadores e de escalonamento de tarefas num flow shop com um exemplo didático. Também define matematicamente o problema de designação de trabalhadores heterogêneos e de escalonamento de tarefas num flow shop (ou *heterogeneous workforce assignment and flow shop scheduling problem*, Het-FSSP) e o problema de designação de trabalhadores heterogêneos e de escalonamento de tarefas num job shop (ou *heterogeneous workforce assignment and job shop scheduling problem*, Het-JSSP), discute a complexidade destes problemas e introduz novas instâncias para eles.

No Capítulo 9 propomos uma heurística *scatter search* com *path relinking* para resolver o Het-FSSP, descrevemos os componentes do método proposto, e estudamos a efetividade dele, concluindo que é possível produzir soluções que compensam as diferentes capacidades e deficiências dos trabalhadores com pequenas perdas nos objetivos da produção. Além do mais, a designação de trabalhadores heterogêneos pode ser integrada em outros problemas de escalonamento, como fizemos no Capítulo 10 onde propomos uma busca local *multi-start* para resolver o Het-JSSP com resultados similares.

O Capítulo 11 descreve brevemente uma pesquisa relacionada na inclusão de um ou dois trabalhadores heterogêneos num flow shop que foi executada por um estudante de mestrado do nosso grupo de pesquisa, Germano Carniel. Embora as principais contribuições sejam do Germano, este capítulo é incluído na tese por completude.

A seguir estudamos as diferenças práticas entre o FSSP permutacional e não-permutacional, com o objetivo de minimizar a soma total dos tempos de conclusão das tarefas (ou *total completion time*) no Capítulo 5, e com o objetivo de minimizar o tempo de conclusão da última tarefa (ou *makespan*) no Capítulo 6. Para isso, nos dois capítulos estudamos o grau de esforço necessário para encontrar escalonamentos não-permutacionais de boa qualidade com as heurísticas propostas, comparamos os requerimentos de armazenamento intermediário (ou *buffers*) dos escalonamentos permutacionais e não-permutacionais obtidos e avaliamos a quantidade de reordenamento de tarefas nos escalonamentos não-permutacionais obtidos.

No Capítulo 7 propomos uma representação permutacional para escalonamentos não-permutacionais, um método acelerado para calcular o makespan que resulta da inserção de uma tarefa numa heurística construtiva para escalonamentos não-permutacionais, e duas buscas locais para escalonamentos não-permutacionais. As heurísticas propostas são embutidas em algoritmos *iterated greedy* para avaliar a efetividade deles em encontrar escalonamentos não-permutacionais de boa qualidade.

Finalmente, o Capítulo 12 apresenta as conclusões desta pesquisa, e possíveis rotas para pesquisa no futuro. A duas principais conclusões desta tese são apresentadas a seguir.

Primeiro, a integração do problema de designação de trabalhadores heterogêneos nos problemas de escalonamento de tarefas é possível e vantajosa. O Het-FSSP e o Het-JSSP podem ser resolvidos satisfatoriamente pelos métodos propostos, e as soluções produzidas compensam as deficiências e as diferentes capacidades dos trabalhadores, com poucas ou sem perdas na produtividade.

Segundo, a consideração de escalonamentos não-permutacionais para resolver o FSSP com os métodos propostos é possível, usando o mesmo tempo que métodos do estado da arte usam para resolver o FSSP permutacional, e produzindo escalonamentos não-permutacionais com melhor qualidade do que escalonamentos permutacionais e não-permutacionais produzidos por métodos do estado da arte.