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

GERMANO CAUMO CARNIEL

Including workers with disabilities in flow shop scheduling

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

Advisor: Prof. Dr. Marcus Ritt

Porto Alegre
December 2015

CIP — CATALOGING-IN-PUBLICATION

Carniel, Germano Caumo

Including workers with disabilities in flow shop scheduling / Germano Caumo Carniel. – Porto Alegre: PPGC da UFRGS, 2015.

56 f.: il.

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

1. Flow shop scheduling. 2. Integer programming. 3. Workers with disabilities. I. Ritt, Marcus. II. Título.

“THE ROAD TO WISDOM?
Well, it’s plain
and simple to express.
Err and err and err again,
but less and less and less.”

— PIET HEIN

ACKNOWLEDGEMENTS

I would like to hugely thank my parents for the lessons and examples of work and dedication, in special my mother, which is a life model for me, by having given so much to my development and taught me the values of education, culture and respect to others.

To my family and close friends for always being with me and for always being ready to help. I also would like to really thank my wife Leini, for all the love, affection and comprehension during all times.

To my research group colleagues, André, Tadeu, Borba, Moura, Paulo, Marcelo, Renato, Arton and Fernando for all their help with difficult questions and for the friendship.

A special thanks to my advisor Marcus Ritt for the patience and for teaching me countless lessons, without him I would never make this far.

This work was supported by CNPq (Brazilian National Research Council) and by Instituto de Informática da UFRGS.

ABSTRACT

Persons with disabilities have severe problems participating in the job market and their unemployment rate is usually much higher than the average of the population. This motivates the research of new modes of production which allow to include these persons at a low overhead. In this work we study the inclusion of persons with disabilities into flow shops with the objective of minimizing the makespan. Since flow shops usually have only a few machines, we focus on the inclusion of one and two workers. We define the problem, propose mathematical models and a heuristic solution, as well as realistic test instances. In computational tests we evaluate the performance of the models and the heuristic, and assess the utility of such a model of inclusion. We conclude that the problem can be solved satisfactorily, and that including workers with disabilities into flow shops is economically feasible.

Keywords: Flow shop scheduling. Integer programming. Workers with disabilities.

Incluindo trabalhadores com deficiência em flow shops

RESUMO

Pessoas com deficiências possuem muitas dificuldades em participar do mercado de trabalho, possuindo uma taxa de desemprego bem maior do que a média populacional. Isso motiva o estudo de novos modos de produção que permitam incluir essas pessoas com baixo custo operacional.

Neste trabalho é feito um estudo sobre a inclusão de pessoas com deficiências em flow shops com o objetivo de minimizar o makespan. Como flow shops normalmente possuem poucas máquinas, o foco do estudo é na inserção de um e dois trabalhadores. O problema é definido, são propostos modelos matemáticos e uma solução heurística para resolvê-lo, assim como instâncias de teste realistas. Nos testes computacionais a performance dos modelos e da heurística é avaliada e a utilidade prática deste modelo de inclusão é analisada. Nós concluímos que o problema pode ser resolvido de forma satisfatória e que a inclusão de trabalhadores com deficiêcia emn flow shops é economicamente viável.

Palavras-chave: Escalonamento de tarefas, Programação inteira, Trabalhadores com deficiências.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS AND ACRONYMS

ALWABP	Assembly Line Worker Assignment and Balancing Problem
APRD	Average Percentage Relative Deviation
DP	Dynamic Programming
FSSP	Flow Shop Scheduling Problem
FSISP	Flow Shop Insertion and Scheduling Problem
HFSISP	Hybrid Flow Shop Insertion and Scheduling Problem
HPFSISP	Hybrid Permutation Flow Shop Insertion and Scheduling Problem
IGA	Iterated Greedy Algorithm
LP	Linear Program
MILP	Mixed Integer Linear Program
PFSSP	Permutation Flow Shop Scheduling Problem
PFSISP	Permutation Flow Shop Insertion and Scheduling Problem
SWD	Sheltered Work Centers for Disabled
WHO	World Health Organization
WWD	Worker With Disabilities

CONTENTS

1 INTRODUCTION

In 2004, the World Health Survey and the Global Burden of Disease project estimated the population of persons of 15 years and older with a disability around 785 (15.6%) to 975 (19.4%) million. According to the World Health Organization (WHO) and the International Labour Organization, unemployment rates are much higher for persons with disabilities than for persons without disabilities in both developed and developing countries. However, almost all tasks can be performed by persons with disabilities, since they often have the necessary skills, and most of them can be productive in an appropriate environment (WHO, 2011). Among the workers with disabilities (WWDs), many are employed in production occupations in the manufacturing industry (BRAULT, 2012)

Governments of many countries adopt strategies to incorporate persons with disabilities into the labour force, for example in Sheltered Work Centers for Disabled (SWDs), which are facilities that employ people with disabilities exclusively or primarly, or by laws that oblige companies to contract a minimum percentage of WWDs. These models of socio-labor integration try to overcome the stereotype that considers people with disabilities as unable to develop continuous professional work (MIRALLES et al., 2010).

Motivated by similar studies that have demonstrated that such workers can be integrated successfully in assembly lines (e.g. Miralles et al. (2007)), we study in this work the integration of WWDs into flow shops scheduling problem with the objective of minimizing the makespan. Since flow shops have relatively few machines, and legislation usually foresees an integration of about 2% to 5% of WWDs for companies with at least 100 employees (BRASIL, 1991), we focus on the case of the integration of one and two workers into a flow shop.

The rest of this text is organized as follows: the next sections reviews the literature on related problems and states the contributions of this work. In Chapter 2 we explain flow shops in general and motivate two variants of the problem of integration of WWDs into flow shops by means of examples. In Chapter 3 we formulate integer programming (IP) models, and in Chapter 4 we propose heuristic solutions procedures for the problems. In Chapter 5 we define a set of instances modelling realistic conditions, present computational experiments with several solution methods, and analyze the results. Finally, in Chapter 6 we discuss the results, and offer some conclusions.

1.1 Literature review

Several researchers have been studying the problem of integrating persons with disabilities in production processes. For assembly lines, Miralles et al. (2007) have proposed the Assembly Line Worker Assignment and Balancing Problem (ALWABP). In this problem the task execution time depends on the worker, and tasks as well as workers must be assigned to a fixed number of stations such that the production rate of the assembly line is maximized. This problem is NP-hard, since it generalizes the NP-hard Simple Assembly Assembly Line Balancing Problem. Research has focused mainly on heuristic methods for solving it (MIRALLES et al., 2008; MIRALLES et al., 2010; BLUM; MIRALLES, 2011; ARAÚJO; COSTA; MIRALLES, 2012; MUTLU; POLAT; AYCA, 2013; MOREIRA et al., 2012) but effective exact solution techniques are available (VILA; PEREIRA, 2014; BORBA; RITT, 2014).

The ALWABP assumes that all workers have some disability and take therefore different times to execute the tasks. Moreira, Miralles and Costa (2015) studied the effect of including only one disabled worker in an assembly line. This models the case of conventional factories, where WWDs must be employed among regular workers. The authors conclude that it is possible to integrate person with disabilities with little impact on the efficiency of the assembly line.

Benavides, Ritt and Miralles (2014) investigated flow shops with heterogeneous workers, where each machine is operated by a different worker. They discovered that the modified problem is harder than the traditional flow shop, since mathematical models could not solve even small instances, but a scatter search heuristic managed to solve it satisfactorily. Their methods are specially useful in practice at SWDs, where all workers have different execution times and depending on the available resources, the optimal schedule varies.

The regular flow shop scheduling problem (FSSP) has been extensively studied in the literature. Most research focuses on the simpler permutation flow shop problem (PFSSP), where all the jobs have to be processed in the same order on all machines, although Potts, Shmoys and Williamson (1991) showed that there are instances for which the makespan of an optimal solution for the PFSSP is worse than the optimal solution for the non-permutation FSSP by more than a factor of $\sqrt{m}/2$, where m is the number of machines in the flow shop. The PFSSP can be solved in polynomial time for two machines (JOHNSON, 1954) but is NP-hard for three or more machines (GAREY; JOHNSON, 1979). The existence of a constant factor polynomial-time approximation algorithm for flow shop scheduling is open (although there exists a polynomial-time approximation scheme for a fixed number of machines (HALL, 1998)). We also study a variation of the hybrid flow shop scheduling problem which allows parallel machines. The problem is known to be NP-hard even when restricted to two stages where one of the stages has two parallel machines and the other has a single machine (GUPTA, 1988). For more details on approaches to solve the flow shop scheduling problems, we refer to the excellent surveys of (GUPTA; STAFFORD, 2006) and (POTTS; STRUSEVICH, 2009). For hybrid flow shops the survey of Ruiz and Vázquez-Rodríguez (2010) gives a good overview of solution methods.

1.2 Contributions

The specific case of integrating a small percentage of WWDs into flow shops has, to the best of our knowledge, not been studied in the literature so far. Since WWDs take different, usually higher processing times to perform an operation, the processing time of the "machine" or work center will depend on the worker that is assigned to it. The additional component of finding the optimal allocation of the WWDs to machines increases the number of possible solutions of the permutation flow shop by a factor m. Computation time in flow shops are important in practice due to rescheduling caused by workers rotation, turnover and absenteeism, mainly with WWDs, where periodical health and psychological support are more usual (MIRALLES et al., 2007). Therefore, advanced techniques are necessary to obtain the optimal allocation of the WWDs in addition to the optimal schedule in a reasonable computation time.

This work intends to present new models and approaches to cope with issues that arise in medium or large companies that have a small percentage of WWDs employed, when the diversity of the workers needs to be considered. The solution methods proposed to this problem allow to measure the productivity lost or gained when compared to a regular flow shop with no WWDs, helping the managers of the companies to estimate costs and make better decisions when integrating these workers.

2 PROBLEM DEFINITION

In a manufacturing environment, the steps that are needed to make the products, called *jobs*, can be divided in one or more *tasks* or *operations*, where each must be executed on some kind of machine and is the smallest piece of work that is suitable to consider (EMMONS; VAIRAKTARAKIS, 2013). Each task has a processing time and may require different materials, machinery, equipment and operators. Thus, the job goes through several *stages* of work, each one consisting of one or more productive facilities (e.g. drill presses, ovens, human inspectors, etc), called *machines*.

A *schedule* is a specification of when each task of a given job is to be processed and with what equipment, personnel, etc. The problem of *job shop scheduling* is to determine an *optimal schedule:* a schedule that satisfies all constraints and has the best value for the objective desired. Figure 2.1 shows a diagram of how the information flows in a manufacturing system.

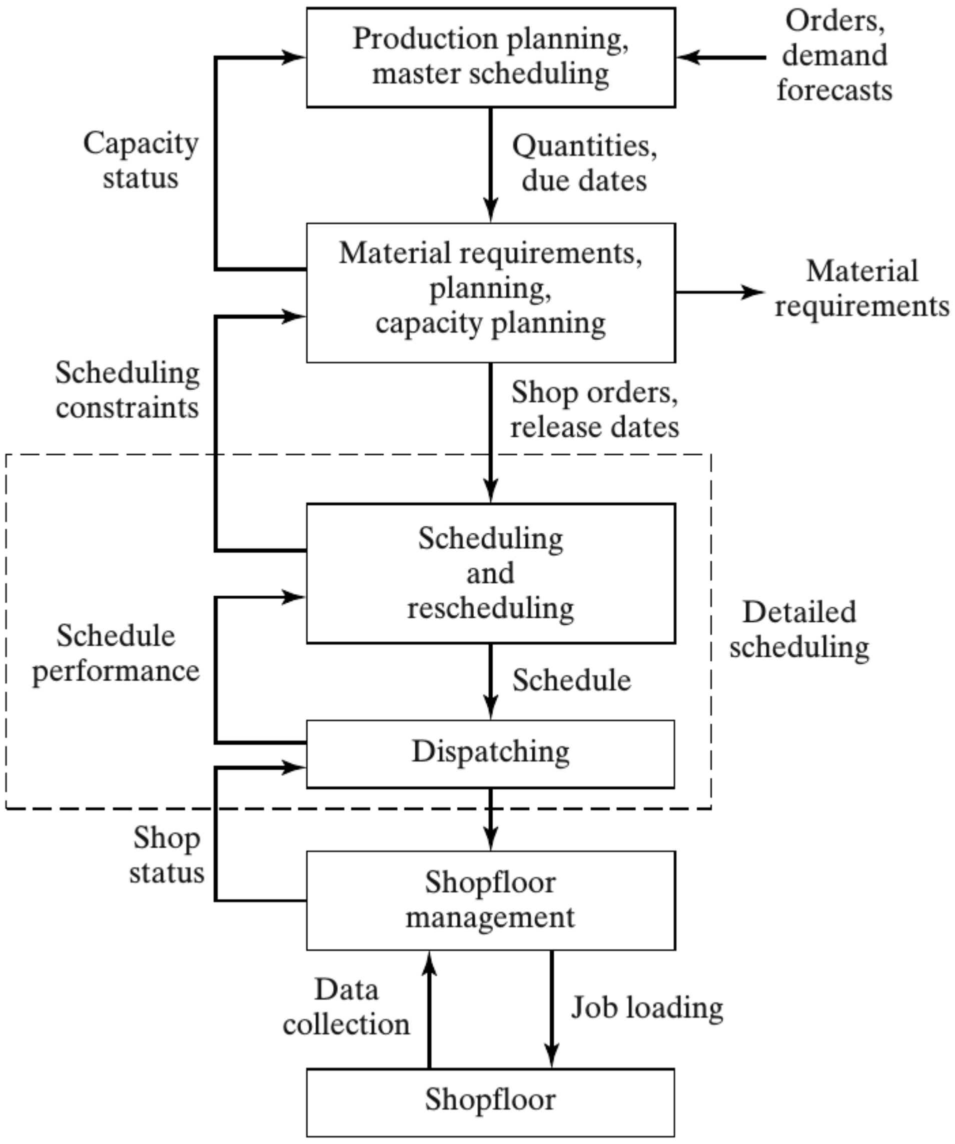
A *flow shop* is defined as a processing system where the task sequence of each job is linearly ordered, as depicted in Figure 2.2, and all jobs have to pass through the stages in the same order, they follow the same path. Here we will consider that a job can never revisit a stage, so the stages are numbered $1,2,\ldots,m$, and every job visits them in numerical order.

2.1 Assumptions and notation

There are a lot of different variations of the general flow shop, each using different constraints. The version studied here is the most common and fundamental, with the following characteristics:

- n independent jobs have to be processed, each made up of m tasks where task i of job j needs the processor i for a processing time $p_{ij} > 0$. So, each task of a job requires a different machine and the machine sequence is the same for all jobs;
- jobs are available at time zero, and continue to be available until all the work is completed;
- each job can only be processed on one machine at a time;
- any machine can process only one operation at a time;
- there is no *preemption* between the execution of the tasks: once started, a task must be completed until the end without interruptions;

Figure 2.1 – Information flow diagram in a manufacturing system.



Source: Pinedo (2012b, p. 5)

- the machine setup times are negligible: when a job completes processing on one machine, it is immediately available to begin on the next;
- the capacity of the buffer between machines (the space for the work to queue up) is unlimited;
- Problem data are deterministic and known in advance.

A more general variation of the flow shop that will also be considered is the *hybrid* flow shop, also known as flow shop *with multiple processors* or *flexible* in the literature. In this variation, each stage can have more than one machine that operates in parallel and the machines can be identical or unrelated, but all are capable of processing the tasks assigned to that stage. *Unrelated machines* means each machine in a stage processes a job at a different speed. This

Figure 2.2 – Chain precedence for a job in a flow shop

$$\begin{tikzcd} T_1 \arrow[r] & T_2 \arrow[r] & T_3 \arrow[r] & \cdots\cdots \arrow[r] & T_m \end{tikzcd}$$

Source: Emmons and Vairaktarakis (2013, p. 2)

form of the problem is especially relevant in practice, since it covers cases that are more likely to be found in real production systems (RUIZ; VÁZQUEZ-RODRÍGUEZ, 2010).

Where not otherwise stated, the common notation shown in Tables 2.1 and 2.2 are used to describe the problem[1].

Table 2.1 – Common notation used in flow shop scheduling problems

Notation	Description
n	Number of jobs to be scheduled
m	Number of stages in the shop. In a simple flow shop, each stage has one machine, so m is also the number or machines in this shop
J_j	Job j ($j \in [n]$)
M_i	Machine i, i.e., the machine at stage i ($i \in [m]$) in a simple flow shop
T_{ij}	Task i of J_j, i.e., the task of J_j processed at stage i
p_{ij}	Processing time of job J_j on stage i
r_j	Release date of J_j: when J_j is available to be processed
d_j	Due date of J_j: a desired latest completion time. Violating this date usually generates a penalty.
w_j	The *weight* of J_j, a measure that models the importance of a job. It may represent the cost of the job, volume, relative priority, among others.

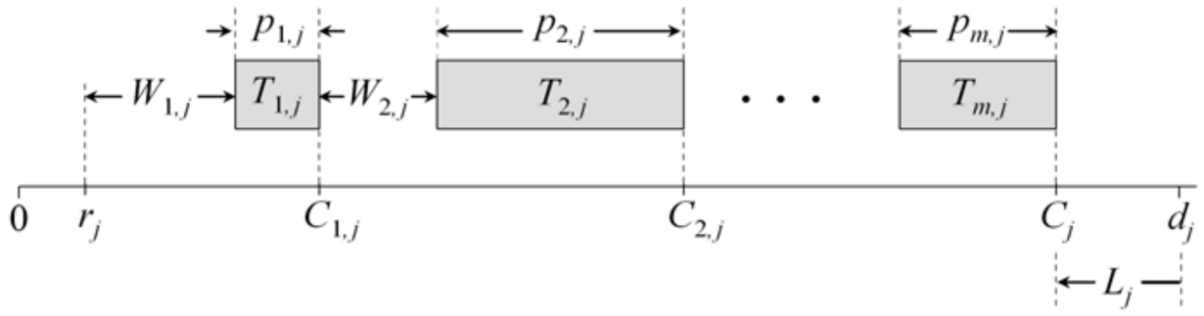
Table 2.2 – Schedule-dependent variables

Notation	Description
C_{ij}	Completion time of job j at stage i
C_j	Completion time of job j at the last stage (m)
$F_j = C_j - r_j$	Flow time of job j: the time elapsed from release to completion
W_{ij}	Waiting time between the completion of J_j at stage $i-1$ and the start at stage i
$L_j = C_j - d_j$	*Lateness* of J_j. L_j is negative if the job is early
$T_j = \max\{0, L_j\}$	The *tardiness* of J_j
U_j	A penalty measure for every tardy job, $U_j = 1$ if J_j is late, 0 otherwise

A *Gantt chart* is used to present a schedule graphically. Figure 2.3 presents an example of the progress of a job with the components presented above. Tasks are represented by a bar

[1]We use the notation $[n] = \{1, 2, \ldots, n\}$

Figure 2.3 – Gantt chart example



Source: Emmons and Vairaktarakis (2013, p. 5)

along a horizontal time axis, where the position and length corresponds to its time and duration. In this example the job is completed early, since it completes before the due date.

The $\alpha|\beta|\gamma$ notation, introduced by Graham et al. (1979) is commonly used to classify scheduling problems precisely and in a concise manner.

The α field represents the type and size of the shop, indicated by a letter and a number: Fm is the simple flow shop with m machines. Other examples include FH for Hybrid flow shop, Gm for the general job shop, where the machine sequence of the jobs may differ and Om for the open shop where the tasks of the jobs have no precedence, i.e. they can be executed in any order.

The second field β comprises constraints and assumptions that deviate from the simple flow shop. The most common examples are r_j where jobs may be ready at any time $r_j > 0$, $pmtn$ for allowing preemption, and *prmu* indicating that only *permutation schedules* are permitted: the job order must be the same on all machines.

Finally, the γ parameter specifies the *objective function* to be minimized. The objectives of a schedule are a function of the completion times of the jobs: they depend only on the completion time of the last operation of each job. The most common and extensively studied objective is $C_{\max} = \max C_j$, i.e. the maximum over all completion times C_j, $j \in [n]$ of the jobs, also called the *makespan*. It is the time the last job leaves the system and minimizing it implies a good utilization of the machines. Other functions can be seen in Table 2.3.

An example of an instance and a solution for the $Fm|prmu|C_{max}$ and $Fm||C_{max}$ is shown in Figure 2.4. In the table on the top we are given four jobs and their processing times on four machines. The optimal schedule shown on the left is a permutation schedule formed by the sequence $2,4,3,1$ and has a makespan of 11. The schedule on the right also has a makespan of 11 but note that this is no permutation schedule, since jobs two and three exchange their processing order on machine three.

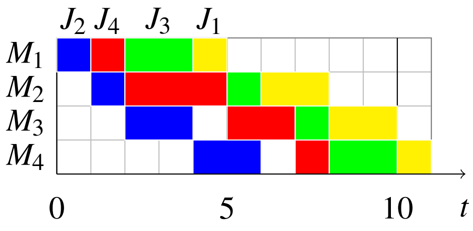
Table 2.3 – Common objective functions.

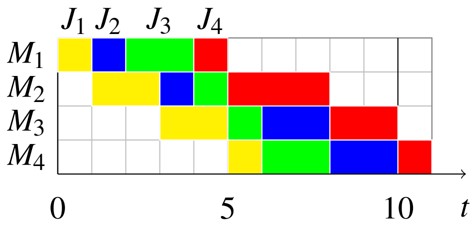
Description	Meaning
max C_j	Maximum completion time
max (F_j)	Maximum flow time
max (L_j)	Maximum lateness
max (T_j)	Maximum tardiness
$\sum C_j$	Total/average completion time
$\sum w_j C_j$	Total/average weighted completion time
$\sum F_j$	Total/average flow time
$\sum w_j F_j$	Total/average weighted flow time
$\sum U_j$	Number of late jobs
$\sum w_j U_j$	Total weighted number of late jobs

Source: Ruiz and Vázquez-Rodríguez (2010, p. 2).

Figure 2.4 – Example of a flow shop instance. Top: Processing times. Left and right: Gantt chart of optimal schedules for PFSSP and FSSP.

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





A shop with the *prmu* constraint, called *permutation flow shop* is the most studied in the literature and will be the focus of this work. Johnson (1954) has shown that there always exists an optimal schedule such that the processing order of the jobs on the first two machines and the last two machines is the same. Thus, discounting these schedules the non-permutation variant has up to $(n!)^{max(m-2,1)}$ different possible solutions, while the permutation variant permits only the $n!$ with the same order on all machines. This is the reason most of the research has focused on the permutation variant, although for instances found in practice Tandon, Cummings and LeVan (1991), Liao, Liao and Tseng (2006) have shown that the makespan can be improved by 1% to 3% for the non-permutation flow shop. The concept of a permutation schedule for hybrid flow shops is more subtle: each job is scheduled in order at the stages at the earliest feasible time that a machine at that stage becomes available (EMMONS; VAIRAKTARAKIS, 2013).

2.2 Inserting a single worker into a flow shop

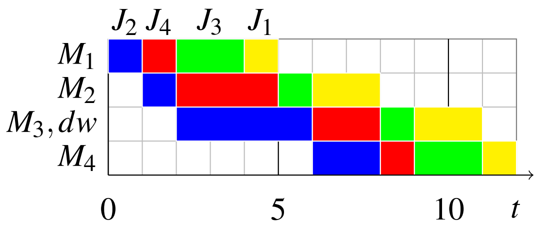
The situation encountered when we need to integrate WWDs into a regular workforce can be seen in the left part of Figure 2.5. In addition to the times that regular workers take to perform the operations, we have times for a worker with disabilities (WWD), which normally exceed the time of the regular 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 represented by a processing time of ∞ on this machine.

The problem of inserting a WWD into a flow shop (flow shop insertion and scheduling problem, FSISP) is defined as follows: we have to assign the WWD to a machine he is able to operate, and find a valid schedule of the jobs, such that the makespan is minimized. We call the variant restricted to permutation schedules the permutation flow shop insertion and scheduling problem (PFSISP).

The optimal makespan for PFSISP in the above example is 12, obtained when assigning the disabled worker to machine M_3, and can be seen in the right part of Figure 2.5. The permutation in this case is $2, 4, 3, 1$.

Figure 2.5 – An instance of PFSISP, where a single WWD must operate one machine in a flow shop. Left: Processing times for regular worker and WWD. Right: Gantt chart of optimal schedule

	Regular				With disabilities			
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	∞



2.3 Inserting two workers into a hybrid flow shop

When assigning a single, possibly slow worker to a machine in a flow shop, he will most likely be a bottleneck and increase the makespan compared to a solution with regular workers. Therefore we also study a hybrid flow shop in which two WWDs are assigned to a single stage with two parallel machines. Such a design allows to integrate more WWDs into the production line and at the same time has the potential to compensate for their increased execution times. This is the assumption we aim to validate.

There are many different variants of hybrid flow shops (RUIZ; VÁZQUEZ-RODRÍGUEZ, 2010), but here we are going to use the same assumptions as for the FSSP, the only difference being that the stage with the WWDs will contain two parallel machines. Since the two workers have different processing times, the machines are unrelated. Using the $\alpha|\beta|\gamma$ notation for hybrid flow shop from Ruiz and Vázquez-Rodríguez (2010) the permutation variant is denoted $FH_m, (R2)^k, (1)^i_{i\in[m]\setminus\{k\}}|prmu|C_{max}$. We call this problem the Hybrid Flow Shop Insertion and Scheduling Problem (HFSISP), and its permutation variant the Hybrid Permutation Flow Shop Insertion and Scheduling Problem (HPFSISP).

An instance of these problems consists of the processing times for the regular workers, and two (usually different) sets of processing times for the WWDs. A solution is given by an assignment of the two workers to some stage, and a processing order for the jobs. In the permutation version, we require the jobs to obey the processing order on each of the parallel machines. However, with this it is still possible that the operations start out of order on a stage, making the problem harder, since we must find a schedule between the machines of the stage.

An instance based on the example for the single worker insertion is shown in Figure 2.6. In the upper part, WD 1 and WD 2 refer to the two WWDs. An optimal makespan of 11 can be obtained when the WWDs are assigned to the third stage. The makespan is the same as the one obtained for the regular problem without WWDs. This shows that the parallel stage was able to compensate the longer processing times of the WWDs. This will be investigated in more detail in the computational experiments.

Figure 2.6 – An instance of HPFSISP, where two WWDs must operate a dual-machine stage in a flow shop. Top: Processing times for regular worker and WWDs. Bot: Gantt chart of an optimum solution.

	Regular				WD 1				WD 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	∞

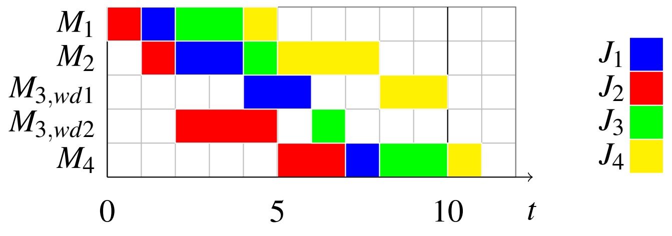
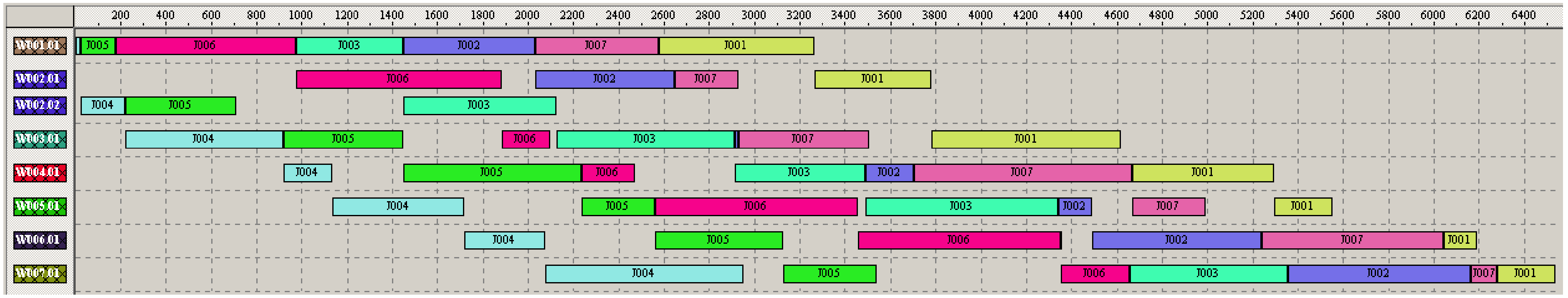


Figure 2.7 shows another example of an optimal schedule with two WWDs for a real benchmark instance: the instance number 7 from Carlier (1978). The Gantt chart was generated

using the Lekin system (PINEDO, 2012a). The processing times for the WWDs are again in the $[p,2p]$ range, and they are assigned to machine 2 in the figure. The optimal makespan is 6537 compared to an optimal makespan of 6558 in the normal flow shop.

Figure 2.7 – Example of a schedule for a benchmark instance



3 MATHEMATICAL FORMULATION OF THE FLOW SHOP INSERTION PROBLEMS

Mathematical formulation or mathematical programming is a method to model an optimization problem in terms of mathematical variables and constraints. The most basic example is the *Linear Program* (LP) where the objective and constraints are linear in the decision variables, meaning that the effect of changing a decision variable is proportional to its magnitude. When there is no integer restrictions on the decision variables, the formulation is solvable efficiently (polynomial time) by techniques such as the Simplex Algorithm and Karmarkar's Algorithm. When some decision variables are required to be integer, indicating a discrete choice, the problem is called a Mixed-Integer Linear Program (MILP), and it becomes NP-Hard. There exists free and commercial software that solves MILP with sophisticated mathematical techniques.

Most scheduling problems can be modelled as MILPs. In this chapter we adapt some well known models existing for the regular and hybrid flow shop problem to the integration case in order to have an exact solution to the new problems and to be able to compare with other methods.

3.1 Flow Shop

In this section we present a mathematical formulation of the FSISP, and two formulations for the PFSISP that consider the inclusion of a WWD. When presenting the models we use subscripts $r \in [m]$ for machines, $i \in [n]$ for jobs, and $j \in [n]$ for sequence positions. Table 3.1 explains the variables used in the moldels.

3.1.1 The flow shop scheduling and insertion problem

The model for flow shop scheduling is based on the model of Wagner (1959) using dichotomous constraints for ordering the jobs on each machine. We extend this model by modifying the processing times according to the assignment of the WWD and introducing a decision variable corresponding to the machine the worker is assigned to (X_r).

$$\textbf{min.} \quad C_{\max}, \tag{3.1}$$

$$\textbf{s.t.} \quad S_{ri} + T_{ri} \leq C_{\max}, \qquad \forall r \in [m], i \in [n], \tag{3.2}$$

Table 3.1 – Notation used in the mixed integer programming models.

A	set of machines which the WWD can operate
p_{ri}	processing time of job J_i on machine M_r when executed by regular workers
d_{ri}	processing time of job J_i on machine M_r when executed by the WWD
P	a large constant
C_{ri}	completion time of job J_i on machine M_r
S_{ri}	starting time of job J_i on machine M_r
B_{rj}	starting time of job in sequence position j on machine M_r
Y_{rj}	waiting time of job in sequence position j after it finishes processing on machine M_r
T_{ri}	processing time of job J_i on machine M_r after assigning the WWD
T_{rj}	processing time of job in sequence position j on machine M_r after assigning the WWD
D_{ik}	1, if job J_i precedes job J_k, 0, otherwise
D_{ikr}	1, if job J_i precedes job J_k on machine r, 0, otherwise
Z_{ij}	1, if job J_i is assigned to sequence position j, 0 otherwise
X_r	1, if the WWD is assigned to machine M_r, 0 otherwise

$$S_{ri} + T_{ri} \leq S_{r+1,i}, \qquad \forall r \in [m-1], i \in [n], \tag{3.3}$$

$$S_{ri} + T_{ri} \leq S_{rk} + P(1 - D_{ikr}), \qquad \forall r \in [m], i,k \in [n], i < k, \tag{3.4}$$

$$D_{ikr} + D_{kir} = 1, \qquad \forall i,k \in [n], i < k, r \in [m], \tag{3.5}$$

$$T_{ri} = p_{ri}(1 - X_r) + d_{ri}X_r, \qquad \forall r \in [m], i \in [n], \tag{3.6}$$

$$\sum_{r \in A} X_r = 1. \tag{3.7}$$

In this model constraint (3.2) defines $C_{\max}$ as the latest completion time. Constraints (3.3) and (3.4) set the starting time of all operations according to the precedences. Constraint (3.5) enforces precedence relations for operations on the same machine. The processing time of an operation depends on the machine the WWD is assigned to, and is defined in constraint (3.6). Constraint (3.7) requires that the disabled worker is assigned to one of the machines she can operate.

3.1.2 The permutation flow shop insertion and scheduling problem

Stafford, Tseng and Gupta (2004) and Tseng and Stafford (2007) present an extensive comparison of models for the PFSSP. The model the authors identified as performing best, called TS3, however, is harder to adapt for the PFSISP, since it multiplies the processing times with binary decision variables Z_{ij}, which define the permutation of the jobs. For this reason, we study the extension of two models to the PFSISP: the best model using dichotomous constraints, called LYeq, as well as a linearization of the TS3 model.

3.1.2.1 Adapting the LYeq model to the PFSISP

The LYeq model of Liao and You (1992) for the job shop scheduling problem was applied by Pan (1997) to the PFSSP. It can be modified in the same manner as the Wagner model to include an additional WWD. The model uses the additional surplus variables Q_{rik} for the time between the completion of job J_k and the start of job J_i, if k precedes i. The constant P can be set to $\sum_{j\in[n]} \max_{r\in[m]} d_{ri}$.

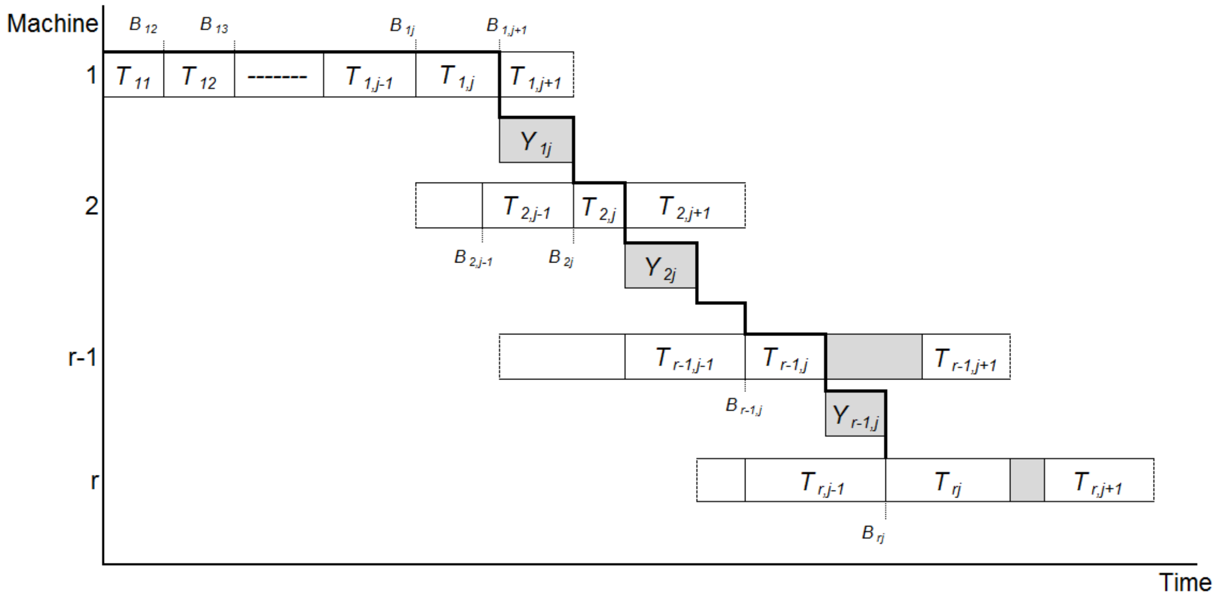
$$
\begin{aligned}
\textbf{min.} \quad & C_{\max}, && && (3.8)\\
\textbf{s.t.} \quad & C_{mi} \le C_{\max}, && \forall i \in [n], && (3.9)\\
& C_{1i} \ge T_{1i}, && \forall i \in [n], && (3.10)\\
& C_{r+1,i} - C_{ri} \ge T_{r+1,i}, && \forall r \in [m-1], i \in [n], && (3.11)\\
& PD_{ik} + C_{ri} - C_{rk} - T_{ri} = Q_{rik}, && \forall r \in [m], i,k \in [n], i<k, && (3.12)\\
& Q_{rik} \le P - T_{ri} - T_{rk}, && \forall r \in [m], i,k \in [n], i<k, && (3.13)\\
& Q_{rik} \ge 0 && \forall r \in [m], i,k \in [n] && (3.14)\\
& \text{equations (3.6) and (3.7).}
\end{aligned}
$$

Constraint (3.9) defines $C_{\max}$ as the latest completion time on the last machine. Constraint (3.10) ensures each job can only be completed on machine 1 after it is fully processed on that machine. Constraint (3.11) states that each job is processed on only one machine at a time. Constraint (3.12) replaces the dichotomous constraints in the Wagner model, and equation (3.13) upper bounds the surplus variables Q_{rik}, in case i precedes k.

3.1.2.2 Adapting the TS3 model to the PFSISP

The TS3 model was proposed by Tseng and Stafford (2007). The underlying principle of the formulation is to define the starting times of job J_j at machine M_r, called B_{rj}, by the sum of its predecessors on the first machine, plus the sum of its processing times on machines $1,\dots,r-1$ and the waiting time on these machines before starting on the next machines. This principle can be seen by the thick line in Figure 3.1. The comparison of these partial sums yields compact constraints for expressing the relative starting times for a given job permutation.

Figure 3.1 – Principle of the TS3 model



Source: Tseng and Stafford (2007)

$$\textbf{min.} \quad C_{\max} = \sum_{p\in[n]} T_{1p} + \sum_{q\in[2,m]} T_{qn} + \sum_{q\in[m-1]} Y_{qn}, \tag{3.15}$$

$$\textbf{s.t.} \quad \sum_{i\in[n]} Z_{ij} = 1, \qquad \forall j \in [n], \tag{3.16}$$

$$\sum_{j\in[n]} Z_{ij} = 1, \qquad \forall i \in [n], \tag{3.17}$$

$$T_{1,j-1} - T_{r,j-1} + \sum_{q\in[r-1]} T_{qj} - T_{q,j-1} + \sum_{q\in[r-1]} Y_{qj} - Y_{q,j-1} \geq 0, \qquad \forall r \in [2,m], j \in [2,n], \tag{3.18}$$

$$T_{rj} = \sum_{i\in[n]} p_{ri}(1-X_r)Z_{ij} + d_{ri}X_rZ_{ij}, \qquad \forall r \in [m], j \in [n], \tag{3.19}$$

equation (3.7).

Constraint (3.15) defines $C_{\max}$ as the sum of three components: (i) the processing time of all jobs on the first machine, (ii) the processing time of last job on all remaining machines, and (iii) the waiting times of the last job on all machines, except the last one. Constraints (3.16) and (3.17) model the assignment of jobs to positions: each job is assigned to only one sequence position and each sequence position has only one job assigned to it. Constraint (3.18) relates the starting times of the jobs at sequence positions $j-1$ and j according to the principle explained above. Constraint (3.19) defines the processing times of the jobs at each sequence position according to the assignment of the WWD to one of the machines he is able to operate. Note that these constraints are non-linear because of the product $X_r Z_{ij}$, but can be easily linearized by introducing new binary variables N_{rij} and adding the constraints (WILLIAMS, 2013):

$$N_{rij} \leq X_r, \qquad \forall r \in [m], i, j \in [n], \tag{3.20}$$

$$N_{rij} \leq Z_{ij}, \qquad \forall r \in [m], i, j \in [n], \tag{3.21}$$

$$N_{rij} \geq X_r + Z_{ij} - 1, \qquad \forall r \in [m], i, j \in [n]. \tag{3.22}$$

where the new variables capture the value of the product in a linear form.

3.2 Hybrid permutation flow shop insertion and scheduling problem

The formulation for the HPFSISP was based on that for the standard Hybrid Flow Shop proposed in Brah (1988). We use indices j and q for jobs, k for stages, l for machines and w for workers. The model uses dichotomous constraints for ordering the jobs in each stage. Table 3.2 describes the data and variable notation used.

Essentialy, the modification needed was to only allow a job to be scheduled in a second machine at the stage the WWDs are assigned to, disabling the parallel machines at other stages (constraint 3.26 below). The constant Q can be set to $\sum_{j \in [n]} \sum_{k \in [m]} max_{w \in [2]} d_{jkw}$.

Table 3.2 – Notation used in the HPFSISP model.

A	set of stages which the WWDs can operate
p_{jk}	processing time of job j at stage k when executed by regular workers
d_{jkw}	processing time of job j at stage k when executed by the WWD w
T_{jk}	processing time of job j at stage k after assigning the WWDs
U_{jkl}	1, if job j at stage k is assigned to machine l, 0 otherwise
P_{jq}	1, if job j precedes job q, 0 otherwise
C_{jk}	completion time of job j at stage k
X_k	1, if the WWDs are assigned to stage k, 0 otherwise
W_{wl}	1, if the WWD w is assigned to machine l in the dual-machine stage, 0 otherwise
Q	a sufficiently large constant

$$\textbf{min.} \quad C_{\max}, \tag{3.23}$$

$$\textbf{s.t.} \quad C_{\max} \geq C_{jm}, \qquad \forall j \in [n], \tag{3.24}$$

$$\sum_{l\in[2]} U_{jkl} = 1, \qquad \forall j \in [n], k \in [m], \tag{3.25}$$

$$U_{jk2} \le X_k, \qquad \forall j \in [n], k \in [m], \tag{3.26}$$

$$C_{jk} - T_{jk} \ge C_{j,k-1}, \qquad \forall j \in [n], k \in [m], \tag{3.27}$$

$$Q(2 - U_{jkl} - U_{qkl} + P_{jq}) + C_{jk} - T_{jk} \ge C_{qk}, \qquad \forall j,q \in [n], k \in [m], l \in [2], \tag{3.28}$$

$$Q(3 - U_{jkl} - U_{qkl} - P_{jq}) + C_{qk} - T_{qk} \ge C_{jk}, \qquad \forall j,q \in [n], k \in [m], l \in [2], \tag{3.29}$$

$$T_{jk} = p_{jk}(1 - X_k) + \sum_{l\in[2]} (d_{jkw} X_k W_{wl}), \qquad \forall j \in [n], k \in [m], l,w \in [2], \tag{3.30}$$

$$\sum_{k\in A} X_k = 1, \tag{3.31}$$

$$\sum_{l\in 2} W_{wl} = 1, \qquad \forall w \in [2], \tag{3.32}$$

$$\sum_{w\in 2} W_{wl} = 1, \qquad \forall l \in [2], \tag{3.33}$$

$$C_{jk} \ge 0 \qquad \forall j \in [n], k \in [m]. \tag{3.34}$$

The objective function (3.23) is defined as the latest completion time in constraint (3.24). Constraint (3.25) requires that all jobs are assigned to only one of the machines at each stage. Constraint (3.26) guarantees that only the stage to which the WWDs have been assigned can use a second, parallel machine. Constraint (3.27) ensures that each job can only start on a stage after it has finished on the previous one. Constraints (3.28) and (3.29) prevent any two jobs to be executed on the same machine at the same time. The processing time of a job depends on the stage the WWDs were assigned to, and is defined in constraint (3.6). Restriction (3.31) requires that the disabled workers are assigned to one of the stages they can operate. Finally, constraints (3.32) and (3.33) require that each worker is assigned to exactly one of the parallel machines and each machine has only one worker assigned to it.

Again, the presented model is not linear due to the term $X_k W_{wl}$ in constraint (3.30), but can be linearized in the same way explained for the TS3 model in Section 3.1.2.2.

4 HEURISTICS FOR THE PFSISP AND THE HPFSISP

In this chapter, heuristic solutions are presented to solve the new problems. Heuristics are not guaranteed to find optimal solutions, but they can speed up the process of finding a satisfactory solution when finding an optimal one is impractical due to the large search space.

To solve a worker insertion problem we must find the best stage for the WWDs and an optimal schedule of the operations. For a single worker, a simple approach is to solve a standard PFSSP for each possible insertion in one of the m stages. This makes it possible to use existing methods to solve each subproblem, but ignores that some stages may be better for inserting the worker than others, and thus should receive more search time. We propose a pooled strategy to solve this problem. The same approaches work when inserting a pair of workers, but we additionally have to solve the sub-problem of scheduling the jobs on the stage with two parallel machines.

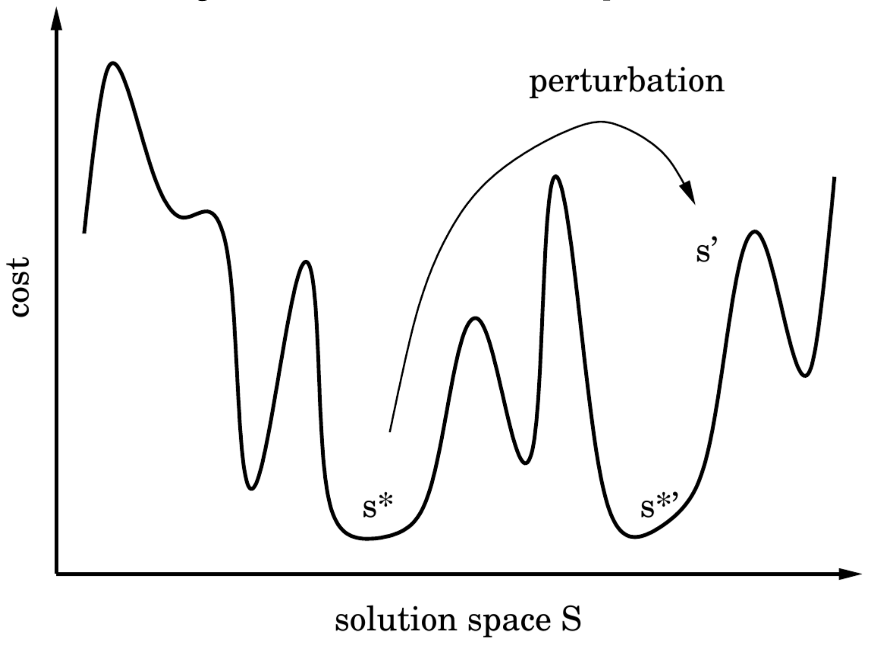
We have chosen to base our heuristic on an iterated greedy algorithm (IGA), a special form of an iterated local search (RUIZ; STÜTZLE, 2007). Both methods were successful in finding good schedules for the permutation flow shop and related problems. For most of these problems they are, or are part of, the current best heuristics (see e.g. Dubois-Lacoste, López-Ibáñez and Stützle (2011), Pan and Ruiz (2012), Fernandez-Viagas and Framinan (2014)). In the following, the heuristics are described in more detail.

4.1 Iterated Local Search

A local search is a heuristic optimization algorithm that searches the solution space of a problem by applying a modification to a candidate solution. The set of solutions resulting of a modification form the neighbourhood of a solution. The local search moves to better neighbours until the solution can't be improved and a local minimum is reached (considering a minimization problem).

Iterated Local Search is a high-level heuristic, or *metaheuristic* that iterates over local search solutions. It can be seen as a search in the local minimum space, as shown in Figure 4.1. In order to be able to jump from one local minimum to another, a large enough modification in the current solution is needed. This modification is called a *perturbation*. The way this perturbation is applied defines the neighbourhood between the local minimum.

Figure 4.1 – Iterated Local Search operation



Source: Lourenço, Martin and Stützle (2010)

This simple idea is old is also known as *iterated descent* or *chained local optimization* (LOURENÇO; MARTIN; STÜTZLE, 2010). Algorithm 1 presents the generic idea of an Iterated Local Search.

Algorithm 1 Generic Iterated Local Search

```
Input: A solution $s$.
Output: An improved solution $s'$
1: $s \leftarrow$ localSearch($s$)
2: while termination criterion not satisfied do
3:     perturb the current solution $s$ to obtain $s'$
4:     $s' \leftarrow$ localSearch($s'$)
5:     if accept($s, s'$) then
6:         $s \leftarrow s'$
7:     end if
8: end while
9: return the best solution $s'$ found during the search
```

4.2 The NEH Algorithm

The IGA takes as input an initial solution, a job sequence π for the PFSSP. One way to obtain a good initial solution is to use a constructive heuristic. We choose to use a variant of NEH (NAWAZ; ENSCORE; HAM, 1983), one of the best known constructive heuristics for the PFSSP to date. The idea is to order the jobs by non-increasing total processing time (sum of all times of a job) and insert each job in the best possible position at each step of the permutation construction. Algorithm 2 explains NEH in more detail.

Algorithm 2 NEH Algorithm

```
Input: Processing times $p_{ij}$ of the $n$ jobs on $m$ machines.
Output: Constructed permutation $\pi$.
 1: for all $j \in [n]$ do
 2:     compute $P_j = \sum_{i=1}^{m} p_{ij}$
 3: end for
 4: sort the jobs in non-increasing order of $P_j$, getting the initial sequence $J_1, J_2, \ldots, J_n$
 5: evaluate two partial sequences $J_1, J_2$ and $J_2, J_1$ and choose the one with the smaller
    makespan
 6: for all $k \in \{3, \ldots, n\}$ do
 7:     insert $J_k$ in the $k$ possible places of the partial solution with k-1 jobs
 8:     evaluate the $k$ partial sequences
 9:     choose the sequence $\pi$ with smallest makespan to be the $k$-th partial solution
10: end for
11: return $\pi$
```

This is the original NEH algorithm. Considering NEH as a case of a family of heuristics, there are several options that can affect the performance of the algorithm: the starting order, i.e. how to arrange the jobs in the first step (line 4), the sequence generation, i.e. how the candidate (sub)sequences are generated, and the tie-breaking mechanism, i.e. how to handle ties in the makespan evaluation of the (sub)sequences (NEH does not define tie-breaking rules).

Kalczynski and Kamburowski (2008) proposed an improved NEH heuristic where they define the weighted times $\hat{a}_j$ and $\hat{b}_j$ as follows:

$$\hat{a}_j = \sum_{i=1}^{m} [(m-1)\frac{m-2}{2} + m - i]p_{ij} \tag{4.1}$$

$$\hat{b}_j = \sum_{i=1}^{m} [(m-1)\frac{m-2}{2} + i - 1]p_{ij} \tag{4.2}$$

The initial order is given by sorting the jobs by their non-increasing c_j, where $c_j = \hat{a}_j$ if $\hat{a}_j \leq \hat{b}_j$, or $c_j = \hat{b}_j$ if $\hat{a}_j > \hat{b}_j$. The sequence generation is the same as the original NEH but when ties occur during the insertion candidates evaluation, the best candidate is the first index for which the minimum is achieved if $\hat{a}_j \leq \hat{b}_j$, or the last last index if $\hat{a}_j > \hat{b}_j$. We decided to use this version of NEH in our iterated greedy algorithm since it is better than the original and has performance very similar to the best known (FERNANDEZ-VIAGAS; FRAMINAN, 2014). We refer to this algorithm as NEH_{KK}.

4.2.1 Complexity of NEH algorithm

In line 8 of Algorithm 2, it takes $O(mk)$ to calculate the makespan of each partial sequence. The for loop can evaluate up to $n(n+1)/2-1 = O(n^2)$ schedules, totalling a computational complexity of $O(mn^3)$.

Taillard (1990) introduced optimizations to the NEH algorithm that accelerate the makespan calculations. These optimizations uses data structures to store information that is needed more than once during the partial sequences insertions, since the times before a insertion position remains the same. This reduces the complexity of computing the makespan of all partial sequences in a step of the for loop in line 6 to $O(mk)$, giving a final complexity of $O(mn^2)$. The data structures used are (when job J_k is about to be inserted in a partial sequence):

- e_{ij}, the earliest completion time of job J_j at machine M_i (also called *head*):

$$e_{ij} = \max\{e_{i-1,j}, e_{i,j-1}\} + p_{ij}, \qquad e_{0j} = e_{i0} = 0, i \in [m], j \in [k-1];$$

- q_{ij}, the interval between the latest start time of job J_j at machine M_i and the total completion time (also called *tail*):

$$q_{ij} = \max\{q_{i+1,j}, q_{i,j+1}\} + p_{ij}, \quad q_{m+1,j} = q_{ik} = 0, i \in \{m,\dots,1\}, j \in \{k-1,\dots,1\};$$

- f_{ij}, the earliest completion time of job J_j at machine M_i after it is inserted at position j (also called *relative head*):

$$f_{ij} = \max\{f_{i-1,j}, e_{i,j-1}\} + p_{i,J_k}, \qquad f_{0,j} = 0, i \in [m], j \in [k].$$

- C_j: the makespan of the partial sequence after J_k is inserted at place j:

$$C_j = \max_i\{f_{ij} + q_{ij}\}, i \in [m], j \in [k].$$

4.3 The Iterated Greedy Algorithm

The IGA works by performing repeated perturbations on the solution: it destroys and reinserts jobs of a given permutation, followed by a local search. Each iteration of the IGA can be divided in four phases:

- *Destruction Phase:* d, a number given as parameter, jobs are randomly chosen without repetition and then are removed from π in the order they were taken;
- *Construction Phase:* The d jobs removed are inserted back again one by one in the best possible position of π;
- *Local Search:* The resulting solution from the construction phase is improved by a local search. A *insertion neighbourhood* is used: the neighbours of a solution are all the permutations obtained removing a job and inserting it in all possible positions. The local search only allow modifications that improve the current solution, and the strategy for choosing the new solution is *first improvement*, i.e. it selects the first neighbour that improves the current solution. The local search can be seen in Algorithm 3.
- *Acceptance Criterion:* A simulated annealing-like acceptance criterion is used, where worse solutions can be accepted according to a constant probability given as a parameter named T. The worse solutions are used for diversification and to avoid stagnation.

Algorithm 3 Local search algorithm.

```
Input: Job permutation $\pi$.
Output: $\pi$ improved to a local minimum.
1:  improve $\leftarrow$ true
2:  while improve do
3:      improve $\leftarrow$ false
4:      for all $i \in [n]$ do
5:          $k \leftarrow$ random job removed from $\pi$ without repetition
6:          $\pi' \leftarrow$ best permutation obtained by inserting $k$ in any possible positions of $\pi$
7:          if $C_{max}(\pi') < C_{max}(\pi)$ then
8:              $\pi \leftarrow \pi'$
9:              improve $\leftarrow$ true
10:         end if
11:     end for
12: end while
13: return $\pi$
```

Algorithm 4 presents the IGA for the PFSSP and Figure 4.2 presents an example of an iteration of this algorithm. The temperature used in the acceptance criterion depends on the instance size and is computed as

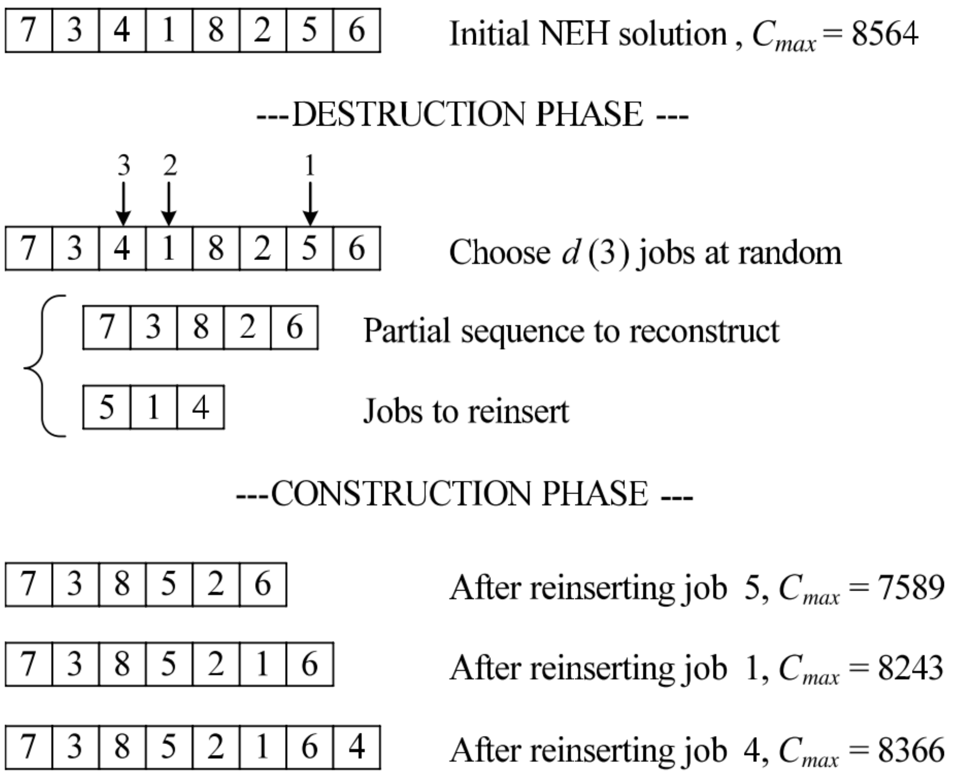
$$Temperature = T \cdot \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}}{n \cdot m \cdot 10}$$

Algorithm 4 Iterated Greedy Algorithm. *random* is a random number distributed uniformly in [0,1].

```
Input: A permutation schedule $\pi$.
Output: An improved permutation schedule $\pi_b$.
 1: $\pi \leftarrow$ improve solution with Algorithm 3
 2: $\pi_b \leftarrow \pi$
 3: while termination criteria not satisfied do
 4:     $\pi' \leftarrow \pi$
 5:     remove $d$ random jobs from $\pi'$ and insert in $\pi'_R$
 6:     for all $i \in [d]$ do
 7:         $\pi' \leftarrow$ best permutation obtained by inserting $\pi'_R[i]$ in any possible positions of $\pi'$
 8:     end for
 9:     $\pi'' \leftarrow$ improve solution with Local Search (Algorithm 3)
10:     if $C_{max}(\pi'') < C_{max}(\pi)$ then
11:         $\pi \leftarrow \pi''$
12:         if $C_{max}(\pi'') < C_{max}(\pi_b)$ then
13:             $\pi_b \leftarrow \pi$
14:         end if
15:     else if random $\leq exp\{-(C_{max}(\pi'') - C_{max}(\pi))/Temperature\}$ then
16:         $\pi \leftarrow \pi''$
17:     end if
18: end while
19: return $\pi_b$
```

A characteristic that makes the heuristic faster is that it uses the Taillard (1990) optimizations as explained in Section 4.2.1. The optimizations are used to calculate the makespan after each new insertion, and are used in the NEH heuristic, in the local search and also in the construction phase of the IGA.

Figure 4.2 – Example of an iteration of the IGA


7 3 4 1 8 2 5 6
Initial NEH solution , $C_{max} = 8564$
---DESTRUCTION PHASE ---
3 2 1
Choose d (3) jobs at random
7 3 8 2 6
Partial sequence to reconstruct
5 1 4
Jobs to reinsert
---CONSTRUCTION PHASE ---
7 3 8 5 2 6
After reinserting job 5, $C_{max} = 7589$
7 3 8 5 2 1 6
After reinserting job 1, $C_{max} = 8243$
7 3 8 5 2 1 6 4
After reinserting job 4, $C_{max} = 8366$

Source: Ruiz and Stützle (2007)

4.4 A Pooled IGA for inserting WWDs into flow shops

Remembering that for the flow shop insertion variants, besides the permutation, we also need to assign the WWDs to one of the machines or stages. To find this assignment, we propose Algorithm 5, a pooled variant of an IGA. Firstly, a pool with m solutions is created. For each candidate solution, the WWDs are assigned to one of the m stages and the NEH_{kk} heuristic is applied to obtain an initial solution. Then, the algorithm proceeds in m phases. In each phase, the IGA is applied to each candidate solution for a fixed time t and then the solution of worst makespan in the pool is discarded. The total running time is therefore $m(m+1)t/2$, and the kth best solution receives $(m+1-k)t$ of this time. This ensures that solutions with a shorter makespan receive more time than the those that get stuck early. This approach is preferable to more complex methods like that of Benavides, Ritt and Miralles (2014), because we are inserting only a few WWDs. Similar approaches have been applied successfully for selecting the best among several sets of parameters in racing methods (LÓPEZ-IBÁNEZ et al., 2011) and to improve local search in the “Go with the winners” heuristic (ALDOUS; VAZIRANI, 1994). In Algorithm 5, function $NEH_{KK}(k)$ returns an initial solution using the NEH_{KK} algorithm, when assigning the WWDs to the kth stage, and function $IGA(\pi,t)$ improves the current schedule π

by applying the IGA for time t.

Algorithm 5 A pooled IGA for PFSISP or HPFSISP.

```
Output: A solution $(\pi,k)$ for the PFSISP or HPFSISP .
 1: $P$: pool of solutions
 2: for all $k \in [m]$ do
 3:     $P \leftarrow P \cup \{(NEH_{KK}(k),k)\}$          ▷ Create the solution pool
 4: end for
 5: while $|P| > 0$ do
 6:     for all $(\pi,k) \in P$ do
 7:         $(\pi,k) \leftarrow (IGA(\pi,t),k))$          ▷ Algorithm 4
 8:     end for
 9:     $(\pi_0,k_0) \leftarrow \text{argmax}_{(\pi,k)\in P} C_{max}(\pi)$
10:     $P \leftarrow P \setminus \{(\pi_0,k_0)\}$
11: end while
12: return the latest solution $\pi_0,k_0$ removed from the pool
```

The makespan calculation adjustments for the PFSISP are trivial, we simply need to use the correct processing time, depending on whether the WWD is assigned to a machine or not. For the HPFSISP, we also have to perform the scheduling of the jobs on the parallel machines of the stage at which the WWDs were assigned. This subproblem is treated in the next section.

4.5 Solving the two-machine subproblem

A solution for 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 (ATTAR; MOHAMMADI; TAVAKKOLI-MOGHADDAM, 2013). For a permutation π of the jobs, and a assignment of the WWDs to stage k, heads are defined as $e_j = e_{k-1,j}$ and tails $q_j = q_{k+1,j}$, where e_{ij} is the earliest completion time of job j on stage i, as defined in Section 4.2.1 and q_{ij} is the shortest time from the start of job j on stage i to the completion of the last operation, also defined in Section 4.2.1. Then we have to find starting times S_j for the jobs $j \in [n]$ on the two machines, such that $S_j \geq e_j$, minimizing $C_{\max} = \max_j\{S_j + q_j\}$. Since we impose the order of the permutation flowshop π on all machines the problem reduces to finding an optimal assignment of the jobs to the parallel machines.

This problem is NP-hard, since it generalizes $P2||C_{max}$ (LENSTRA; KAN; BRUCKER, 1977), but can be solved by dynamic programming (DP) in time $O(n\overline{C}^2)$ for some upper bound $\overline{C}$ of the makespan. It works by processing the jobs in order of non-decreasing heads, breaking ties by longer tails. In each step, two sub-problems are generated when assigning the current job to one of the two machines, and updating current times of the two machines accordingly. Let $C(j,t_1,t_2)$ be the minimal completion time when scheduling jobs $j,...,n$ on the two parallel machines, starting not earlier than t_1 on the first, and not earlier than t_2 on the second machine. Then the optimal solution is given by $C(1,0,0)$, where

$$\begin{aligned}C(j,t_1,t_2) = \min\{ &\max\{C_1(t_1,j)+q_j, C(j+1,C_1(t_1,j),t_2)\},\\ &\max\{C_2(t_2,j)+q_j, C(j+1,t_1,C_2(t_2,j))\}\}.\end{aligned}$$

and $C_l(t,j) = \max\{t,e_j\}+p_{jl}$ is the completion time of job J_j when starting not earlier than t on parallel machine l. The base case is $C(n+1,t_1,t_2) = 0$ for all t_1 and t_2.

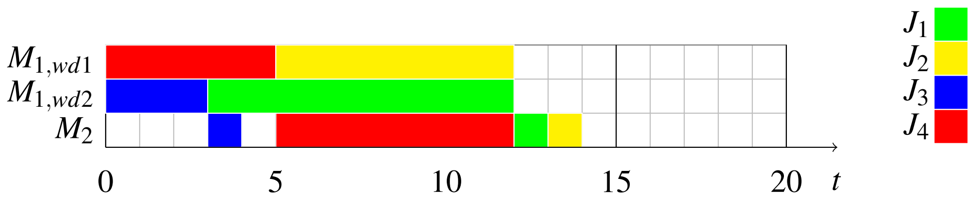
The solution by DP can be very expensive for large instances, therefore, we also decided to use a heuristic to solve this subproblem. The heuristic processes the jobs in order, and greedily assigns each job to the parallel machine that results in the earliest completion time (tail).

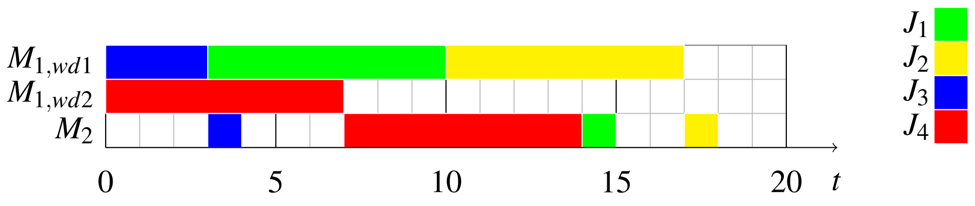
Figure 4.3 shows an example of the best schedule found by the two approaches for a given permutation. We can see that the optimal makespan found by the DP is 14 while the heuristic finds a makespan of 18. This difference happens because the first job J_3 is allocated to the second machine of the first stage in the DP, while the heuristic greedily assigns it to the first machine since the tails are the same.

To summarize, we have two solutions for the dual machine stage: an exact solution using DP and an earliest completion time heuristics.

Figure 4.3 – Example of an instance with the schedules found by the DP and by the earliest completion time heuristic for the job permutation sequence $3,4,1,2$

	Regular		WD 1		WD 2	
Job	M_1	M_2	M_1	M_2	M_1	M_2
J_1	1	1	7	∞	9	∞
J_2	1	1	7	∞	10	∞
J_3	1	1	3	∞	3	∞
J_4	1	7	5	∞	7	∞





5 COMPUTATIONAL EXPERIMENTS

In this chapter the results of computational tests are presented. We first analyze the performance of the models. Then, the heuristics are compared on small instances that can be solved to optimality. In a second experiment we evaluate the heuristics on instances of practical sizes.

The computational experiments compare the performance of different methods on each problem, and help to study the benefit of inserting WWDs into traditional flow shops. We first propose test instances in Section 5.1 and then present the numerical results in Section 5.2.

5.1 Test instances and experimental methodology

For the computational experiments we created instances for the inclusion problem based on the well-known flow shop instances proposed by Carlier (1978) and Taillard (1993). The Carlier benchmark is composed of eight small instances that are relatively simple to solve with processing times ranging from 1-999, while the Taillard benchmark contains 120 instances divided into groups of 10 that are more difficult to solve, with processing times randomly generated in the range [1,99]. From the Taillard benchmark we used the first 60 instances with sizes up to 50 jobs and 20 machines. Taillard's benchmark has instances based on real dimensions of industrial problems. Table 5.1 reports the size of the instances used.

Table 5.1 – Size of the Carlier and Taillard instances

Inst.	n	m	Group	n	m
car1	11	5	ta01	20	5
car2	13	4	ta02	20	10
car3	12	5	ta03	20	20
car4	14	4	ta04	50	5
car5	10	6	ta05	50	10
car6	8	9	ta06	50	20
car7	7	7			
car8	8	8			

We assume that the processing times p_{ij} of a flow shop instance are those of a regular worker. To model a WWD, we modify these processing times in two ways. First, a fixed percentage of incompatibilities is introduced. An incompatibility models the case of a worker who is unable to operate some machine, as, for example, machine 4 for the WWD in the instance given in Figure 2.5. Second, the processing times are increased to reflect that a WWD usually needs more time to execute a job. Based on experiences made with workers in SWDs, we opted to produce instances with no incompatibilities, as well as 10% and 20% of incompatibilities per worker. The processing time p of a regular worker for some job on some machine is increased by choosing uniformly at random a processing time in the interval $[p,2p]$ or $[p,5p]$.

For the mathematical models, we limited our tests to the Carlier instances, and the first group of ten Taillard instances with 20 jobs and 5 machines, since the larger instances are hard to solve exactly even in the regular case. The heuristics were run for the first sixty Taillard instances.

With three levels of incompatibilities and two levels of task time variation, we obtain a total of 408 test instances.

5.2 Numerical results

We solved the instances described above using the models presented in Section 3, two IGA based heuristics, as well the DP and heuristic solution for the hybrid variation subproblem. To further compare the quality of the heuristic solutions, we run the tests for the PFSISP instances with the exact method LOMPEN (COMPANYS; MATEO, 2007), a branch-and-bound algorithm for the PFSSP.

LOMPEN is a branch-and-bound algorithm that uses the reversibility property: a permutation and the inverse of this permutation have the same makespan (the inverse of a permutation is formed by the jobs of the permutation in reversed order). LOMPEN use this property to apply simultaneously branch-and-bound algorithms to both direct and inverse instances, sharing lower and upper bounds between them. In our experiments, each machine assignment of the WWD was solved separately and then the best solution was collected.

The mathematical models have been solved using the commercial solver CPLEX 12.5 running with a single thread and a time limit of one hour. The proposed heuristics were implemented in C++, and compiled with GNU C++ 4.7.3 with optimization level 3 (-O3). All computational tests were executed on a PC with an Intel Core i7 processor running at 2.8 GHz, and with 12 GB of main memory.

5.2.1 Comparison of the mathematical models

5.2.1.1 Results of the FSISP and PFSISP models

The results for the FSISP and the two PFSISP models described in Section 3.1 are presented in Table 5.2 for the Carlier instances and in Table 5.3 for the Taillard instances. Each table compares the overall averages of the three models for each of the two time variations, and each level of incompatibilities. In the tables model (3.2)–(3.7) is denoted by FS. For each combination we report the average percentage relative deviation (APRD) between the lower and upper bounds found by CPLEX ($\overline{Gap\%}$), the average running time taken by CPLEX (Time), and the number of optimum solutions found for each group of instances (Opt).

Table 5.2 – Results of the models on the Carlier instances.

		FS			LYeq			TS3		
Var.	Inc.	Gap	Time	Opt	Gap	Time	Opt	Gap	Time	Opt
2	0	14.25	2168.7	4	5.31	1261.9	6	0.00	26.8	8
2	10	14.08	1971.8	4	3.86	1122.3	6	0.00	17.8	8
2	20	13.92	2162.5	4	4.32	1191.8	6	0.00	14.6	8
5	0	25.04	2267.8	3	14.79	1440.9	5	0.00	55.7	8
5	10	17.19	2276.8	3	10.91	1453.0	5	0.00	46.7	8
5	20	12.06	2279.5	3	8.54	1423.8	5	0.00	11.3	8

Table 5.3 – Results of the models on the Taillard instances.

		FS			LYeq			TS3		
Var.	Inc.	Gap	Time	Opt	Gap	Time	Opt	Gap	Time	Opt
2	0	60.02	-	0	52.72	-	0	0.86	1861.7	7
2	10	59.84	-	0	52.78	-	0	0.03	2065.4	9
2	20	58.44	-	0	51.63	-	0	0.00	1301.8	10
5	0	67.78	-	0	64.15	-	0	1.64	2496.1	7
5	10	65.12	-	0	63.55	-	0	1.20	2182.8	7
5	20	58.83	-	0	62.34	-	0	1.90	2557.5	5

The FS model had the worst performance, solving about half of the Carlier instances in an average of 36 minutes and none of the Taillard instances. The LYeq model is clearly better, solving two more Carlier instances in about 60% of the time of the FS model, but was still unable to solve any of the Taillard instances. The relative performance of the two models was expected, since both use dichotomous constraints for defining the job order, but the FS model solves a harder problem. The TS3 model performed much better than the other models, solving all Carlier instances in a small fraction of the time and being able to solve 75% of the Taillard instances. Thus, the TS3 model for the PFSSP continues to be the strongest model for PFSISP, even with the additional overhead from linearizing constraints (3.19). We can also see that an increased time variation makes the problem in general more difficult to solve, while increasing the incompatibilities makes it easier, since it reduces the number of feasible assignments of the WWDs to machines.

5.2.1.2 Results of the HPFSISP model

Table 5.4 presents the results for the HPFSISP model. We can see that the model solved about 80% of the Carlier instances and none of the Taillard instances. The time taken to solve the Carlier instances was more than 5 minutes in average and the gap for the Taillard instances was still very high, showing that the model cannot solve even the smallest group of instances. The observations about the problem difficulty with increasing incompatibilities stills applies for this variant.

Table 5.4 – Results of the HPFSISP model

		Carlier			Taillard		
Var.	Inc.	Gap	Time	Opt	Gap	Time	Opt
2	0	6.86	239.18	5	54.12	-	0
2	10	6.40	334.61	6	53.63	-	0
2	20	5.13	832.87	6	51.59	-	0
5	0	4.13	486.60	7	57.27	-	0
5	10	3.71	513.17	7	54.88	-	0
5	20	3.75	386.46	7	50.15	-	0

5.2.1.3 Comparison of permutation and non-permutation schedules

Table 5.5 presents the average makespan obtained by the non-permutation (FS) and the best permutation model (TS3) for the Carlier and Taillard instances. The numbers are averages over all instances of each group. We can see that for the Carlier instances, the non-permutation variant obtained best average results for all time and incompatibilities variations, but with an improvement of less than 1%. On the other hand, for the Taillard instances, the permutation variant was better in the majority of cases. This can be explained by the large gap between between the lower and upper bounds in the non permutation model, not being able to find good solutions in the time limit, since accordingly to Potts, Shmoys and Williamson (1991), the optimal solutions of non-permutation schedules should always be better than those of the permutation schedules.

Overall, we can see that a small increase of the makespan can be obtained using non-permutation schedules instead of permutation schedules. However, this small increase is still relevant in practice because the methods that solves the permutation problem are already very optimized and close to the optimum solution.

Table 5.5 – Comparison of the makespan obtained by the FS and TS3 models

		Carlier		Taillard	
Var.	Inc.	FS	TS3	FS	TS3
2	0	**8050.2**	8103.0	**1450.7**	1465.0
2	10	**8087.5**	8133.1	1475.8	**1427.4**
2	20	**8204.1**	8238.2	1474.9	**1443.7**
5	0	**13197.5**	13210.9	2541.5	**2521.7**
5	10	**13197.5**	13210.9	2547.2	**2540.9**
5	20	**13324.2**	13337.8	2678.2	**2673.9**

5.2.2 Results of the heuristics algorithms

We evaluated the pooled (P) IGA heuristic as well a simple variation (S) where the IGA is run for each possible machine assignment of the WWDs. As stopping criterion for the IGA based heuristics, we run tests using four variations of time limit (in milliseconds): $3nm$ and a longer version $3nm^2$ (identified as L in the results) and also $30nm$ and a longer counterpart $30nm^2$. This was done to compare if more time is really necessary for the assignments or if a shorter time can give similar results. For the IGA parameters, we used $d = 4$, $T = 0.4$, identified as the best in Ruiz and Stützle (2007).

To be able to evaluate the impact of inserting WWDs, the solution quality is reported as the APRD ($(C_{max} - C^*_{max})/C^*_{max} \cdot 100$) from the best known makespan C^*_{max} of the corresponding flow shop instance (called "Rd." in the results). For example, if the best solution for the original flow shop instance has a makespan of 100, and our heuristic finds a solution with a makespan of 99 when inserting a disabled worker, the result is reported as $-1(\%)$, indicating a decrease when compared to the original problem. Due to the heuristic non-determinism, we report averages over 5 replications with different seeds. All times are reported in seconds.

5.2.2.1 Experiments with the Carlier instances

On the Carlier instances, we present the CPLEX results of the TS3 model for the PFSISP and the model of section 3.2 for the HPFSISP. For the PFSISP we also present the LOMPEN results.

Table 5.6 reports the relative deviations when inserting one worker, and Table 5.7 when inserting two workers. For the exact methods, we also report the solution time ($\bar{t}$), and for the heuristics, the initial solution obtained by NEH_{KK}. All instances when inserting one worker could be solved to optimality. When inserting two workers, the optimality gap (Gap) is shown when no optimal solution is found in one hour.

The PFSISP is easy to solve on the Carlier instances. LOMPEN finds the optimum very quickly, and the heuristics also find the optimal solutions even in the version with the shortest time limit (less than 1 second). The constructive heuristic NEH_{KK} gives solutions at most 1.3% from the optimum. The HPFSISP is harder to solve, and the average solution time of the CPLEX solver increases by a factor of almost 40, and the NEH_{KK} starts with a solution up to about 9% worse. The heuristics find good solutions with a makespan that is at most 0.4% longer in 1/500th of the time.

Table 5.6 – Results for Carlier instances when inserting one worker

		Exact				Heuristics								
		CPLEX		LOMPEN			$3nm$		$3nm^2$		$30nm$		$30nm^2$	
Var.	Inc.	$\bar{t}$	Rd.	$\bar{t}$	Rd.	*NEH*	S	P	SL	PL	S	P	SL	PL
2	0	26.7	7.4	0.1	7.4	8.5	7.4	7.4	7.4	7.4	7.4	7.4	7.4	7.4
2	10	17.8	7.9	0.1	7.9	9.2	7.9	7.9	7.9	7.9	7.9	7.9	7.9	7.9
2	20	14.5	9.2	0.1	9.2	10.2	9.2	9.3	9.2	9.3	9.2	9.3	9.2	9.2
5	0	55.7	75.8	0.0	75.8	76.2	75.8	75.8	75.8	75.8	75.8	75.8	75.8	75.8
5	10	46.7	75.8	0.0	75.8	76.2	75.8	75.8	75.8	75.8	75.8	75.8	75.8	75.8
5	20	11.3	77.7	0.0	77.7	78.1	77.7	77.7	77.7	77.7	77.7	77.7	77.7	77.7
Avg.		28.8	42.3	0.0	42.3	43.1	42.3	42.3	42.3	42.3	42.3	42.3	42.3	42.3

Table 5.7 – Results for Carlier instances when inserting two workers

		Exact			Heuristics										
		CPLEX				$3nm$		$3nm^2$			$30nm^2$		$30nm^2$		
Var.	Inc.	Gap	$\bar{t}$	Rd.	*NEH*	S	P	SL	PL	DP	S	P	SL	PL	DP
2	0	6.9	1499.5	-4.2	1.0	-4.1	-4.0	-4.1	-4.0	-3.9	-4.1	-4.0	-4.1	-4.0	-4.1
2	10	6.4	1151.0	-2.2	2.4	-2.1	-2.1	-2.1	-2.1	-1.9	-2.1	-2.1	-2.1	-2.1	-2.1
2	20	5.1	1524.7	-0.6	2.6	-0.6	-0.5	-0.6	-0.5	-0.4	-0.6	-0.5	-0.6	-0.5	-0.5
5	0	4.1	875.8	3.6	11.5	4.2	4.2	4.2	4.3	4.4	4.2	4.3	4.2	4.2	4.0
5	10	3.7	899.0	5.0	14.6	5.4	5.5	5.4	5.6	5.4	5.4	5.5	5.4	5.5	5.2
5	20	3.7	788.2	5.4	14.6	5.8	5.9	5.8	5.9	6.1	5.8	5.9	5.8	5.9	5.8
Avg.		5.0	1123.0	1.2	7.8	1.4	1.5	1.4	1.5	1.6	1.4	1.5	1.4	1.5	1.4

We can also see that there were very small differences between the different heuristic strategies and time limits. Tables 5.8 and 5.9 compares the number of iterations run by the IGA algorithm in each heuristic with the different time limits for one and two workers. The numbers are rounded averages over all instances in each group. We can see that the simple and pooled variant have a very similar number of iterations, with the simple version having slightly more. The DP on the other hand was not able to perform many iterations due to the increased calculation complexity, but nonetheless found solutions of good quality.

We can observe that inserting a single worker introduces, as expected, a large overhead, in particular for a high time variation, whereas the makespan does not increase more than 1.2% when inserting two workers.

Table 5.8 – Average number of iterations of the IGA on the Carlier instances when inserting one worker

		$3nm$		$3nm^2$		$30nm$		$30nm^2$	
Var.	Inc.	S	P	SL	PL	S	P	SL	PL
2	0	12053	10685	74279	62665	115940	110431	716972	683450
2	10	10015	8982	61589	55642	97153	92571	596361	582444
2	20	7992	7244	44515	44960	79739	73087	486529	479786
5	0	15188	14410	81270	87799	152227	146619	900621	917233
5	10	12596	11900	69787	75156	126541	120724	759788	766446
5	20	10026	9449	62518	62149	100628	93589	617590	623819
Avg.		11312	10445	65660	64728	112038	106170	679644	675530

Table 5.9 – Average number of iterations of the IGA on the Carlier instances when inserting two workers

		$3nm$		$3nm^2$			$30nm$		$30nm^2$		
Var.	Inc.	S	P	SL	PL	DP	S	P	SL	PL	DP
2	0	1607	1558	10590	10498	169	26905	26877	172607	179782	2170
2	10	1541	1163	8193	8139	141	20622	20592	133548	139812	1726
2	20	826	816	5647	5619	101	14321	14236	92541	94544	985
5	0	1584	1581	10578	10382	113	26230	26786	172571	184709	862
5	10	1240	1488	8056	8170	91	19822	20465	133341	143524	756
5	20	861	854	5564	5787	64	13773	13825	93220	99095	504
Avg.		1277	1243	8105	8099	113	20279	20463	132971	140244	1167

5.2.2.2 Experiments with the Taillard instances

We have repeated the experiments on Taillard's instances. We do not report results for CPLEX, which was unable to solve the larger instances, but report results for LOMPEN when inserting a single worker, which was able to solve all instances up to 10 machines and about 70% of the instances with 20 machines in two hours. We also do not report results for the heuristic variant computing an exact schedule on the parallel stage by dynamic programming, since it turned out to be too slow and thus found solutions of inferior quality.

Tables 5.10 and 5.11 present the results for the insertion of one and two workers, respectively. They report the APRD for each group of Taillard's instances of the same size (n and m), time variation (Var.) and each percentage of incompatibilities (Inc.), and additionally, when inserting one worker, the solution time of LOMPEN ($\bar{t}$).

We first look at the performance of the methods. Comparing the results of LOMPEN and the single worker case, we find that the heuristics again find very good solutions in a short time. The exact solver now has more difficulties, in particular for a time variation of $[p,2p]$ and runs about three hours whereas the heuristics run for at most 10 min, but the $3nm^2$ simple version already finds equal or better solutions than LOMPEN in at most one min. For a single worker the different strategies have only small impact on the solution quality. When inserting two workers, the choice of the heuristic strategy makes some difference: in the $3nm$ version, using a solution pool improves the solution by up to 0.6% for a time variation of $[p,5p]$ against the simple variant. When adding an extra m time this difference decreases to 0.2%. With even more time ($30nm$), the simple IGA managed to reach the pooled version, presenting very similar results, and adding extra m time improves the solution quality by about 0.3% for both variants.

Tables 5.12 and 5.13 report the number of iterations of the heuristics for one and two workers, respectively. The numbers are rounded averages over all instances in each group. We can see the simple IGA did more iterations than the pooled one in all cases, but still the pooled variant found slightly better results. Comparing the number of iterations when inserting one and two workers, the two worker variant is a factor 10 slower than the single one. This can be explained by the lack of the Taillard optimizations from Section 4.2.1 to speed up the makespan computations in the double insertion case, since the two machine subprolem requires a different makespan calculation. Further investigation is needed to check if it is possible to still apply the optimizations with this variant.

We now turn to the practical value of the solutions. As before, inserting a single worker has a visible overhead of about 12% for a small time variation. A large time variation with a single worker is unpractical, since it leads to about 90% overhead. In contrast, the disabilities of two workers can be hidden completely for a small time variation and never exceed 5% for large time variations. In summary, the insertion of WWDs if feasible for moderate time variations, independent of the percentage of incompatibilities.

Table 5.10 – Results for Taillard instances when inserting one worker.

				Exact		Heuristics								
				LOMPEN			$3nm$		$3nm^2$		$30nm$		$30nm^2$	
Var.	Inc.	n	m	$\bar{t}$	Rd.	*NEH*	S	P	SL	PL	S	P	SL	PL
2	0	20	5	0.4	14.7	16.1	14.7	14.7	14.7	14.7	14.7	14.7	14.7	14.7
2	0	20	10	3141.9	2.5	7.8	3.7	3.7	3.5	3.7	3.5	3.6	3.4	3.6
2	0	20	20	54258.4	1.2	4.0	1.5	1.3	1.2	1.3	1.2	1.3	1.2	1.3
2	0	50	5	6.9	27.4	27.5	27.4	27.4	27.4	27.4	27.4	27.4	27.4	27.4
2	0	50	10	127.0	19.2	19.7	19.4	19.4	19.4	19.3	19.3	19.3	19.2	19.3
2	0	50	20	21934.4	5.2	8.2	5.1	4.7	4.2	4.1	4.4	4.2	3.8	3.9
Avg.				13244.9	11.7	13.9	12.0	11.9	11.7	11.8	11.8	11.8	11.6	11.7
2	10	20	5	0.4	17.0	18.3	17.0	17.0	17.0	17.0	17.0	17.0	17.0	17.0
2	10	20	10	3795.5	3.5	7.8	3.7	3.8	3.5	3.6	3.5	3.7	3.5	3.7
2	10	20	20	48653.9	1.3	4.2	1.5	1.4	1.3	1.4	1.3	1.3	1.3	1.4
2	10	50	5	6.2	28.4	28.5	28.4	28.4	28.4	28.4	28.4	28.4	28.4	28.4
2	10	50	10	80.4	20.2	20.4	20.2	20.2	20.2	20.2	20.2	20.2	20.2	20.2
2	10	50	20	19483.3	5.5	8.3	5.6	5.0	4.6	4.5	4.7	4.6	4.3	4.4
Avg.				12003.3	12.6	14.6	12.7	12.6	12.5	12.5	12.5	12.5	12.4	12.5
2	20	20	5	0.4	18.3	19.5	18.3	18.3	18.3	18.3	18.3	18.3	18.3	18.3
2	20	20	10	1872.0	5.2	8.7	5.4	5.4	5.2	5.4	5.2	5.4	5.2	5.3
2	20	20	20	44304.2	1.3	4.2	1.5	1.4	1.3	1.3	1.3	1.3	1.3	1.4
2	20	50	5	4.7	28.7	28.8	28.7	28.7	28.7	28.7	28.7	28.7	28.7	28.7
2	20	50	10	76.4	20.5	20.7	20.6	20.6	20.5	20.6	20.5	20.6	20.5	20.6
2	20	50	20	16410.8	5.8	8.7	5.7	5.4	5.0	4.9	5.1	5.0	4.7	4.8
Avg.				10444.8	13.3	15.1	13.3	13.3	13.2	13.2	13.2	13.2	13.1	13.2
5	0	20	5	0.2	106.5	106.5	106.5	106.5	106.5	106.5	106.5	106.5	106.5	106.5
5	0	20	10	3.7	58.0	58.5	58.0	58.0	58.0	58.0	58.0	58.0	58.0	58.0
5	0	20	20	390.8	24.6	26.0	24.6	24.7	24.6	24.7	24.6	24.7	24.6	24.7
5	0	50	5	6.8	149.2	149.2	149.2	149.2	149.2	149.2	149.2	149.2	149.2	149.2
5	0	50	10	30.8	125.5	125.6	125.6	125.5	125.5	125.5	125.5	125.5	125.5	125.5
5	0	50	20	491.5	77.0	77.3	77.1	77.0	77.0	77.0	77.0	77.0	77.0	77.0
Avg.				154.0	90.1	90.5	90.2	90.2	90.1	90.2	90.1	90.2	90.1	90.2
5	10	20	5	0.2	108.3	108.4	108.3	108.4	108.3	108.4	108.3	108.4	108.3	108.4
5	10	20	10	3.6	58.0	58.5	58.0	58.0	58.0	58.0	58.0	58.0	58.0	58.0
5	10	20	20	389.5	26.6	27.9	26.6	26.6	26.6	26.6	26.6	26.6	26.6	26.6
5	10	50	5	5.8	154.7	154.7	154.7	154.7	154.7	154.7	154.7	154.7	154.7	154.7
5	10	50	10	27.5	125.5	125.7	125.5	125.5	125.5	125.5	125.5	125.5	125.5	125.5
5	10	50	20	476.9	77.8	78.0	77.8	77.8	77.8	77.8	77.8	77.8	77.8	77.8
Avg.				150.6	91.8	92.2	91.8	91.8	91.8	91.8	91.8	91.8	91.8	91.8
5	20	20	5	0.1	119.1	119.1	119.1	119.1	119.1	119.1	119.1	119.1	119.1	119.1
5	20	20	10	3.4	68.2	68.8	68.2	68.2	68.2	68.2	68.2	68.2	68.2	68.2
5	20	20	20	383.6	27.6	28.9	27.6	27.6	27.6	27.6	27.6	27.6	27.6	27.6
5	20	50	5	4.4	154.7	154.7	154.7	154.7	154.7	154.7	154.7	154.7	154.7	154.7
5	20	50	10	24.1	126.0	126.2	126.1	126.0	126.0	126.0	126.0	126.0	126.0	126.0
5	20	50	20	462.1	78.5	78.7	78.5	78.5	78.5	78.5	78.5	78.5	78.5	78.5
Avg.				146.3	95.7	96.1	95.7	95.7	95.7	95.7	95.7	95.7	95.7	95.7

Table 5.11 – Results for Taillard instances when inserting two workers.

					$3nm$		$3nm^2$		$30nm$		$30nm^2$	
Var.	Inc.	n	m	*NEH*	S	P	SL	PL	S	P	SL	PL
2	0	20	5	5.0	-3.7	-3.7	-3.8	-3.7	-3.8	-3.7	-3.8	-3.7
2	0	20	10	6.4	-0.7	-1.0	-1.3	-1.2	-1.4	-1.3	-1.5	-1.4
2	0	20	20	3.6	-0.2	-0.5	-0.9	-0.9	-0.9	-0.9	-1.1	-1.0
2	0	50	5	2.6	-1.3	-1.3	-1.3	-1.4	-1.4	-1.4	-1.4	-1.4
2	0	50	10	10.2	0.5	0.2	-0.5	-0.8	-0.6	-0.7	-1.2	-1.3
2	0	50	20	8.2	2.7	2.1	0.7	0.4	0.9	0.5	-0.1	-0.3
Avg.				6.0	-0.5	-0.7	-1.2	-1.3	-1.2	-1.2	-1.5	-1.5
2	10	20	5	5.0	-3.7	-3.6	-3.8	-3.7	-3.8	-3.7	-3.8	-3.7
2	10	20	10	6.6	-0.9	-0.8	-1.3	-1.3	-1.3	-1.2	-1.5	-1.3
2	10	20	20	3.7	-0.2	-0.4	-1.0	-1.0	-0.9	-0.9	-1.1	-1.1
2	10	50	5	3.0	-0.9	-1.0	-1.0	-1.0	-1.1	-1.1	-1.1	-1.1
2	10	50	10	10.3	0.8	0.3	-0.4	-0.6	-0.4	-0.6	-1.0	-1.1
2	10	50	20	8.3	2.7	2.1	0.8	0.3	0.9	0.6	-0.1	-0.3
Avg.				6.2	-0.4	-0.6	-1.1	-1.2	-1.1	-1.2	-1.4	-1.4
2	20	20	5	5.1	-2.0	-2.0	-2.1	-2.0	-2.1	-2.0	-2.1	-2.0
2	20	20	10	6.7	-0.5	-0.6	-1.1	-1.0	-1.1	-1.0	-1.3	-1.1
2	20	20	20	3.8	0.1	-0.4	-0.8	-0.8	-0.7	-0.9	-1.0	-1.0
2	20	50	5	4.0	-0.6	-0.7	-0.7	-0.7	-0.7	-0.7	-0.7	-0.7
2	20	50	10	10.8	1.3	0.9	0.0	-0.2	-0.1	-0.2	-0.6	-0.6
2	20	50	20	8.9	2.7	2.1	0.7	0.5	0.9	0.7	-0.1	-0.3
Avg.				6.5	0.1	-0.1	-0.6	-0.7	-0.6	-0.7	-1.0	-1.0
5	0	20	5	20.4	5.9	5.9	5.4	5.4	5.2	5.2	5.1	5.3
5	0	20	10	19.0	4.4	4.2	3.7	3.5	3.5	3.5	3.3	3.4
5	0	20	20	14.1	3.0	2.5	2.1	2.1	2.2	2.1	1.9	2.0
5	0	50	5	20.2	2.9	2.8	2.4	2.2	2.1	2.1	1.9	1.9
5	0	50	10	24.1	5.7	4.8	3.8	3.5	3.8	3.5	2.7	2.6
5	0	50	20	20.3	5.8	5.0	3.4	3.0	3.7	3.2	2.5	2.2
Avg.				19.7	4.6	4.2	3.5	3.3	3.4	3.3	2.9	2.9
5	10	20	5	20.8	5.9	5.8	5.4	5.4	5.2	5.2	5.1	5.2
5	10	20	10	19.0	4.6	4.1	3.6	3.6	3.5	3.6	3.3	3.5
5	10	20	20	14.2	3.1	2.7	2.1	2.1	2.2	2.1	1.9	2.0
5	10	50	5	20.2	3.1	2.9	2.3	2.2	2.1	2.1	2.0	2.0
5	10	50	10	24.7	6.4	5.6	4.5	4.1	4.3	3.9	3.4	3.2
5	10	50	20	20.5	6.1	5.0	3.5	3.0	3.9	3.2	2.5	2.2
Avg.				19.9	4.9	4.3	3.6	3.4	3.5	3.3	3.0	3.0
5	20	20	5	20.8	5.8	5.8	5.4	5.5	5.2	5.3	5.1	5.2
5	20	20	10	19.3	4.4	4.1	3.6	3.6	3.6	3.5	3.3	3.4
5	20	20	20	14.8	3.6	3.1	2.6	2.5	2.6	2.6	2.3	2.4
5	20	50	5	27.1	8.0	7.6	7.1	7.1	6.8	6.8	6.5	6.6
5	20	50	10	25.5	6.6	5.6	4.8	4.2	4.6	3.9	3.5	3.2
5	20	50	20	20.7	6.1	5.0	3.4	3.0	3.7	3.3	2.5	2.2
Avg.				21.4	5.7	5.2	4.5	4.3	4.4	4.2	3.9	3.8

Table 5.12 – Average number of iterations of the IGA on the Taillard instances when inserting one worker.

		$3nm$		$3nm^2$		$30nm$		$30nm^2$	
Var.	Inc.	S	P	SL	PL	S	P	SL	PL
2	0	4035	3423	41526	37013	52891	46011	570846	495379
2	10	3079	2646	37499	32640	46228	40206	507060	460024
2	20	2544	2240	32058	28535	39087	34109	440719	401540
5	0	5118	5200	61051	60996	71232	65880	825620	723665
5	10	4533	4276	53441	50117	61749	57135	731607	700503
5	20	3274	2990	44881	42421	52500	49133	637723	614744
Avg.		3764	3462	45076	41954	53948	48746	618929	565976

Table 5.13 – Average number of iterations of the IGA on the Taillard instances when inserting two workers.

		$3nm$		$3nm^2$		$30nm$		$30nm^2$	
Var.	Inc.	S	P	SL	PL	S	P	SL	PL
2	0	292	269	3608	3558	4484	3666	54300	50638
2	10	229	204	3182	3222	3597	3434	43392	43394
2	20	170	158	2614	2571	2690	2574	32474	32345
5	0	288	248	3946	4146	4486	4278	54057	53389
5	10	230	207	3411	3272	3585	3428	43324	38273
5	20	170	170	2521	2453	2686	2573	32403	32344
Avg.		230	209	3214	3204	3588	3325	43325	41730

6 CONCLUDING REMARKS

In this work we have studied two scheduling problems that aim to insert WWDs into flow shops. We have proposed mathematical formulations, heuristic solutions and a set of realistic test instances.

Our results show that our methods find close to optimal schedules for the inclusion of WWD into flow shops. From a practical point of view, our results show that the inclusion of WWDs can be optimized with standard techniques for moderate problem sizes. For larger flow shop instances, the heuristic becomes necessary to reach good solutions fast. The incompatibilities have almost no impact on the resulting makespan, such that workers which cannot operate some machines, but have almost regular processing times for the remaining machines can be integrated into a flow shop with virtually no overhead. When the processing time of the WWDs is about 50% slower in average, they can be included in a flow shop with a small overhead.

We can also conclude that a company that needs to integrate two WWDs in a flow shop, can duplicate the machine they are going to operate and even achieve a reduction in the makespan. However, the companies will have to consider the costs of purchasing and installing the new machine, as well the maintenance costs, in order to evaluate the real benefits of the integration. The governments could participate in this matter, providing some financial help, given that in some countries they are obligating a percentage of disabled workers. This would make the idea much more attractive for the companies.

This suggests further that companies can contribute to integrate people with disabilities in their production systems with no or only moderate losses in productivity. We hope this can lower the prejudice and help to increase the participation of persons with disabilities in the market and in society.

REFERENCES

ALDOUS, D.; VAZIRANI, U. “Go with the winners” algorithms. In: IEEE. **Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on**. [S.l.], 1994. p. 492–501.

ARAÚJO, F.; COSTA, A.; MIRALLES, C. Two extensions for the assembly line worker assignment, and balancing problem: parallel stations and collaborative approach. **Int. J. Prod. Econ.**, v. 140, p. 483–495, 2012.

ATTAR, S.; MOHAMMADI, M.; TAVAKKOLI-MOGHADDAM, R. Hybrid flexible flowshop scheduling problem with unrelated parallel machines and limited waiting times. **Int. J. Adv. Manuf. Technol.**, v. 68, n. 5-8, p. 1583–1599, 2013.

BENAVIDES, A. J.; RITT, M.; MIRALLES, C. Flow shop scheduling with heterogeneous workers. **Eur. J. Oper. Res.**, v. 237, n. 2, p. 713–720, 2014.

BLUM, C.; MIRALLES, C. On solving the assembly line worker assignment and balancing problem via beam search. **Comput. Oper. Res.**, v. 38, n. 2, p. 328–339, 2011.

BORBA, L. M.; RITT, M. A heuristic and a branch-and-bound algorithm for the assembly line worker assignment and balancing problem. **Comput. Oper. Res.**, v. 45, p. 87–96, may 2014. Online supplement: alwabp2.herokuapp.com/instances.

BRAH, S. A. **Scheduling in a flow shop with multiple processors**. Thesis (PhD) — University of Houston, 1988.

BRASIL. **Lei n^{o} 8123, de 24 de julho de 1991**. Presidência da República, 1991. Available online: <http://www.planalto.gov.br/ccivil_03/leis/l8213cons.htm>.

BRAULT, M. W. **Americans with disabilities: 2010**. [S.l.]: US Department of Commerce, Economics and Statistics Administration, US Census Bureau, 2012.

CARLIER, J. Ordonnancements a contraintes disjonctives. **R.A.I.R.O. Recherche operationelle/Operations Research**, v. 12, n. 4, p. 333–351, 1978.

COMPANYS, R.; MATEO, M. Different behaviour of a double branch-and-bound algorithm on $Fm \mid prmu \mid C_{\max}$ and $Fm \mid block \mid C_{\max}$ problems. **Comput. Oper. Res.**, v. 34, n. 4, p. 938–953, 2007.

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.

EMMONS, H.; VAIRAKTARAKIS, G. **Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications**. [S.l.]: Springer, 2013. (International Series in Operations Research & Management Science). ISBN 9781461451525.

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.

GAREY, M. R.; JOHNSON, D. S. **Computers and intractability: A guide to the theory of NP-completeness**. [S.l.]: Freeman, 1979.

GRAHAM, R. L.; LAWLER, E. L.; LENSTRA, J. K.; KAN, A. H. G. R. Optimization and approximation in deterministic sequencing and scheduling: a survey. **Ann. Discrete Math.**, v. 5, p. 287–326, 1979.

GUPTA, J.; STAFFORD, E. A comprehensive review and evaluation of permutation flowshop heuristics. **Eur. J. Oper. Res.**, v. 169, n. 3, p. 699–711, 2006.

GUPTA, J. N. Two-stage, hybrid flowshop scheduling problem. **Journal of the Operational Research Society**, JSTOR, p. 359–364, 1988.

HALL, L. A. Approximability of flow shop scheduling. **Math. Prog. B**, v. 82, n. 1–2, p. 175–190, 1998.

JOHNSON, S. M. Optimal two- and three-stage production schedules with setup times included. **Nav. Res. Logist.**, v. 1, n. 1, p. 61–68, 1954.

KALCZYNSKI, P. J.; KAMBUROWSKI, J. An improved NEH heuristic to minimize makespan in permutation flow shops. **Comput. Oper. Res.**, v. 35, p. 3001–3008, 2008.

LENSTRA, J. K.; KAN, A. R.; BRUCKER, P. Complexity of machine scheduling problems. **Annals of discrete mathematics**, Elsevier, v. 1, p. 343–362, 1977.

LIAO, C.; LIAO, L.; TSENG, C. 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. **J. Oper. Res. Soc.**, v. 43, n. 11, p. 1047–1054, 1992.

LÓPEZ-IBÁNEZ, M.; DUBOIS-LACOSTE, J.; STÜTZLE, T.; BIRATTARI, M. The irace package, iterated race for automatic algorithm configuration. **IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2011-004**, Citeseer, 2011.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search: Framework and applications. In: GENDREAU, M.; POTVIN, J.-Y. (Ed.). **Handbook of Metaheuristics**. [S.l.]: Springer US, 2010, (International Series in Operations Research & Management Science, v. 146). p. 363–397.

MIRALLES, C.; GARCIA-SABATER, J. P.; ANDRéS, C.; CARDOS, M. Advantages of assembly lines in Sheltered Work Centres for Disabled. A case study. **Int. J. Prod. Res.**, v. 110, n. 2, p. 187–197, 2007.

MIRALLES, C.; GARCIA-SABATER, J. P.; ANDRéS, C.; CARDOS, M. Branch and bound procedures for solving the assembly line worker assignment and balancing problem: Application to sheltered work centres for disabled. **Discrete Appl. Math.**, v. 156, n. 2, p. 352–367, 2008.

MIRALLES, C.; MARIN-GARCIA, J.; FERRUS, G.; COSTA, A. OR/MS tools for integrating people with disabilities into employment. a study on valencia's sheltered work centres for disabled. **Int. Trans. Oper. Res.**, v. 17, p. 457–473, 2010.

MOREIRA, M. C. O.; MIRALLES, C.; COSTA, A. M. Model and heuristics for the assembly line worker integration and balancing problem. **Computers & Operations Research**, Elsevier, v. 54, p. 64–73, 2015.

MOREIRA, M. C. O.; RITT, M.; COSTA, A. M.; CHAVES, A. A. Simple heuristics for the assembly line worker assignment and balancing problem. **J. Heuristics**, v. 18, n. 3, p. 505–524, 2012.

MUTLU, O.; POLAT, O.; AYCA, A. An iterative genetic algorithm for the assembly line worker assignment and balancing problem of type-ii. **Comput. Oper. Res.**, v. 40, n. 1, p. 418–426, jan. 2013.

NAWAZ, M.; ENSCORE, E.; HAM, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. **Omega**, v. 11, n. 1, p. 91–95, 1983.

PAN, C.-H. A study of integer programming formulations for scheduling problems. **J. Comput. Syst. Sci.**, v. 28, n. 1, p. 33–41, 1997.

PAN, Q.-K.; RUIZ, R. Local search methods for the flowshop scheduling problem with flowtime minimization. **Eur. J. Oper. Res.**, v. 222, p. 31–43, 2012.

PINEDO, M. The Lekin system. In: **Scheduling**. [S.l.]: Springer US, 2012. p. 615–621. ISBN 978-1-4614-1986-0.

PINEDO, M. **Scheduling: Theory, Algorithms, and Systems**. [S.l.]: Springer, 2012. (SpringerLink : Bücher). ISBN 9781461423614.

POTTS, C. N.; SHMOYS, D. B.; WILLIAMSON, D. P. Permutation vs. non-permutation flow shop schedules. **Oper. Res. Lett.**, v. 10, n. 5, p. 281–284, 1991.

POTTS, C. N.; STRUSEVICH, V. A. Fifty years of scheduling: a survey of milestones. **J. Oper. Res. Soc.**, v. 60, p. S41–S68, 2009.

RUIZ, R.; STÜTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. **Eur. J. Oper. Res.**, v. 177, n. 3, p. 2033–2049, 2007.

RUIZ, R.; VÁZQUEZ-RODRÍGUEZ, J. A. The hybrid flow shop scheduling problem. **Eur. J. Oper. Res.**, v. 205, n. 1, p. 1–18, 2010.

STAFFORD, E. F.; TSENG, F. T.; GUPTA, J. N. D. Comparative evaluation of MILP flowshop models. **J. Oper. Res. Soc.**, v. 56, n. 1, p. 88–101, jul. 2004. ISSN 0160-5682.

TAILLARD, E. Some efficient heuristic methods for the flow shop sequencing problem. **Eur. J. Oper. Res.**, v. 47, n. 1, p. 65–74, 1990.

TAILLARD, E. Benchmarks for basic scheduling problems. **Eur. J. Oper. Res.**, v. 64, n. 2, p. 278–285, 1993.

TANDON, M.; CUMMINGS, P.; LEVAN, M. Flowshop sequencing with non-permutation schedules. **Computers & chemical engineering**, Elsevier, v. 15, n. 8, p. 601–607, 1991.

TSENG, F. T.; STAFFORD, E. F. New MILP models for the permutation flowshop problem. **J. Oper. Res. Soc.**, v. 59, n. 10, p. 1373–1386, aug. 2007. ISSN 0160-5682.

VILA, M.; PEREIRA, J. A branch-and-bound algorithm for assembly line worker assignment and balancing problems. **Comput. Oper. Res.**, v. 44, p. 105–114, 2014.

WAGNER, H. M. An integer linear-programming model for machine scheduling. **Nav. Res. Logist.**, v. 6, n. 2, p. 131–140, 1959.

WHO. **World report on disability**. [S.l.]: World Health Organization, 2011.

WILLIAMS, H. P. **Model building in mathematical programming**. [S.l.]: John Wiley & Sons, 2013.