Scheduling in Flowshops with No-Idle Machines

Chapter · January 1970

DOI: 10.1007/978-3-642-02836-6_2

3 authors, including:

Rubén Ruiz
Universitat Politècnica de València
169 PUBLICATIONS **8,272** CITATIONS

SEE PROFILE

Eva Vallada
Universitat Politècnica de València
20 PUBLICATIONS **973** CITATIONS

SEE PROFILE

Scheduling in flowshops with no-idle machines

Rubén Ruiz, Eva Vallada, and Carlos Fernández-Martínez

Grupo de Sistemas de Optimización Aplicada. Instituto Tecnológico de Informática (ITI). Ciudad Politécnica de la Innovación, Edificio 8G. Acceso B. Universidad Politécnica de Valencia. Camino de Vera s/n, 46022 Valencia, Spain.
rruiz@eio.upv.es, evallada@eio.upv.es, cfernandez@iti.upv.es

Summary. This chapter deals with an interesting and not so well studied variant of the classical permutation flowshop problem with makespan criterion. In the studied variant, no idle time is allowed on machines. In order to ensure this no-idle constraint, the start times of jobs on machines must be delayed until all assigned jobs can be processed without incurring in idle times. This is a real situation arising in practice when expensive machinery is operated or when specific machines cannot be easily started and stopped due to technological constraints.
We provide a comprehensive characterization and modelization of the no-idle permutation flowshop, along with a detailed literature review. Existing methods are critically evaluated. We propose several improvements over existing approaches as well as adaptations of state-of-the-art algorithms that were proposed for related problems. An extensive computational campaign is conducted. Results are carefully analyzed by means of sound statistical techniques. The results indicate that the recent Iterated Greedy methods outperform existing algorithms by a significant margin.

1 Introduction

Flowshop scheduling is a very active field of research with close to 55 years of history. Flowshop problems are easy to formulate yet remarkably complex, both from a mathematical as well as from a computational point of view. In a flowshop, there is a set $N = 1, 2, \ldots, n$ of n unrelated product orders to produce. These are usually referred to as "jobs". The production shop is composed of a set $M = 1, \ldots, m$ of m machines that are disposed in series. Each job visits each machine in order. This order might be, without loss of generality, machine 1, machine 2 and so on until machine m. As a result of this, each job $j, j \in N$ is composed of m serial tasks, each one to be performed on a machine $i, i \in M$. The processing time of the tasks is referred to as $p_{j,i}$ which basically denotes the non-negative, known and deterministic processing time of job j at machine i.
The flowshop problem then consists of finding a production sequence of the

n jobs in the m machines so that a given performance criterion is optimized. The total number of feasible solutions to this problem is derived from the possible job's arrangements on machines. For each machine, we have $n!$ possible job permutations. Thus, the total number of feasible solutions or schedules is $(n!)^m$. However, this general case is seldom considered in the flowshop research field. Instead, a simplification is to consider a single permutation of jobs for all the machines. This brings the overall number of solutions down to $n!$ Under this simplification, the problem is referred to as permutation flowshop scheduling problem or PFSP in short.

In this chapter we study an interesting variant of this problem where no idle time is allowed on machines. As we will see, this results in a different problem. The chapter continues with detailed characterizations of both the regular as well as the no-idle flowshops. Later, the chapter also provides a detailed literature review in Section 2, along with a discussion of existing approaches, improvements over published methods and adaptations of high performing state-of-the-art algorithms in Section 3. A complete computational and statistical campaign is performed in Section 4. Finally, conclusions and suggestions for future research are given in Section 5.

1.1 Regular flowshop problem

Before going into details, a more formal definition of the PFSP is given. First of all, a number of assumptions are usually considered: (Baker, 1974):

- All jobs are independent and available for processing at time 0.
- All machines are continuously available.
- Each machine can process at most one job at a time and each job can be processed only on one machine at a time.
- The processing of a given job at a machine cannot be interrupted once started, i.e, no preemption is allowed.
- Setup times are sequence independent and are included in the processing times or are otherwise ignored.
- An infinite in-process storage buffer is assumed. If a given job needs an unavailable machine then it joins a queue of unlimited size waiting for that machine.

Most optimization criteria are based on the completion times of the jobs at the different machines which are denoted by $C_{j,i}$. Similarly, C_j denotes the time at which job j is completed at the last machine. The completion times $C_{j,i}$ can be easily calculated as follows:
Given a permutation π of n jobs, where $\pi_{(j)}$ denotes the job in the j-th position, the completion times are calculated in $\mathcal{O}(nm)$ with the following recursive expression:

$$C_{\pi_{(j)},i} = \max\left\{C_{\pi_{(j)},i-1}, C_{\pi_{(j-1)},i}\right\} + p_{\pi_{(j)},i} \tag{1}$$

where $C_{\pi_{(j)},0} = 0$ and $C_{\pi_{(0)},i} = 0$, $\forall i \in M, \forall j \in N$. The most common optimization criterion is the minimization of the maximum completion time or makespan (C_{max}) where $C_{max} = C_{\pi_{(n)},m}$. Under this objective, the PFSP is denoted as $F/prmu/C_{max}$ following the well known $\alpha|\beta|\gamma$ notation for scheduling problems given in Graham et al. (1979).

The earliest research papers on the PFSP focused on makespan minimization. The seminal paper of Johnson (1954) is widely recognized as the first study. However, a closer look brings even earlier papers, like the one of Salveson (1952).
Johnson mainly studied the PFSP with only two machines ($m = 2$) and provided a polynomial algorithm of $\mathcal{O}(n \log n)$ steps to solve this special case to optimality. The three or more machines problem is known to be $\mathcal{NP}$-Complete in the strong sense (Garey et al., 1976).

Exact approaches for the PFSP under makespan criterion (PFSP-C_{max}) are fairly effective, but only for a small number of jobs, and specially, machines. For the sake of completeness, and in order to completely characterize the PFSP, we introduce the following Mixed Integer Programming (MIP) model. Note that this is a well known model and certainly not the only possible one.

Decision variables:

$$X_{j,k} = \begin{cases} 1, \text{ if job } j \text{ occupies position } k \text{ of the sequence} \\ 0, \text{ otherwise} \end{cases}$$
$$j, k = \{1, \ldots, n\}$$
$$C_{k,i} = \text{ Completion time of job at position } k \text{ on machine } i$$
$$k = \{1, \ldots, n\}, i = \{1, \ldots, m\}$$

Objective function:

$$\min C_{max} = C_{n,m} \tag{2}$$

Constraints:

$$\sum_{k=1}^{n} X_{j,k} = 1, \quad j = \{1, \ldots, n\} \tag{3}$$

$$\sum_{j=1}^{n} X_{j,k} = 1, \quad k = \{1, \ldots, n\} \tag{4}$$

$$C_{k,1} \geq \sum_{j=1}^{n} X_{j,k} \cdot p_{j,1}, \quad k = \{1, \ldots, n\} \tag{5}$$

$$C_{k,i} \geq C_{k,i-1} + \sum_{j=1}^{n} X_{j,k} \cdot p_{j,i}, \quad k = \{1, \ldots, n\}, i = \{2, \ldots, m\} \tag{6}$$

$$C_{k,i} \geq C_{l,i} + \sum_{j=1}^{n} X_{j,k} \cdot p_{j,i}, \quad k = \{2, \ldots, n\}, l = \{1, \ldots, k-1\}, i = \{1, \ldots, m\} \tag{7}$$

$$C_{k,i} \geq 0, \quad k = \{1, \ldots, n\}, i = \{1, \ldots, m\} \tag{8}$$

$$X_{j,k} \in \{0, 1\}, \quad j, k = \{1, \ldots, n\} \tag{9}$$

We can see that minimizing C_{max} is equivalent to minimizing the completion time of the job in the last position of the sequence and on the last machine. Constraint sets (3) and (4) ensure that each position is occupied by exactly one job. Sets (5) and (6) control the completion times of all jobs in the first and on subsequent machines, making sure that these completion times are larger than those of previous machines. With constraint set (7) we ensure that completion times also take into account jobs in preceding positions on all machines. Finally, sets (8) and (9) define the nature of the decision variables.

The PFSP-C_{max} has been thoroughly studied in the literature. Here we provide just some of the most cited papers, like the heuristics by Page (1961), Palmer (1965), Campbell et al. (1970) or Dannenbring (1977). By far, the most known heuristic for the $F/prmu/C_{max}$ problem is the NEH by Nawaz et al. (1983). NEH is considered as the champion among heuristics, according to many studies like Turner and Booth (1987), Taillard (1990) and more recently, Ruiz and Maroto (2005). As a matter of fact, in the study of Ruiz and Maroto, NEH is confronted against more modern –and complex– heuristics like the ones of Koulamas (1998), Suliman (2000) and Davoud Pour (2001) and NEH is proved to perform better.
Apart from the review and computational evaluation of Ruiz and Maroto (2005), the reader might find additional valuable information in the reviews of Framinan et al. (2004) and Hejazi and Saghafian (2005).

Of course, C_{max} is not the only criterion studied. Total completion time, defined as $TCT = \sum_{j=1}^{n} C_j$, results in a $\mathcal{NP}$-Hard problem already for $m \geq 2$ (Gonzalez and Sahni, 1978). Furthermore, if there are no release times for the jobs, i.e., if $r_j = 0, \forall j \in N$, then the total or average completion time equals the total or average flowtime, denoted as F in the literature. Other studied criteria are those based in due dates. Given a due date d_j for job j, T_j denotes the tardiness of job j, which is defined as $T_j = \max\{C_j - d_j, 0\}$. Total tardiness minimization results in a $\mathcal{NP}$-Hard problem in the strong sense for $m \geq 2$ as shown in Du and Leung (1990). A recent review for the total tardiness version of the PFSP (the $F/prmu/\sum T_j$ problem) is given by Vallada et al. (2008). Lastly, there is a recent trend in which several objectives are jointly considered. A comprehensive review and evaluation of multiobjective approaches for PFSP is provided by Minella et al. (2008).

1.2 No-idle flowshop variant

In this chapter we are interested in a variant of the PFSP that arises when no idle time is allowed at machines. This constraint models an important practical situation that arises when expensive machinery is employed. Idling on such expensive equipment is often not desired. Clear examples are the steppers used in the production of integrated circuits by means of photolithography. Other examples come from sectors where less expensive machinery is used but where machines cannot be easily stopped and restarted. Ceramic roller kilns, for example, consume a large quantities of natural gas when in operation. Idling is not an option because it takes several days to stop and to restart the kiln due to a very large thermal inertia. In all such cases, idling must be avoided.

In order to better understand the no-idle constraint, we make use of an example problem with five jobs and four machines. The processing times $p_{j,i}$ are given in Table 1. The optimum solution, easily obtainable by complete

Table 1. Processing times for a PFSP example with four machines and five jobs.

	jobs (j)				
machines (i)	1	2	3	4	5
1	31	39	23	23	33
2	22	25	22	22	41
3	25	41	47	14	27
4	30	34	22	13	19

enumeration or by solving the corresponding instance of the model given in Section 1.1 is $\pi^*_{idle} = \{3, 1, 2, 5, 4\}$ and is depicted in Figure 1. It is straightforward to see that all machines, with exception of machine 1, have idle times. For example, there is an idle time on the second machine of 18 time units between the completion time of the job 1 in second position and the beginning of job 2 in the third position. These idle times are necessary because by the time job 1 is finished in machine 2, job 2 cannot start processing since it is still being processed in machine 1. Despite idle times, the makespan value is of 226 time units.

In the no-idle flowshop problem with makespan criterion, denoted as $F/prmu, no-idle/C_{max}$, these idle times are not allowed. In order to ensure this, and following the previous example, the start times of jobs 3, 1 and 2 on machine 2 need to be delayed so that no idle time is present in the schedule. The same previous sequence $\{3, 1, 2, 5, 4\}$ results in a makespan of value 258 if the no-idle constraint is enforced. As a result, the makespan value is more than 14% worse.

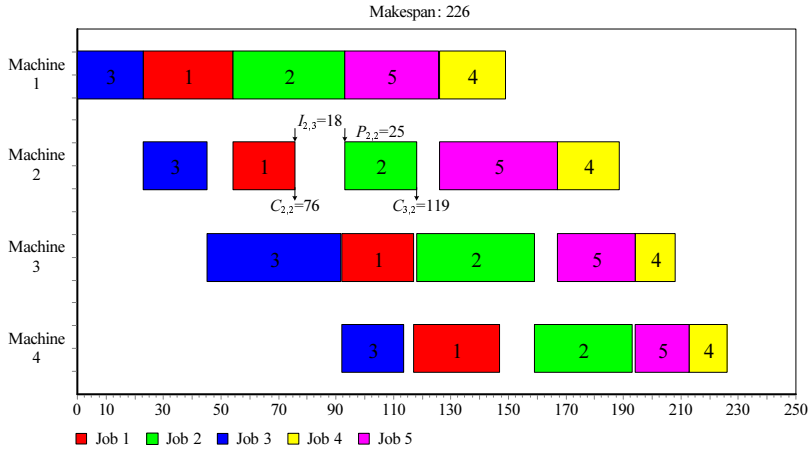

Fig. 1. Optimum solution for the PFSP example. Idle time allowed.

Although related, the no-idle PFSP and the regular PFSP are very different. As a matter of fact, the optimum solution for the example problem of Table 1 is $\pi^*_{no-idle} = \{2, 5, 1, 3, 4\}$, with a makespan value of 247 and is shown in Figure 2. We can see that π^*_{idle} and $\pi^*_{no-idle}$ are very different and only job

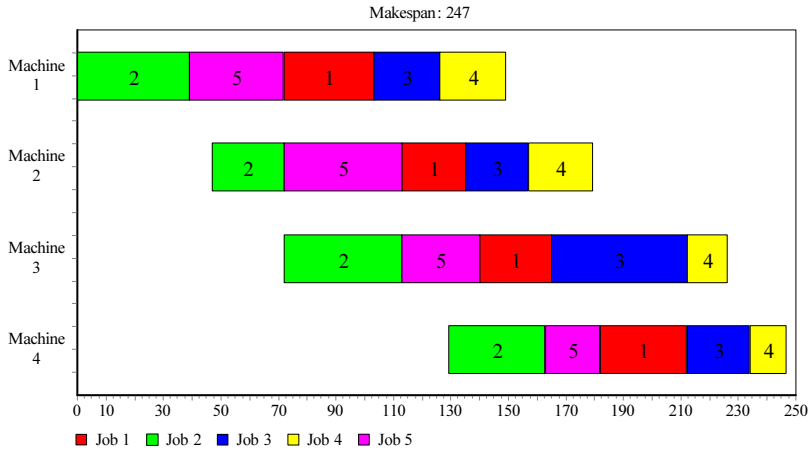

Fig. 2. Optimum solution for the PFSP example. No idle time allowed.

4 is located in the same position in both sequences. Similarly, the optimum makespan with the no-idle constraint is better than the makespan obtained by enforcing the no-idle constraint to π^*_{idle}.

Calculating the completion times $C_{j,i}$ in a no-idle flowshop is not trivial. Following the previous examples of Figures 1 and 2 we see that for each machine, jobs are delayed until we are sure that they can be processed without idle time. Therefore, we first need to calculate when a given machine can start processing with no needed idle time. We denote this as $S_i, i = \{1, \ldots, m\}$. Obviously, $S_1 = 0$. With this in mind, we calculate the S_i values as follows:

$$S_i = S_{i-1} + \max_{1 \leq h \leq n} \left\{ \sum_{j=1}^{h} p_{\pi_{(j)},i-1} - \sum_{j=1}^{h-1} p_{\pi_{(j)},i} \right\}, \quad i = \{2, \ldots, m\} \tag{10}$$

Once the S_i values are known, calculating the completion times is straightforward since the jobs are processed with no-idle time:

$$C_{\pi_{(1)},i} = S_i + p_{\pi_{(1)},i}, \quad i = \{1, \ldots, m\} \tag{11}$$

$$C_{\pi_{(j)},i} = C_{\pi_{(j-1)},i} + p_{\pi_{(j)},i}, \quad j = \{2, \ldots, m\}, i = \{1, \ldots, m\} \tag{12}$$

As a result, $C_{max} = C_{\pi_{(n)},m}$. However, it is interesting to mention that the completion times are not really needed for makespan criterion as $C_{max} = S_m + \sum_{j=1}^{n} p_{j,m}$. As we can see, calculating C_{max} for the no-idle PFSP has the same complexity as for the regular PFSP ($\mathcal{O}(nm)$) although more calculations are needed at each step. Note that in order to come up with a $\mathcal{O}(nm)$ complexity, the summations inside the max term in expression (10) have to be stored at each step. For example:

$$\sum_{j=1}^{h} p_{\pi_{(j)},i-1} = \sum_{k=1}^{h-1} p_{\pi_{(k)},i-1} + p_{\pi_{(h)},i-1}$$

Similarly to the PFSP, it is easy to come up with a MIP model to obtain the optimum solution for the no-idle PFSP. We propose an adaptation of the model presented in Saadani et al. (2005) here. The variable definition, objective function and constraint sets (3), (4) and (8), (9) are unchanged. Constraint sets (5)-(7) are changed by:

$$C_{1,1} = \sum_{j=1}^{n} X_{j,1} \cdot p_{j,1} \tag{13}$$

$$C_{k+1,i} = C_{k,i} + \sum_{j=1}^{n} X_{j,k+1} \cdot p_{j,i}, \quad k = \{1, \ldots, n-1\}, i = \{1, \ldots, m\} \tag{14}$$

$$C_{k,i+1} \geq C_{k,i} + \sum_{j=1}^{n} X_{j,k} \cdot p_{j,i+1}, \quad k = \{1, \ldots, n\}, i = \{1, \ldots, m-1\} \tag{15}$$

We see that the structure of these constraints has changed. The most important aspect is constraint set (14) where we enforce that the completion time

of a job in position $k+1$ is exactly equal to the completion time of job in position k plus the processing time of the job in position $k+1$. This ensures the no-idle constraint.

The computational complexity of the $F/prmu, no-idle/C_{max}$ problem is briefly commented in Tanaev et al. (1994) which in turn refers to an older communication in Russian. In any case, the $\mathcal{NP}$-Hardness of the $F3/prmu, no-idle/C_{max}$ was proved in Baptiste and Hguny (1997). Similarly, and according to Adiri and Pohoryles (1982), when Garey et al. (1976) proved the $\mathcal{NP}$-Completeness in the strong sense of the problem $F2/prmu/\sum C_j$ they did so with a no-idle instance, and therefore, the problem $F2/prmu, no-idle/\sum C_j$ is also $\mathcal{NP}$-Complete.

2 Literature review

To the best of our knowledge, Adiri and Pohoryles (1982) where the first to address the no-idle PFSP. They studied also the no-wait flowshop. The main contribution is a polynomial algorithm for solving the $F2/prmu, no-idle/\sum C_j$ problem to optimality. They also provided results for $m > 2$ but for special cases with dominating machines only.

Vachajitpan (1982) was the first to study the makespan objective. He proposed a MIP model with the additional characteristic that non-permutation sequences are allowed. Of course, the proposed model is shown to be impracticable even for small problem sizes. A Branch and Bound (B&B) method is also presented, but in this case for the permutation case. No computational results are provided beyond a small example.

Heuristics for the general m-machine no-idle PFSP were first examined by Woollam (1986) for the makespan objective. Basically, several heuristics were taken from the literature, including some of the aforementioned ones like the NEH. From the solution given in those heuristics (idle time allowed), a no-idle sequence was calculated, followed by a series on $n-1$, adjacent pairwise exchange moves. Computational results were carried out with five heuristics and instances of up to 25 jobs and 25 machines in size (25×25). Nowadays, such sizes are deemed as small. However, for such cases, NEH produced the best results.

Baptiste and Hguny (1997) proposed a B&B method for the general m-machine no-idle PFSP with makespan criterion. They also proved the $\mathcal{NP}$-Hardness of the problem.

Čepek et al. (2000) pointed out some errors found in the paper by Adiri and Pohoryles (1982). Furthermore, they demonstrated that in the case of total completion time criterion and two machines, it suffices to search permutation schedules only.

In a fairly unknown paper, Narain and Bagga (2003) study the $F3/prmu, no-idle/C_{max}$ problem. They provide a MIP model and a B&B algorithm along

with some rather limited computational results. The same problem with three machines is studied by Saadani et al. (2003). They proposed a lower bound and an effective heuristic. This heuristic compared favorably against an earlier method by the authors (Saadani et al., 2001). Notice that this work was later published in Saadani et al. (2005).
Kamburowski (2004) elaborates over Saadani et al. (2003) paper. The author proposes a network representation and identifies some paradoxes by which reducing some processing times might result in a prolongation of the makespan and viceversa.
Saadani et al. (2005) propose a Traveling Salesman Problem (TSP)-based heuristic for the $F/prmu, no-idle/C_{max}$. Basically, the authors modelize the distance between any two possible jobs as the resulting no-idle makespan value when sequencing these two jobs in all m machines. Starting from the minimum distance, the Nearest Insertion Rule (NRI) heuristic is applied by inserting, one by one, and in all positions, all pending jobs. The heuristic has a complexity of $\mathcal{O}(n^3)$ and is easily implementable. The authors tested the proposed heuristic against a MIP model and its optimum solution provided by LINGO in problems of sizes up to 17×30.
In two similar papers, Narain and Bagga (2005a,b), study the $F2/prmu, no-idle/\sum C_j$ and $F/prmu, no-idle/C_{max}$ problems, respectively. However, in the second case, only special variants with dominating machines are studied and heuristics are presented.
Kalczynski and Kamburowski (2005) proposed a heuristic for the $F/prmu, no-idle/C_{max}$ problem with a reported computational complexity of $\mathcal{O}(n^2m)$. The heuristic is compared against that of Saadani et al. (2005) with instances of size up to 100×40. Better results are reported on most instance sizes. The authors also present an adaptation of the NEH for the no-idle problem. Their proposed method is also shown to outperform this NEH heuristic.

As of late, no-idle flowshop has received renewed interest. Kalczynski and Kamburowski (2007) study special situations and problem combinations between the no-idle and no-wait flowshops.
Recently, Baraz and Mosheiov (2008) have proposed a simple two stage heuristic for the $F/prmu, no-idle/C_{max}$. In the first stage, pending jobs are added, one at a time, at the end of an incomplete sequence, and the job resulting in the least no-idle added makespan, is appended to the sequence. This phase carries out $\mathcal{O}(n^2)$ steps. In the second phase, all possible job interchanges are tested and the best moves are performed. There are $n(n-1)$ possible job pairs. Therefore, the authors conclude that the running time of their proposed heuristic is $\mathcal{O}(n^2)$. However, we want to point out a very important mistake here. The authors are not considering the added complexity of calculating the no-idle makespan at each step. Since this calculation has a computational complexity of $\mathcal{O}(nm)$, we conclude that the correct total computational complexity of their proposed heuristic is actually $\mathcal{O}(n^3m)$. In any case, the authors demonstrate the superiority of their proposed heuristic against that of Saadani

et al. (2005) but bypass other important papers like the one of Kalczynski and Kamburowski (2005). The size of the instances tested go all the way up to 400×8.
Also recently, in two similar papers, Pan and Wang (2008a,b) propose discrete differential evolution and a discrete particle swarm algorithms for the same problem. In both papers, an acceleration for the insertion neighborhood is proposed. This reduces the computational complexity of a single insertion neighborhood scan from $\mathcal{O}(n^3m)$ to $\mathcal{O}(n^2m)$ if the insertion is done in order. This acceleration is based on the very well known accelerations presented in Taillard (1990) for the same neighborhood but for the PFSP. Both algorithms use a form of advanced local search called Iterated Greedy (Ruiz and Stützle, 2007) that will be discussed later. The authors use the also well known benchmark of Taillard (1993) –extended to the no-idle flowshop– to test the results. In both papers, the authors test the proposed methods against the heuristics of Baraz and Mosheiov (2008) and Kalczynski and Kamburowski (2005). The results indicate that both the differential evolution and the particle swarm methods provide state-of-the-art results. However, these two methods are not compared between them.
As we can see, not many approaches have been suggested for the general m-machine $F/prmu, no-idle/C_{max}$ problem. However, it seems that this trend is reversing as several papers have appeared recently. It is one of the objectives of this paper to quantitatively compare these last approaches in order to identify the state-of-the-art.

There are other related papers that consider no-idle times, although not as a hard constraint or on related settings. For example, Liao (1993) relaxes the no-idle constraint and tries to minimize the number of idle intervals instead. This number is treated as a goal, that is subject to minimum makespan. The author presents a MIP model and a heuristic. A similar problem is studied in Saadani and Baptiste (2002) where the no-idle constraint is relaxed. In this case, a B&B algorithm is proposed for the three machine case where an optimal placement for one or more idle intervals is sought. A different paper is that by Giaro (2001) where "compact" open and flowshop problems are studied. Compact means that both no-idle and no-wait constraints exist. The author shows the incredible complexity of these problems where even proving the existence of a feasible compact schedule is already $\mathcal{NP}$-Hard.
Other shop settings are also studied in the literature. Narasimhan and Panwalkar (1984) and Narasimhan and Mangiameli (1987) study a two stage hybrid flowshop with no-idle parallel machines in the first stage. The symmetric problem is studied by Wang et al. (2005) where some heuristics are proposed for the case where the no-idle machines are on the second stage.
Niu and Gu (2006) study a no-idle PFSP with the additional consideration of fuzzy processing times. In this case, the mean makespan along with the makespan spread are studied with a mixture between particle swarm optimiza-

tion and genetic algorithms. Deteriorating jobs on no-idle dominant machines –a very special case– is studied in two related papers, Cheng et al. (2007a,b).

3 New approaches, discussion and adaptation of existing state-of-the-art methods

It is frequent in the scheduling literature to propose new algorithms for a given specific problem and to compare against existing approaches for that problem only. While this is reasonable, sometimes it is worthwhile to look for state-of-the-art methods in related problems. As we have seen in Sections 1.1 and 1.2, the regular and no-idle flowshop problems are different from a mathematical point of view. However, the search space is based on permutations and much of the existing knowledge –specially in the field of metaheuristics– might be applicable.
Therefore, in this Section we discuss improvements to some of the earlier reviewed methods that were specifically proposed for the no-idle flowshop. We also propose adaptations of high performing existing methods that where proposed for related problems like the regular PFSP.

Let us first analyze the recent proposal of Baraz and Mosheiov (2008). We refer to this heuristic as GH_BM. For the sake of complexity, we detail the heuristic here.

1. STEP 1 (greedy). Perform n iterations. At each iteration, append to the current sequence the unscheduled job yielding the least additional no-idle makespan.
2. STEP 2 (pairwise job interchange). From the sequence obtained at STEP 1, perform a single pass in the interchange neighborhood, testing all possible pairs of job exchanges. Accept those exchanges improving makespan.

At STEP 1, n jobs are tested in the first iteration, $n-1$ in the second and so on until the last job. Therefore, we have $n(n+1)/2$ steps. At each step, the makespan has to be calculated, with a cost of nm. Therefore the computational complexity, as discussed before, is $\mathcal{O}(n^3m)$. STEP 2 is essentially similar, as it is well known that the cardinality of the interchange neighborhood is also $n(n+1)/2$.
We want to focus our attention to this heuristic. The choice of the constructive heuristic in STEP 1 is probably not the best one. It has been long known that the NEH (Nawaz et al., 1983) heuristic is the best performer for flowshop problems in many different scenarios. See for example Framinan et al. (2003) or Ruiz and Maroto (2005) for recent results on this and for different objectives. Furthermore, NEH was shown very recently to be an excellent performer for the regular PFSP (see Rad et al., 2009). NEH considers lengthy jobs early in the sequence and then carries out insertions as shown in the following steps:

1. Sum the processing times of all jobs on all machines: $P_j = \sum_{i=1}^{m} p_{j,i}$.
2. Sort jobs in descending order of P_j.
3. Take job j, $j = 1, \ldots, n$ from the sorted list, insert it in all possible j positions of the partial incumbent sequence and place it in the position that results in the lowest C_{max}.

Even without the accelerations of Taillard (1990), the complexity of the NEH heuristic is $\mathcal{O}(n^3m)$, which is the same as STEP 1 in the GH_BM heuristic. Considering the good performance of the NEH, it seems reasonable to substitute STEP 1 by the NEH heuristic. Furthermore, using Taillard (1990) accelerations reduces the complexity of the NEH to $\mathcal{O}(n^2m)$. The extension of these accelerations to the no-idle flowshop have been proposed, as already mentioned, by Pan and Wang (2008a,b). Accelerations for the PFSP are extremely effective. As shown in Rad et al. (2009), a very efficient NEH implementation results in CPU times of only 77 milliseconds for instances as large as 500×20 in a modern desktop computer.
As regards STEP 2, it has been long known that for the PFSP, insertion neighborhoods give better results than adjacent interchange and general interchange neighborhoods. This was tested in many domains as early as in the work of Osman and Potts (1989). This is also true even for genetic mutation operators as an insert mutation performs much better than a swap or interchange mutation as shown in Reeves (1995) or more recently, in Ruiz et al. (2006). Furthermore, when scanning all the insertion neighbors of a single job, the same accelerations discussed before can be applied. This effectively brings down the application of a single pass of the insertion neighborhood to $\mathcal{O}(n^2m)$. As a result, a better alternative is to apply the insertion local search in STEP 2.

Considering the previous discussion, we propose an improvement of the GH_BM heuristic –which we call GH_BM2– that uses NEH with accelerations in STEP 1 and that uses a single pass insertion local search, also with accelerations, in STEP 2.

The second heuristic we want to draw our attention upon is the TSP-based SGM method by Saadani et al. (2005). This heuristic is composed of several steps:

1. Calculate the distance D_{jk} between any possible pair of jobs $j, k = \{1, \ldots, n\}, i \neq j$. D_{jk} is actually equivalent to S_m from expression (10) if just jobs j and k are scheduled in all m machines considering no-idle constraints.
2. Take the minimum D_{jk} and schedule jobs j and k in this order. Store the scheduled jobs in a partial sequence π_p
3. For every unscheduled job l, insert the job in every possible position of the incomplete sequence, this is, insert job l in the first position, in the second and so on until position $|\pi_p| + 1$. Among all pending jobs and all possible

positions, insert the job in the position that resulted in the minimum tour length increase.
4. Go back to step 3 until all jobs are scheduled.

At first, SGM heuristic might look expensive. However, it is actually very fast if implemented with care. As the heuristic is no more than an adaptation of the Nearest Insertion Rule (NRI) from the TSP to the no-idle PFSP, we do not have to calculate any makespan value. For example, the different "tour lengths" when inserting a given job 5 into all positions of $\pi_p = \{1, 2, 3\}$ are obtained by two simple summations and a single substraction. For this example, the current tour length is $L = D_{12} + D_{23} + D_{31}$. As a result, when inserting job 5 in the second position the new length is calculated as follows: $L' = L - D_{12} + D_{15} + D_{52}$. This does not reduce the theoretical complexity of the heuristic, as step 1 has a complexity of $\mathcal{O}(n^2)$ and step 3 has a complexity of $\mathcal{O}(n^3)$[1]. However, the amount of work per step is very low and the result is a very fast heuristic. For our tests, we have implemented such a fast version of the SGM algorithm.

Another heuristic considered in this chapter is the one proposed by Kalczynski and Kamburowski (2005). This heuristic is referred to as KK and consists of the following steps:

1. Calculate the sequence π_i by applying Johnson's (1954) algorithm on machines i and $i+1$, for $i = \{1, \ldots, m-1\}$. Set $\omega = \varnothing$, $K = |\omega| = 0$, and $\sigma_i = \pi_i$ for $i = \{1, \ldots, m-1\}$.
2. Assume that the current sequences are $\pi_i = (\sigma_i, \omega)$, where ω is the subsequence of scheduled jobs. For every unscheduled job s and position $k = \{1, \ldots, \lceil n/(n-K) \rceil\}$ in ω, compute $R(s,k) = \sum_{i=1}^{m-1} C_{max}((\sigma_i - \{s\}, \omega(s,k)); M_i, M_{i+1})$. The sequence $\sigma_i - \{s\}, \omega(s,k)$ is that obtained by deleting job s from σ_i and inserting it into the k-th position of ω. The objective is to find s^* and k^* that minimize $R(s,k)$. Set $\omega = \omega(s^*, k^*)$, $\sigma_i = \sigma_i - \{s^*\}$ for $i = \{1, \ldots, m-1\}$ and $K = K + 1$.
3. If $K < n$, return Step 2. Otherwise, $\pi_{kk} = \omega$ is the final sequence.

In this case, the method starts from m-1 sequences built with Johnson's algorithm. For each sequence, Step 2 chooses the best job to be deleted from the current Johnson's sequence and to be inserted in the subsequence of scheduled

[1] In computational complexity, constants multiplying large numbers are normally overlooked. However, in the case of scheduling, n and m are usually not measured in the thousands. Therefore, constants might be relevant. More specifically, in the first iteration of the third step of the SGM heuristic, $n-2$ pending jobs are inserted in 3 possible positions. In the second iteration, $n-3$ jobs are inserted in 4 positions and so on. This gives a total number of insertions of $\sum_{j=2}^{n-1}\{(n-j)\cdot(j+1)\} = \frac{1}{6}n^3 + \frac{1}{2}n^2 - \frac{8}{3}n + 2$. As we have seen, at each insertion, only three basic operations are carried out. The result is that although the heuristic is $\mathcal{O}(n^3)$, its performance is practice is very good.

jobs. The complexity of this method is $\mathcal{O}(n^3m)$. However, the authors state that the complexity of the heuristic is $\mathcal{O}(n^2m)$ since a speed up procedure based on the Critical Path Method (CPM) can be applied to compute the makespan value in $\mathcal{O}(1)$. Nevertheless, the cost to obtain the CPM is $\mathcal{O}(n)$ and it is necessary to compute it again when the sequence changes. On the other hand, the value of m is a very strong factor to compute the $R(s,k)$ values. Moreover, we have to consider that several searches and additional operations have to be carried out before the makespan value can be computed, since it is necessary to know the position of the deleted job in the Johnson's sequence. So, despite great efforts, and after coding all the speed-ups and formulae from Kalczynski and Kamburowski (2005), the complexity of the implemented KK heuristic remains at $\mathcal{O}(n^3m)$. We want to remark that in the original paper, the authors do not report CPU times. Additionally, we contacted Dr. Quan-Ke Pan for help as he did code the KK heuristic in the papers Pan and Wang (2008a,b). The code we received did not include the alleged accelerations either.

Recently, Rad et al. (2009) have proposed some algorithms based on the NEH procedure. From the proposed heuristics, here we adapt the three best performing ones to the no-idle PFSP. Namely, we consider the following heuristics:

- FRB3. It is an extension of the NEH method. After inserting a job in a given position, all jobs inserted in previous iterations are again reinserted in all possible positions. This reinsertion is motivated by the fact that after inserting a new job, existing jobs could be moved in order to better accommodate the new one. FRB3 was shown in Rad et al. (2009) to improve the performance of the NEH method almost a 300% on average. However, this comes at a cost, since the worst case computational complexity rises to $\mathcal{O}(n^3m)$.
- FRB4$_k$. It is a simplification of the FRB3 method. After inserting a job j, only the jobs at positions $\pm k$ from the position where job j has been finally inserted are reinserted. Since typically $k \ll n$, the computational complexity of FRB4$_k$ is $\mathcal{O}(n^2m)$. In any case, the empirical observed running time will be much higher that the accelerated NEH.
- FRB5. It is an extension of FRB3. Basically, after placing a given job j, a full local search in the insertion neighborhood until local optima is carried out. Furthermore, jobs are extracted at each step at random and without repetition to enforce an unbiased and powerful search. Given that the number of local search steps cannot be derived, the worst case complexity cannot be calculated. FRB5 was shown in Rad et al. (2009) to be much slower than NEH on average. However, it also produced the best results.

FRB3, FRB4$_k$ and FRB5 are adapted to the no-idle PFSP by simply calculating no-idle C_{max} values at each step. We employ the mentioned accelerations in the insertions. Furthermore, for FRB4$_k$, we test two k values, namely 4 and

12.

Lastly, we are interested in a recent state-of-the-art algorithm proposed for the regular PFSP-C_{max}. The Iterated Greedy method with local search (IG_{LS}) was shown in Ruiz and Stützle (2007) to produce better results than other much more complex state-of-the-art methods. Iterated Greedy (IG) has been successfully applied to other shop environments, like the SDST flowshop in Ruiz and Stützle (2008), no-wait flowshops (Pan et al., 2008), hybrid flowshops (Ying, 2008) and many others. Recently, Vallada and Ruiz (2008) have devised cooperative IG methods that have improved the results even further. There have been even approaches for multiobjective PFSP like the one shown in Framinan and Leisten (2008). It is clear that there is a strong recent trend in the application of IG methods to flowshop problems.
As the name implies, IG iterates over greedy constructive heuristics. IG_{LS} starts from the solution given by the NEH. Then a local search step is carried out (this local search will be explained later). Then, three phases are iteratively applied until a termination criterion is met. First, we have the destruction. During this phase, some jobs are extracted from the incumbent sequence, at random. The second phase is construction, were the removed jobs are inserted, one by one, in all positions of the partially destructed sequence. Each job is placed in the position resulting in the lowest C_{max} increase. The reconstruction phase is applied until a new complete sequence is obtained. Lastly, this new sequence undergoes a local search step. After the application of these three steps, the new solution is considered for replacing the incumbent one. More details can be seen in Ruiz and Stützle (2007). IG_{LS} uses the principle and accelerations of the NEH. Of special mention is the local search step, which is detailed in Figure 3. As we can see, it is not a straightforward

```
procedure LocalSearch_Insertion(π)
    improve := true;
    while (improve = true) do
        improve := false;
        for i := 1 to n do
            remove a job k at random from π without repetition
            π' := best permutation after inserting k in all positions of π;
            if C_max(π') < C_max(π) then
                π := π';
                improve := true;
            endif
        endfor
    endwhile
    return π
end
```

Fig. 3. Local search employed in IG_{LS} (Ruiz and Stützle, 2007).

local search. First of all, in the inner loop, all jobs are extracted one by one, but instead of doing this in order, it is done at random, to avoid a biased result. Each job is inserted in all possible positions (using accelerations) and if a better C_{max} value is found, the solution is replaced. We continue until all jobs have been reinserted and if an improvement is found, the search starts again for all jobs. The local search is finished when no improvements are found after reinserting all jobs. Note that this is also the local search employed in the FRB5 heuristic. A very similar local search is applied in the second step of the GH_BM2 method. However, we apply a single pass, i.e., the while loop is eliminated.
As mentioned in Section 2, Pan and Wang (2008a,b) employ IG_{LS} as a local search method inside their proposed approaches. More specifically, they carry out a few iterations of IG_{LS} to good solutions found during the search. In this chapter we propose the application of the pure IG_{LS} method and not as a surrogate local search step.

4 Computational evaluation

In this Section we aim at comparing all existing heuristics that we have reviewed with detail in the previous Section. We will comment first on the benchmark employed.

The vast majority of the flowshop literature concentrates on the well known benchmark of Taillard (1993). This benchmark is composed of 12 groups of 10 instances each, totalling 120 instances. Each group is characterized by a combination of n and m values ($n \times m$). The groups are $\{20, 50, 100\} \times \{5, 10, 20\}$, $200 \times \{10, 20\}$ and 500×20. However, this benchmark was proposed as a difficult set of instances for the PFSP-C_{max} only. Despite of this, it is common in the literature to "adapt" this benchmark to other objectives and to other problem variants. We have, however, several concerns with this approach. First of all, Taillard's benchmark is not complete, in the sense that some combinations of n and m are missing. For example, there are no instances in the sets 200×5, and $500 \times \{5, 10\}$. Apart from not being complete, the different values of n and m are not equidistant. These two facts make statistical testing complicated as one cannot easily analyze the factor effect of n and m. Second, a benchmark of only 120 instances is not enough if small differences on performance are to be detected with some statistical significance. Third, there is no guarantee that a set of hard instances for the PFSP-C_{max} will be also hard for other problem variants. Last but not least, Taillard's benchmark is already aging and most instances have been already solved to optimality (at least for the problem for which they were originally devised). As a result of all of the above discussion, we propose an extended benchmark of instances specifically designed for the no-idle PFSP.
In the proposed benchmark we have 250 instances where we have all com-

binations of $n = \{50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$ and $m = \{10, 20, 30, 40, 50\}$. There are five replicates per combination. The processing times are uniformly distributed in the range $[1, 99]$ as usual in the literature. As we can see, there are more values of n and m and all of them are equidistant, with all combinations present. Such a larger benchmark is easier on statistical testing. The proposed benchmark, along with the best known solutions is available for download at `http://soa.iti.es`. Notice that we even have a second smaller benchmark for calibration and testing so that calibration and final results are not carried out over the same set of instances.

In the evaluation, we will test the following heuristic methods. All of them are deterministic:

1. NEH from Nawaz et al. (1983) with the accelerations published in Pan and Wang (2008a,b). Computational complexity $\mathcal{O}(n^2m)$.
2. Original NEH with no accelerations, referred to as NEH_{na}. Computational complexity $\mathcal{O}(n^3m)$
3. SGM from Saadani et al. (2005). Computational complexity $\mathcal{O}(n^3)$. Highly efficient version.
4. KK from Kalczynski and Kamburowski (2005) Computational complexity $\mathcal{O}(n^3m)$.
5. GH_BM from Baraz and Mosheiov (2008). Computational complexity $\mathcal{O}(n^3m)$.
6. New proposed algorithm GH_BM2 with accelerations. Based on the two phases of GH_BM. Computational complexity $\mathcal{O}(n^2m)$.
7. GH_BM2 without accelerations, referred to as GH_BM2_{na}. Computational complexity $\mathcal{O}(n^3m)$.
8. FRB3 from Rad et al. (2009). Computational complexity $\mathcal{O}(n^3m)$.
9. FRB4_k from Rad et al. (2009) with k values of 4 and 12. (FRB4_4 and FRB4_{12}). Computational complexity $\mathcal{O}(kn^2m)$ or $\mathcal{O}(n^2m)$.

As we can see, we also wanted to test NEH and GH_BM2 without accelerations (NEH_{na} and GH_BM2_{na}). This way we can assess the impact of accelerations in the CPU times.

We will also test the following metaheuristics. These are stochastic and do not provide the same result after each run. Most of them also have a stopping criterion that will be discussed later.

1. HDPSO from Pan and Wang (2008a).
2. DDELS from Pan and Wang (2008b). We will simply refer to this method as "DDE".
3. FRB5 from Rad et al. (2009).
4. IG_{LS} from Ruiz and Stützle (2007).

It has to be reminded that the local search step in HDPSO and DDE is actually a few applications of the IG_{LS} method. In turn, IG_{LS} and FRB5 share

the same local search step which was detailed in Figure 3 before.

All methods have been coded in Delphi 2007. All algorithms share most code and especially the critical functions that evaluate the no idle C_{max} as well as accelerations. Therefore, results are completely and fully comparable. For the tests we have used a cluster of 12 PC/AT computers with Intel Core 2 Duo E6600 processors running at 2.4 GHz and with 1 GB of RAM. There is no multi-core or multi-threading programming so a single core on each computer is actually used.
The performance measure that we will be using is the Relative Percentage Deviation (RPD) over the best known solution for each instance:

$$\text{Relative Percentage Deviation } (RPD) = \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \times 100 \qquad (16)$$

where Heu_{sol} is the solution given by any of the tested heuristics for a given instance and $Best_{sol}$ is the best known solution for each instance. These best known solutions are available from `http://soa.iti.es`.
It has to be noted that all metaheuristic methods (HDPSO, DDE, FRB5 and IG_{LS}) are stochastic and therefore five different runs are carried out. Furthermore, these methods –with the exception of FRB5– have a natural stopping criterion. Following previous works like Ruiz and Maroto (2005), Ruiz et al. (2006), Ruiz and Stützle (2007), Vallada et al. (2008) and others, we set a stopping time based on elapsed CPU time (not wall time). This elapsed CPU time is accurately measured inside each method in order to stop it whenever the maximum allowed CPU time has passed. Moreover, this maximum elapsed CPU time is set with the following formula: $n \cdot (m/2) \cdot t$ milliseconds. Setting the time limit in this way allows more computational effort as the number of jobs and/or the number of machines increases. This helps in lessening the effect of the instance size on the results and on the statistical analysis. Lastly, in order to test the effect of CPU time, these three methods (HDPSO, DDE and IG_{LS}) are tested with three different elapsed CPU time termination criteria, where t=10, 20 and 30. This means that for the largest instances of 500×50 and the highest value of $t = 30$, a maximum elapsed time of $500 \cdot (50/2) \cdot 30$ =375,000 milliseconds or 6.25 minutes are allowed. Results are separated in values of t. For example, HDPSO^{10} refers to the same method where t has been set to 10.

All in all, we have 10 heuristics that are run a single time and four metaheuristics that are run five times, three of them are run for three different stopping criteria. As a result we have a total of 15,000 data points. As we will see, with such a large dataset and comprehensive computational campaign, we are able to draw strong and statistically sound conclusions.

4.1 Heuristic results

We first comment on the results of the 10 tested heuristics. The Average Relative Percentage Deviations ($\overline{RPD}$), grouped by n and m values, are given in Table 2. The elapsed CPU times (in seconds) needed by each method are given in Table 3. Note that first we comment on averages but afterwards we will provide statistical analyses.

As expected, NEH and NEH$_{na}$ give the same exact results. The same applies to GH_BM2 and GH_BM2$_{na}$. The only difference is the CPU time employed. We can see that NEH$_{na}$ is about 76 times slower, on average, than NEH. Similarly, GH_BM2$_{na}$ is about 104 times slower than GH_BM2. Clearly, the accelerations proposed by Pan and Wang (2008a,b) should be applied at all costs. Additionally, our initial hypothesis from Section 1.2 that calculating the C_{max} value for the no-idle PFSP is costlier than for the regular PFSP is confirmed. Observing the results from Rad et al. (2009), we see that for the largest instances tested there of size 500×20, the regular PFSP NEH needed 0.0773 seconds to give a solution. For the instances of the same size in this chapter, the NEH tested here for the no-idle PFSP needs 0.134 seconds, which is almost two times more costly. In any case, NEH is the fastest heuristic tested here. If coded with accelerations, it needs about 77 milliseconds, on average, to obtain a solution.

The next fastest method is SGM. As we hypothesized, its complexity of $\mathcal{O}(n^3)$ is compensated with the little work that is needed at each iteration. As a matter of fact, SGM is many times faster, on average, than GH_BM2 or both FRB4 methods, that have a complexity of $\mathcal{O}(n^2m)$. This also confirms the hypothesis that for scheduling problems, high constants in computational complexity calculations should not be overlooked.

As far as the $\overline{RPD}$ goes, we have that SGM gives rather poor results when compared to the other methods. Clearly, SGM is not recommended even if CPU time is considered as it is both slower and worse performing than NEH. The fact that SGM is not a good performer was already observed by Kalczynski and Kamburowski (2005), Baraz and Mosheiov (2008) and Pan and Wang (2008a,b). Although we have carried out a very effective coding, it does not suffice.

KK gives good results, better than NEH, which confirms the original findings of Kalczynski and Kamburowski (2005). However, our results in our implementation of KK are much better (when compared against NEH) than those reported by Pan and Wang (2008a,b). In these two papers, KK is tested against NEH and is shown only marginally better. Our results show that KK improves NEH on almost all instances. This is a very interesting result since as has been mentioned, NEH is unbeatable for the regular PFSP-C_{max}. It seems that this is not true for the no-idle PFSP. In any case, these results have to be considered only when CPU time is also accounted for. As mentioned, we have been unable to reproduce, despite our best efforts, the speed-ups reported in Kalczynski and Kamburowski (2005). As a matter of fact, our implementa-

Table 2. Average Relative Percentage Deviation ($\overline{RPD}$) over the best solution known obtained by the tested heuristics

n	m	NEH	NEH_{na}	SGM	KK	GH_BM	GH_BM2	GH_BM2_{na}	FRB3	$FRB4_4$	$FRB4_{12}$
50	10	8.24	8.24	19.62	3.80	5.15	3.04	3.04	2.51	3.43	3.29
	20	10.78	10.78	24.03	6.90	8.94	5.18	5.18	4.64	5.74	4.74
	30	12.15	12.15	27.20	7.72	8.63	7.04	7.04	4.65	6.49	5.58
	40	12.00	12.00	27.09	7.51	9.40	6.57	6.57	4.21	6.54	5.15
	50	11.75	11.75	29.85	9.39	11.25	7.73	7.73	5.21	7.18	6.89
100	10	5.54	5.54	14.03	1.61	4.21	2.04	2.04	1.55	2.32	1.53
	20	7.95	7.95	23.85	2.29	5.55	3.34	3.34	2.06	4.09	3.76
	30	10.32	10.32	29.83	5.78	6.54	6.11	6.11	3.63	6.25	4.52
	40	11.40	11.40	34.66	6.14	11.30	6.93	6.93	5.28	8.01	6.74
	50	11.60	11.60	30.68	6.64	9.57	7.43	7.43	4.59	7.87	6.16
150	10	2.54	2.54	11.19	0.69	1.26	0.60	0.60	0.06	0.59	0.42
	20	6.54	6.54	21.07	2.52	4.73	2.71	2.71	2.18	3.12	2.96
	30	7.60	7.60	26.08	2.66	5.13	3.33	3.33	1.95	4.54	3.28
	40	11.13	11.13	32.25	4.87	9.21	6.15	6.15	3.93	6.03	5.43
	50	10.35	10.35	31.87	6.15	8.92	5.89	5.89	4.01	7.07	4.52
200	10	2.38	2.38	9.87	0.55	1.09	0.42	0.42	0.34	0.40	0.33
	20	4.08	4.08	18.53	1.43	3.26	1.70	1.70	1.02	2.24	2.03
	30	6.62	6.62	25.60	1.73	4.77	2.80	2.80	1.90	3.66	2.65
	40	9.24	9.24	30.57	3.31	7.53	4.10	4.10	2.38	5.18	3.96
	50	8.70	8.70	33.72	4.24	7.10	4.97	4.97	2.94	5.82	4.57
250	10	1.42	1.42	8.00	0.52	0.35	0.18	0.18	0.22	0.21	0.21
	20	4.53	4.53	19.83	1.35	3.09	1.59	1.59	0.74	2.20	1.34
	30	6.02	6.02	26.72	1.49	4.51	2.20	2.20	1.27	2.83	2.18
	40	8.13	8.13	30.34	1.89	5.95	3.80	3.80	1.51	4.70	2.99
	50	9.46	9.46	34.40	2.81	7.17	4.81	4.81	3.13	5.75	4.79
300	10	1.57	1.57	8.00	0.23	0.64	0.18	0.18	0.08	0.19	0.15
	20	4.08	4.08	18.85	1.06	2.43	1.75	1.75	0.53	1.44	1.22
	30	5.77	5.77	24.73	1.16	3.40	1.59	1.59	1.64	2.61	2.36
	40	6.48	6.48	27.91	1.43	5.03	2.70	2.70	1.71	3.39	2.72
	50	8.49	8.49	32.28	2.46	6.91	3.93	3.93	2.15	4.90	4.23
350	10	1.18	1.18	7.69	0.24	0.58	0.16	0.16	0.17	0.29	0.16
	20	3.16	3.16	15.83	0.79	2.33	0.89	0.89	0.43	1.05	0.90
	30	4.79	4.79	23.43	1.16	3.57	2.07	2.07	0.95	2.30	2.01
	40	5.60	5.60	26.39	1.24	4.29	2.63	2.63	1.48	3.20	2.89
	50	7.31	7.31	29.88	1.27	5.35	3.43	3.43	1.38	4.09	3.16
400	10	1.05	1.05	7.84	0.33	0.49	0.20	0.20	0.04	0.11	0.10
	20	3.12	3.12	16.07	0.76	2.16	1.02	1.02	0.62	1.29	1.00
	30	4.26	4.26	21.36	0.87	3.08	1.63	1.63	1.04	1.95	1.56
	40	5.32	5.32	24.61	1.36	3.19	1.67	1.67	0.57	2.26	1.72
	50	6.66	6.66	28.99	1.44	5.89	2.90	2.90	1.55	3.72	3.19
450	10	1.04	1.04	8.72	0.28	0.49	0.18	0.18	0.12	0.26	0.19
	20	3.01	3.01	17.18	0.75	1.95	0.96	0.96	0.44	1.08	0.71
	30	4.29	4.29	21.41	0.61	2.90	1.36	1.36	0.89	2.18	1.81
	40	4.60	4.60	25.82	0.98	3.42	1.85	1.85	0.78	2.28	1.86
	50	6.67	6.67	28.37	1.24	5.15	2.62	2.62	1.74	3.05	2.39
500	10	1.04	1.04	7.13	0.34	0.50	0.16	0.16	0.03	0.12	0.18
	20	1.89	1.89	13.22	0.47	0.96	0.47	0.47	0.23	0.51	0.44
	30	3.25	3.25	20.70	0.67	1.95	1.16	1.16	0.60	1.30	1.00
	40	4.90	4.90	23.96	1.04	3.54	2.22	2.22	0.99	2.83	2.05
	50	6.11	6.11	29.19	1.25	4.51	2.61	2.61	1.20	3.21	2.30
Average		6.12	6.12	22.61	2.35	4.59	2.82	2.82	1.75	3.24	2.61

Table 3. Elapsed CPU times needed by the tested heuristics (in seconds).

n	m	NEH	NEH$_{na}$	SGM	KK	GH_BM	GH_BM2	GH_BM2$_{na}$	FRB3	FRB4$_4$	FRB4$_{12}$
50	10	0.001	0.009	0.001	0.047	0.041	0.003	0.028	0.022	0.001	0.013
	20	0.003	0.016	0.001	0.088	0.059	0.006	0.056	0.044	0.013	0.022
	30	0.001	0.019	0.003	0.131	0.088	0.009	0.084	0.066	0.016	0.034
	40	0.003	0.031	0.003	0.184	0.106	0.009	0.109	0.088	0.016	0.047
	50	0.006	0.034	0.001	0.228	0.128	0.009	0.134	0.109	0.028	0.053
100	10	0.001	0.050	0.006	0.325	0.228	0.016	0.197	0.172	0.028	0.069
	20	0.003	0.103	0.003	0.681	0.406	0.016	0.403	0.344	0.044	0.100
	30	0.016	0.153	0.006	1.034	0.569	0.022	0.597	0.509	0.069	0.138
	40	0.016	0.203	0.009	1.391	0.756	0.034	0.791	0.684	0.088	0.191
	50	0.016	0.253	0.006	1.753	0.938	0.044	1.000	0.863	0.109	0.250
150	10	0.003	0.163	0.009	1.075	0.684	0.022	0.647	0.563	0.050	0.113
	20	0.016	0.331	0.016	2.241	1.275	0.038	1.303	1.147	0.106	0.231
	30	0.019	0.494	0.016	3.425	1.856	0.056	1.981	1.713	0.156	0.356
	40	0.028	0.666	0.016	4.609	2.475	0.072	2.638	2.284	0.197	0.444
	50	0.031	0.838	0.022	5.788	3.075	0.094	3.313	2.869	0.259	0.578
200	10	0.009	0.375	0.025	2.516	1.572	0.031	1.500	1.334	0.094	0.203
	20	0.031	0.781	0.025	5.297	2.950	0.066	3.100	2.691	0.181	0.416
	30	0.031	1.172	0.031	8.116	4.353	0.103	4.656	4.047	0.281	0.641
	40	0.044	1.563	0.038	10.906	5.769	0.125	6.284	5.403	0.366	0.819
	50	0.047	1.991	0.038	13.691	7.147	0.163	7.881	6.756	0.469	1.075
250	10	0.022	0.747	0.041	4.953	2.994	0.063	3.009	2.597	0.153	0.334
	20	0.031	1.519	0.047	10.441	5.738	0.109	6.159	5.256	0.291	0.650
	30	0.050	2.294	0.053	15.950	8.428	0.150	9.228	7.928	0.428	0.969
	40	0.075	3.100	0.063	21.419	11.238	0.200	12.475	10.559	0.575	1.338
	50	0.081	3.853	0.063	26.847	13.922	0.244	15.525	13.272	0.722	1.622
300	10	0.028	1.309	0.066	8.691	5.219	0.078	5.288	4.528	0.216	0.484
	20	0.050	2.638	0.078	18.269	9.959	0.141	10.747	9.172	0.413	0.959
	30	0.069	4.000	0.084	27.863	14.769	0.216	16.300	13.728	0.628	1.428
	40	0.094	5.431	0.097	37.572	19.650	0.281	22.019	18.256	0.850	1.950
	50	0.122	6.719	0.106	47.197	24.375	0.353	27.275	22.803	1.047	2.450
350	10	0.034	2.091	0.103	14.166	8.341	0.100	8.578	7.222	0.284	0.634
	20	0.063	4.288	0.116	29.753	16.109	0.188	17.431	14.506	0.584	1.394
	30	0.103	6.481	0.131	45.559	23.872	0.291	26.328	21.731	0.853	1.978
	40	0.128	8.675	0.150	61.197	31.822	0.375	35.369	28.931	1.153	2.719
	50	0.166	10.884	0.159	76.753	39.809	0.494	44.147	36.300	1.431	3.391
400	10	0.050	3.138	0.156	21.666	12.497	0.134	12.916	10.666	0.375	0.850
	20	0.084	6.447	0.175	45.522	24.291	0.266	26.316	21.594	0.763	1.784
	30	0.125	9.697	0.188	69.759	36.263	0.375	39.753	32.359	1.119	2.625
	40	0.166	13.122	0.209	93.481	48.697	0.500	53.541	43.313	1.491	3.513
	50	0.213	16.494	0.225	117.806	60.347	0.628	67.084	54.031	1.906	4.438
450	10	0.053	4.513	0.228	31.372	17.734	0.163	18.516	15.213	0.481	1.109
	20	0.109	9.272	0.244	66.203	34.856	0.319	37.803	30.672	0.959	2.256
	30	0.163	14.034	0.266	101.606	52.313	0.475	57.481	46.175	1.450	3.434
	40	0.216	18.959	0.284	136.275	69.600	0.628	77.481	61.731	1.931	4.494
	50	0.269	23.953	0.309	172.106	87.506	0.788	97.922	78.138	2.403	5.853
500	10	0.069	6.228	0.303	43.463	24.322	0.206	25.525	20.881	0.591	1.366
	20	0.134	12.859	0.334	91.925	47.778	0.394	52.341	42.216	1.213	2.906
	30	0.194	19.509	0.359	141.178	73.450	0.584	79.722	63.297	1.772	4.250
	40	0.263	26.309	0.391	189.394	96.788	0.788	107.088	84.556	2.384	5.759
	50	0.325	33.894	0.422	240.416	120.406	0.972	137.872	106.172	2.972	7.088
Average		0.077	5.834	0.114	41.447	21.551	0.229	23.759	19.190	0.680	1.596

tion of the KK heuristic results in a very slow method –the slowest among all tested heuristics–. Actually, and as we will see in next section, the average CPU time required by our KK implementation is very similar to that used by HDPSO^{10}, DDE^{10} or IG$^{10}_{LS}$ while the results are much worse. We do not claim here that KK cannot be implemented more efficiently, but after so much effort, it is clear that something is amiss in Kalczynski and Kamburowski (2005) paper and that additional information might be needed in order to code the speed-ups.
Another interesting result comes after comparing GH_BM and GH_BM2. As we stated initially, GH_BM2 is much faster, as using the insertion neighborhood allows important speed-ups. Actually, GH_BM is about 94 times slower than GH_BM2. Notice that Baraz and Mosheiov (2008) reported CPU times of 2.94 seconds for instances of size 200×8 on a Pentium IV 2.8 GHz. Our closer results are of 1.572 seconds on average for instances of size 200×10. Our Core 2 Duo processor running at 2.4 GHz is actually faster than a Pentium IV 2.8 GHz (even if running at a lower frequency clock). Therefore, we can safely state that we have a good implementation of GH_BM. If we compare the $\overline{RPD}$ values, GH_BM2 is about 63% better. Consequently, we can easily conclude that GH_BM2 is preferable to GH_BM. Of course, it could be argued that GH_BM could have been also accelerated. However, this is only true for step 1 as no accelerations are known for reducing the complexity of scanning the interexchange neighborhood in the PFSP. In any case, GH_BM$_{na}$ is only a bit slower than GH_BM and also a 63% better.
The last three methods in the comparison offer very good results. FRB3, for example, gives the lowest $\overline{RPD}$ among all tested heuristics. The CPU times needed, however, are the third highest after KK and GH_BM. FRB4$_4$ is dominated, both from a CPU time and $\overline{RPD}$ by GH_BM2. FRB4$_{12}$ gives better $\overline{RPD}$ than GH_BM2 but at a significantly larger CPU time.

From the heuristics tested, we can conclude that FRB3 and GH_BM2 are the best performers, the first one as regards $\overline{RPD}$ and the second one as the best compromise between quality of results and CPU time. While KK gives results that are a bit better than GH_BM2, improving its speed to match that of GH_BM2 is certainly a challenge.

Of course, comparing average results could be misleading. We need to carefully test the statistical significance of these observed average differences. This will be done in later sections.

4.2 Metaheuristic results

We now provide the results of the four tested metaheuristics. Recall that for three of them we have tested three different stopping criteria. The ($\overline{RPD}$) values and CPU times, also grouped by n and m values, are given in Tables 4 and 5.

Table 4. Average Relative Percentage Deviation ($\overline{RPD}$) over the best solution known obtained by the tested metaheuristics

n	m	HDPSO^{10}	HDPSO^{20}	HDPSO^{30}	DDE^{10}	DDE^{20}	DDE^{30}	FRB5	IG$^{10}_{LS}$	IG$^{20}_{LS}$	IG$^{30}_{LS}$
50	10	0.97	0.79	0.58	3.86	4.17	4.17	2.64	0.54	0.41	0.25
	20	0.99	0.61	0.52	4.77	4.88	4.89	3.11	0.59	0.39	0.33
	30	1.15	1.11	1.19	5.60	5.67	5.56	4.15	0.97	0.61	0.64
	40	1.20	1.12	1.16	5.81	5.70	5.11	3.51	1.09	0.96	0.78
	50	2.32	1.62	1.47	6.33	6.36	6.22	5.51	1.92	1.42	1.52
100	10	0.26	0.21	0.25	2.34	2.43	2.60	0.90	0.23	0.13	0.17
	20	0.74	0.62	0.58	2.95	2.85	3.04	1.72	0.57	0.44	0.33
	30	1.22	0.90	0.83	4.82	4.81	4.55	2.92	0.87	0.54	0.46
	40	1.65	1.15	1.23	6.49	6.40	6.30	4.56	1.49	0.87	0.87
	50	1.85	1.30	0.93	5.95	5.80	5.85	4.47	1.47	1.09	0.73
150	10	0.10	0.03	0.02	0.83	0.73	0.89	0.08	0.03	0.01	0.01
	20	0.84	0.61	0.54	2.94	2.73	2.61	1.39	0.59	0.45	0.34
	30	0.82	0.75	0.70	3.05	3.24	3.28	1.58	0.78	0.51	0.42
	40	1.95	1.16	1.27	6.25	6.38	6.19	3.02	1.52	0.91	0.73
	50	1.72	1.19	0.81	4.58	4.70	4.51	2.62	1.51	0.96	0.68
200	10	0.15	0.10	0.11	0.60	0.70	0.61	0.24	0.14	0.13	0.06
	20	0.42	0.25	0.25	1.50	1.52	1.62	0.60	0.36	0.22	0.12
	30	0.63	0.56	0.37	3.08	2.83	2.86	1.26	0.50	0.33	0.21
	40	1.24	0.91	0.49	4.07	4.24	4.03	2.05	0.83	0.49	0.44
	50	1.55	0.89	0.75	4.56	4.17	4.18	2.68	1.11	0.63	0.42
250	10	0.05	0.02	0.03	0.37	0.33	0.37	0.13	0.02	0.01	0.01
	20	0.37	0.28	0.23	1.55	1.80	1.69	0.52	0.22	0.21	0.17
	30	0.77	0.56	0.45	2.29	2.25	2.52	0.83	0.52	0.41	0.31
	40	1.02	1.01	0.66	3.60	3.62	3.65	1.45	1.04	0.64	0.54
	50	1.70	0.90	0.68	4.14	4.39	4.45	2.66	1.54	0.91	0.56
300	10	0.05	0.06	0.02	0.32	0.35	0.37	0.09	0.04	0.02	0.01
	20	0.36	0.33	0.30	1.46	1.68	1.75	0.54	0.31	0.28	0.23
	30	0.45	0.41	0.29	2.03	2.02	2.05	0.67	0.47	0.28	0.23
	40	0.75	0.57	0.43	2.93	2.81	2.73	0.87	0.76	0.45	0.26
	50	1.05	0.87	0.67	3.69	3.61	3.60	1.63	1.12	0.60	0.42
350	10	0.09	0.04	0.05	0.30	0.38	0.37	0.14	0.05	0.04	0.03
	20	0.40	0.28	0.28	1.28	1.23	1.33	0.33	0.32	0.21	0.23
	30	0.68	0.53	0.42	2.13	1.83	2.02	0.71	0.59	0.44	0.33
	40	1.01	0.71	0.49	2.33	2.44	2.42	0.97	0.85	0.48	0.39
	50	1.20	0.65	0.57	2.90	2.96	2.79	1.11	1.05	0.68	0.40
400	10	0.05	0.04	0.02	0.22	0.19	0.21	0.08	0.04	0.03	0.01
	20	0.26	0.21	0.23	1.06	1.09	1.06	0.53	0.25	0.17	0.14
	30	0.57	0.51	0.47	1.76	1.79	1.73	0.57	0.60	0.46	0.25
	40	0.57	0.39	0.35	1.87	1.84	2.08	0.51	0.41	0.29	0.33
	50	0.99	0.62	0.66	2.50	2.49	2.38	0.86	0.96	0.57	0.37
450	10	0.07	0.05	0.05	0.32	0.31	0.33	0.05	0.06	0.03	0.02
	20	0.25	0.18	0.19	1.00	1.00	1.13	0.28	0.22	0.18	0.12
	30	0.41	0.29	0.21	1.52	1.72	1.67	0.47	0.37	0.24	0.20
	40	0.58	0.48	0.44	1.66	1.77	1.59	0.48	0.58	0.44	0.36
	50	0.95	0.74	0.57	2.51	2.39	2.59	0.72	0.76	0.68	0.58
500	10	0.07	0.04	0.04	0.25	0.26	0.25	0.08	0.06	0.04	0.03
	20	0.20	0.15	0.13	0.60	0.65	0.72	0.15	0.20	0.15	0.10
	30	0.34	0.30	0.30	1.07	1.08	1.11	0.34	0.28	0.31	0.20
	40	0.82	0.62	0.46	2.08	2.00	2.11	0.65	0.69	0.45	0.35
	50	1.02	0.66	0.46	2.30	2.38	2.31	0.57	0.80	0.55	0.45
Average		0.78	0.57	0.48	2.65	2.66	2.65	1.36	0.65	0.43	0.34

Table 5. Elapsed CPU times needed by the tested metaheuristics (in seconds).

n	m	HDPSO^{10}	HDPSO^{20}	HDPSO^{30}	DDE^{10}	DDE^{20}	DDE^{30}	FRB5	IG$^{10}_{LS}$	IG$^{20}_{LS}$	IG$^{30}_{LS}$
50	10	2.52	5.02	7.51	2.50	5.00	7.50	0.10	2.50	5.00	7.50
	20	5.02	10.01	15.02	5.00	10.00	15.00	0.16	5.00	10.00	15.00
	30	7.51	15.02	22.51	7.50	15.00	22.50	0.22	7.50	15.00	22.50
	40	10.01	20.01	30.01	10.00	20.00	30.00	0.29	10.00	20.00	30.00
	50	12.50	25.01	37.51	12.50	25.00	37.50	0.34	12.50	25.00	37.50
100	10	5.03	10.03	15.02	5.00	10.00	15.00	0.54	5.00	10.00	15.00
	20	10.02	20.02	30.02	10.00	20.00	30.00	1.08	10.00	20.00	30.00
	30	15.02	30.02	45.01	15.00	30.00	45.00	1.74	15.00	30.00	45.00
	40	20.02	40.01	60.02	20.00	40.00	60.00	2.32	20.00	40.00	60.00
	50	25.01	50.01	75.01	25.00	50.00	75.00	2.93	25.00	50.00	75.00
150	10	7.56	15.03	22.54	7.50	15.00	22.50	1.36	7.50	15.00	22.50
	20	15.02	30.03	45.01	15.00	30.00	45.00	3.36	15.00	30.00	45.00
	30	22.52	45.02	67.54	22.50	45.00	67.50	5.49	22.50	45.00	67.50
	40	30.01	60.02	90.01	30.00	60.00	90.00	7.91	30.00	60.00	90.00
	50	37.51	75.03	112.50	37.50	75.00	112.50	10.34	37.50	75.00	112.50
200	10	10.06	20.06	30.07	10.00	20.00	30.00	3.09	10.00	20.00	30.00
	20	20.05	40.03	60.04	20.00	40.00	60.00	7.29	20.00	40.00	60.00
	30	30.02	60.04	90.03	30.00	60.00	90.00	12.71	30.00	60.00	90.00
	40	40.02	80.01	120.03	40.00	80.00	120.00	18.30	40.00	80.00	120.00
	50	50.01	100.02	150.03	50.00	100.00	150.00	24.74	50.00	100.00	150.00
250	10	12.55	25.11	37.62	12.50	25.00	37.50	5.25	12.50	25.00	37.50
	20	25.05	50.06	75.05	25.00	50.00	75.00	13.65	25.00	50.00	75.00
	30	37.52	75.02	112.54	37.50	75.00	112.50	23.72	37.50	75.00	112.50
	40	50.03	100.03	150.03	50.00	100.00	150.00	35.69	50.00	100.00	150.00
	50	62.51	125.02	187.52	62.50	125.00	187.50	48.19	62.50	125.00	187.50
300	10	15.04	30.08	45.05	15.00	30.00	45.00	8.56	15.00	30.00	45.00
	20	30.05	60.09	90.06	30.00	60.00	90.00	23.18	30.00	60.00	90.00
	30	45.01	90.03	135.07	45.00	90.00	135.00	39.63	45.00	90.00	135.00
	40	60.02	120.03	180.02	60.00	120.00	180.00	57.49	60.00	120.00	180.00
	50	75.02	150.01	225.03	75.00	150.00	225.00	84.26	75.00	150.00	225.00
350	10	17.57	35.12	52.57	17.50	35.00	52.50	13.42	17.50	35.00	52.50
	20	35.09	70.03	105.02	35.00	70.00	105.00	30.86	35.00	70.00	105.00
	30	52.56	105.05	157.52	52.50	105.00	157.50	60.53	52.50	105.00	157.50
	40	70.03	140.04	210.04	70.00	140.00	210.00	85.70	70.00	140.00	210.00
	50	87.56	175.00	262.52	87.50	175.00	262.50	126.35	87.50	175.00	262.50
400	10	20.11	40.08	60.10	20.00	40.00	60.00	18.75	20.00	40.00	60.00
	20	40.09	80.05	120.07	40.00	80.00	120.00	49.25	40.00	80.00	120.00
	30	60.03	120.05	180.06	60.00	120.00	180.00	87.66	60.00	120.00	180.00
	40	80.03	160.05	240.05	80.00	160.00	240.00	132.54	80.00	160.00	240.00
	50	100.00	200.05	300.02	100.00	200.00	300.00	182.26	100.00	200.00	300.00
450	10	22.60	45.07	67.58	22.50	45.00	67.50	26.36	22.50	45.00	67.50
	20	45.04	90.04	135.03	45.00	90.00	135.00	70.96	45.00	90.00	135.00
	30	67.53	135.03	202.57	67.50	135.00	202.50	121.36	67.50	135.00	202.50
	40	90.09	180.05	270.04	90.00	180.00	270.00	179.74	90.00	180.00	270.00
	50	112.55	225.03	337.51	112.50	225.00	337.50	249.74	112.50	225.00	337.50
500	10	25.13	50.11	75.12	25.00	50.00	75.00	36.54	25.00	50.00	75.00
	20	50.01	100.02	150.05	50.00	100.00	150.00	84.77	50.00	100.00	150.00
	30	75.03	150.00	225.10	75.00	150.00	225.00	156.72	75.00	150.00	225.00
	40	100.06	200.01	300.04	100.00	200.00	300.00	229.99	100.00	200.00	300.00
	50	125.05	250.04	375.04	125.00	250.00	375.00	344.56	125.00	250.00	375.00
Average		41.29	82.54	123.79	41.25	82.50	123.75	54.64	41.25	82.50	123.75

As expected, the CPU times employed by all three methods that stop at $t = 10$ are almost identical. The same applies to $t = 20$ and $t = 30$. FRB5 has a rather erratic stopping time. This is because the method stops when the local search of Figure 3 reaches a local optimum and this depends on the stochastic order in which the jobs are inserted and on the instance data. Also, the local search is applied after each job is inserted in the NEH method which is extremely lengthy for larger instances. As a matter of fact, for instances with 500 jobs, FRB5 is actually slower than most methods. In any case, when comparing FRB5 with FRB3 we see that the added CPU time produces better results as the $\overline{RPD}$ of FRB5 is 1.36 versus that of FRB3 at 1.75.
An striking outcome are the results of DDE (DDELS as named in Pan and Wang, 2008b). The $\overline{RPD}$ does not improve from the original 2.65 given by DDE^{10} as DDE^{30} results in 2.65 as well. We checked our implementation carefully and found no errors. Furthermore, DDE shares about 90% of the code with the HDPSO method by the same authors, since both use the NEH for initialization, PTL crossover, insertion mutation and IG for local search. The problem is that, at each generation, two populations of size PS are generated. One by applying mutation to the original population, and another one by applying crossover. Then the original population and the two newly created ones undergo selection so to create a single population with PS individuals for the next generation. Only better individuals are passed over. After this, the best individual undergoes several iterations of the IG local search. Our observations indicate that when a certain level of evolution has been achieved, mutation and crossover only deteriorate individuals and after selection, the new PS population is exactly equal to the original population and the algorithm stalls. Obviously, this is a design shortcoming. Our hypothesis is that a selective local search should be applied to good individuals in the mutated and crossed populations, before actually applying selection.
In any case, such improvements are not necessary as the overall performance of DDE is not high. Much faster and simpler methods like GH_BM2 and FRB4_{12} are comparable as far as $\overline{RPD}$ is concerned and are faster in return. Also, FRB3 is both faster and better performing. As a result, DDE is not recommended over other algorithms for the no-idle PFSP.

Lastly, we comment on the performance of HDPSO and IG_{LS}. First of all, it must be reminded that IG_{LS} is used as a subroutine in HDPSO. Since we are stopping both algorithms at the same elapsed CPU time and since both share most code, the results are completely comparable. As we can see, the stand alone IG_{LS} gives significantly better results than HDPSO. Measuring the average percentage deviation between HDPSO and IG_{LS} we have that HDPSO^{10} is a full 20% worse than IG_{LS}^{10} since the two $\overline{RPD}$ values are 0.78 and 0.65, respectively. What is more, the performance lead of IG_{LS} widens as more CPU time is allowed. For example, HDPSO^{20} is 32.56% worse than IG_{LS}^{20} and HDPSO^{30} is 41.18% worse than IG_{LS}^{30}. From the 1250 available results (250 instances and 5 replicates), IG_{LS}^{30} produces better results than

HDPSO^{30} in 736 cases, equal results in 150 cases and worse results in 364 cases. Most importantly, we want to strongly draw our attention to these last type of measurements. Counting "the number of times" a given method is better, equal or worse than another is not an indicator of performance. It is a strongly biased measure and can mislead conclusions. The average percentage deviation of IG_{LS}^{30} over HDPSO^{30} in these 364 cases in which IG_{LS}^{30} gives a worse solution is a mere 0.24%. Therefore, it is easy to see that IG_{LS}^{30} is many times better (and by large) than HDPSO^{30} and for the times where it is worse, it is by a small amount.
Summing up, the Particle Swarm part of the algorithm is actually hindering results. A simple, easy to code and straightforward IG method works much better by itself.

4.3 Statistical analysis of results

As mentioned in previous Sections, careful statistical testing is necessary to really ascertain the observed differences in average values. While it is expected, for example, that SGM will be statistically worse than all other methods, concluding the same when comparing two methods of similar performance like GH_BM2 and $FRB4_{12}$ is risky to say the least.

One of the most powerful and tested methodologies is the Design of Experiments (DOE), Montgomery (2005). DOE is a structured and organized method for determining the relationship between factors affecting the output of a process. In our case, we are interested in studying the effect on the response variable RPD. From the 15,000 data points available from the computational evaluation of the previous Section, we carry out a full factorial analysis where the effect of the following factors is studied:

- Number of jobs n
- Number of machines m
- Algorithm, (NEH, KK, GH_BM, GH_BM2, FRB3, $FRB4_4$, $FRB4_{12}$, HDPSO, DDE, FRB5 and IG_{LS})
- Stopping criterion t

Note that the last factor can only be studied in conjunction with the algorithms HDPSO, DDE and IG_{LS}. We have eliminated from the tests NEH_{na} and GH_BM2_{na} as they give the same results than the accelerated versions. Also, SGM is not tested as it is clear that its results are far worse than the others.
The initial means plot with Tukey 95% confidence intervals is shown in Figure 4. Recall that overlapping intervals for means indicates that the observed means are statistically equivalent. It has to be noted that the means plot of Figure 4 is not explicitly considering the interactions of the algorithms with the different n and m values. Therefore, it is an "overall" picture. For example, the means plots of IG_{LS}^{30} and IG_{LS}^{20} overlap. This means that for the

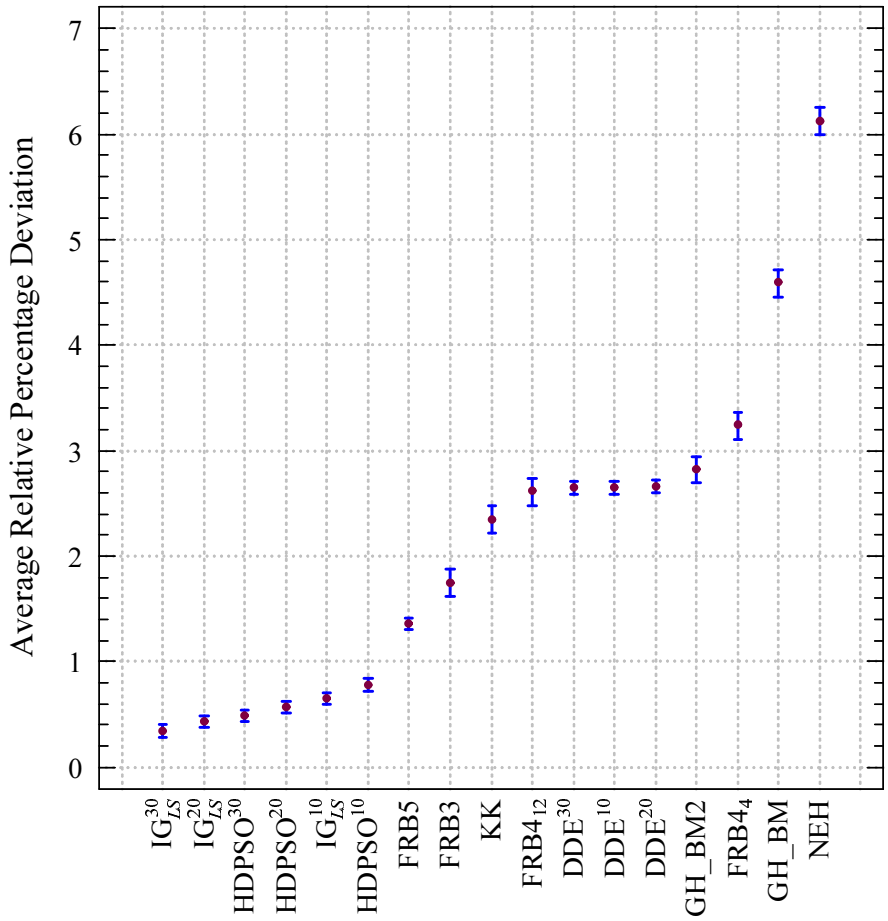

Fig. 4. Means plot for the algorithms $\overline{RPD}$ and 95% Tukey confidence intervals.

overall observed $\overline{RPD}$, there is no statistically significant difference. However, zooming-in for different levels of n and m we observe statistically significant differences. We will provide some additional plots later.
What can be concluded is that most observed differences are statistically significant. For example NEH is statistically worse than all other methods. GH_BM is the second worst and there is a statistically significant difference between $FRB4_4$ and GH_BM2. However, all DDE methods are statistically equivalent to GH_BM2 and $FRB4_{12}$. We also see how IG_{LS}^{30} is indeed statistically better than $HDPSO^{30}$. All in all, most observed averages are statistically different.

Of high interest is to study which instance sizes affect algorithms the most. This information is not easy to see from large tables full of numbers. Instead,

we give an interaction plot of factors n and m in Figure 5. We have to proceed

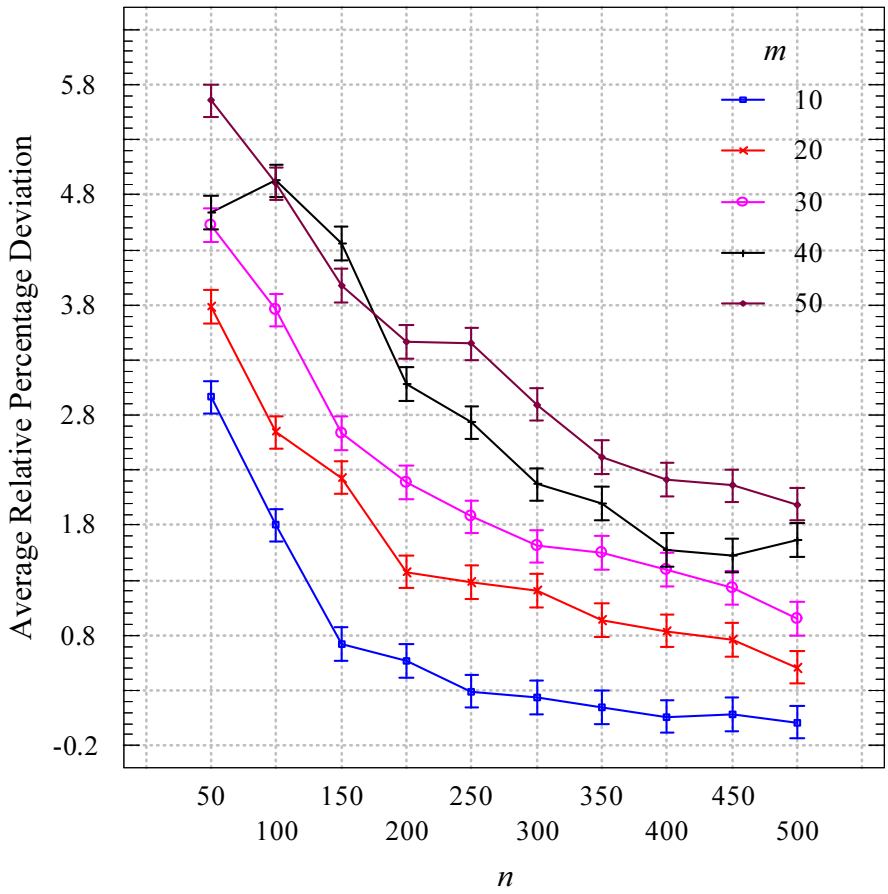

Fig. 5. Interaction plot between factors n and m with 95% Tukey confidence intervals.

with caution in the analysis of this plot. Since we do not know the optimum solution, we cannot state which instances are harder in an absolute way. Instead, we can point which combinations of n and m result in higher and statistically significant $\overline{RPD}$ values for all algorithms. It is clear that increasing the number of machines results in higher percentage deviations. Interestingly, increasing the number of jobs results in lower percentage deviations. We hypothesize that with a larger number of jobs there are more options to come up with a better schedule even though the search space becomes larger. By far, the "hardest" instances are those with 50 jobs and 50 machines. There are not so many jobs and therefore fitting 50 no-idle machines becomes daunting

for all methods.

Lastly, we zoom-in the performance of the two best methods, IG_{LS} and HDPSO. We plot the average performance against the different values of t in Figure 6. As can be seen, there is a clear statistically significant difference

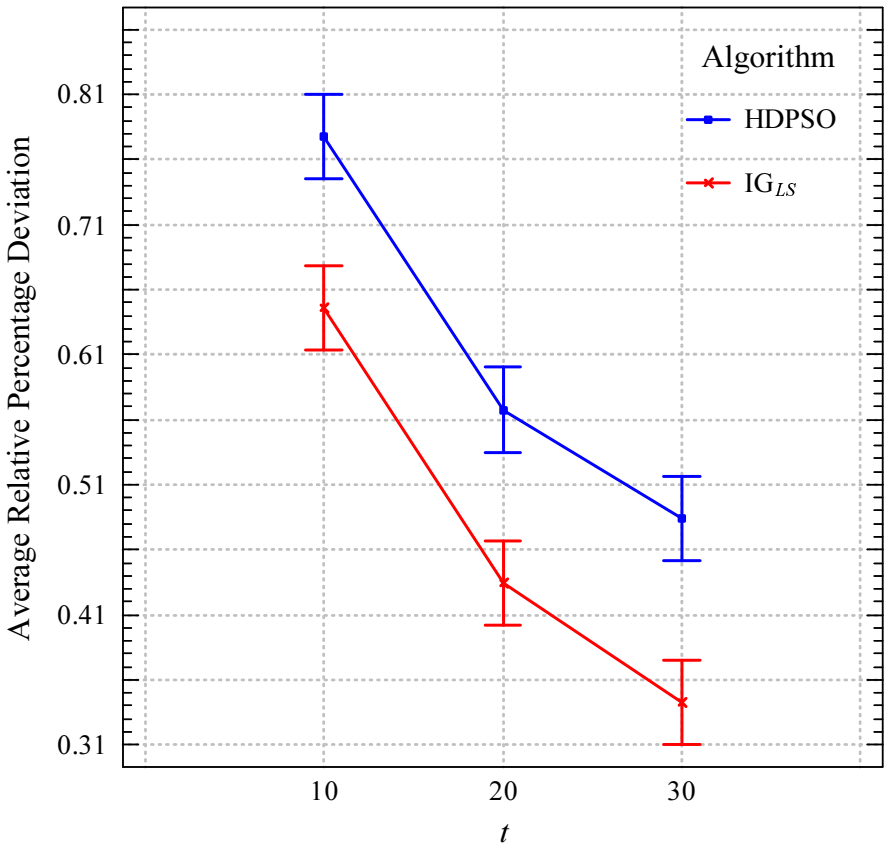

Fig. 6. Interaction plot between the algorithms IG_{LS} and HDPSO and t with 95% Tukey confidence intervals.

between IG_{LS} and HDPSO for all tested t values. As a matter of fact, for some instances sizes –not shown here– IG_{LS}^{20} is statistically better than $HDPSO^{30}$ which effectively means that IG_{LS} is able to reach better quality results when given one third less CPU time than HDPSO.

5 Conclusions and future research

This chapter has focused in a flowshop problem variant where idle times are not allowed on machines. This problem, known as the no-idle permutation flowshop, has been much less studied than the regular counterpart. We have provided a critical review of the existing literature, where each proposed algorithm has been carefully studied, and in some cases, improved. Namely, we have discussed a very effective implementation of the SGM method by Saadani et al. (2005) that despite not having improved its computational complexity of $\mathcal{O}(n^3)$, it has shown much lower empirical CPU running times when compared to other methods with better theoretical computational complexities. We have also provided an enhanced version of the GH_BM heuristic of Baraz and Mosheiov (2008). This improved version –referred to as GH_BM2– is much faster and effective than the original. Along with these improvements, we have made adaptations of methods that have been published very recently for the regular permutation flowshop problem. More specifically, we have adapted the Iterated Greedy (IG) metaheuristic from Ruiz and Stützle (2007) to the no-idle version. Some of the recent heuristics proposed in Rad et al. (2009) have also been adapted.
A total of 14 methods have been evaluated in a comprehensive computational campaign. State-of-the-art algorithms have been identified and validated through thorough statistical analyses. As the results indicate, adapted IG algorithms, as well as the proposed improved GH_BH2, together with the recent heuristics from Rad et al. (2009) constitute the best existing methods up to date for the no-idle permutation flowshop problem with makespan criterion.

There are many open research lines as this interesting problem variant has been seldom studied in the literature. No metaheuristic approaches have been proposed for other objectives apart from makespan. Furthermore, no-idle constraints in other environments like hybrid flowshops or job shops have not been studied. Additionally, more research is needed in exact methodologies, bounds and mathematical approaches for no-idle constraints. This way, researchers would benefit from a better understanding and characterization of this interesting problem.

References

Adiri, I. and Pohoryles, D. (1982). Flowshop no-idle or no-wait scheduling to minimize the sum of completion times. *Naval Research Logistics*, 29(3):495–504.

Baker, K. R. (1974). *Introduction to Sequencing and Scheduling.* John Wiley & Sons, New York.

Baptiste, P. and Hguny, L. K. (1997). A branch and bound algorithm for the $F/no-idle/C_{max}$. In *Proceedings of the International Conference on Industrial*

Engineering and Production Management, IEPM'97, volume 1, pages 429–438, Lyon, France.

Baraz, D. and Mosheiov, G. (2008). A note on a greedy heuristic for flow-shop makespan minimization with no machine idle-time. *European Journal of Operational Research*, 184(2):810–813.

Campbell, H. G., Dudek, R. A., and Smith, M. L. (1970). A heuristic algorithm for the n job, m machine sequencing problem. *Management Science*, 16(10):B630–B637.

Cheng, M. B., Sun, S. J., and He, L. M. (2007a). Flow shop scheduling problems with deteriorating jobs on no-idle dominant machines. *European Journal of Operational Research*, 183(1):115–124.

Cheng, M. B., Sun, S. J., and Yu, Y. (2007b). A note on flow shop scheduling problems with a learning effect on no-idle dominant machines. *Applied Mathematics and Computation*, 184(2):945–949.

Dannenbring, D. G. (1977). An evaluation of flow shop sequencing heuristics. *Management Science*, 23(11):1174–1182.

Davoud Pour, H. (2001). A new heuristic for the n-job, m-machine flow-shop problem. *Production Planning & Control*, 12(7):648–653.

Du, J. and Leung, J. Y.-T. (1990). Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, 15(3):483–495.

Framinan, J. M., Gupta, J. N. D., and Leisten, R. (2004). A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55:1243–1255.

Framinan, J. M. and Leisten, R. (2008). A multi-objective iterated greedy search for flowshop scheduling with makespan and flowtime criteria. *In press at OR Spectrum*.

Framinan, J. M., Leisten, R., and Rajendran, C. (2003). Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *International Journal of Production Research*, 41(1):121–148.

Garey, M. R., Johnson, D. S., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129.

Giaro, K. (2001). NP-hardness of compact scheduling in simplified open and flow shops. *European Journal of Operational Research*, 130(1):90–98.

Gonzalez, T. and Sahni, S. (1978). Flowshop and jobshop schedules: Complexity and approximation. *Operations Research*, 26(1):36–52.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326.

Hejazi, S. R. and Saghafian, S. (2005). Flowshop-scheduling problems with makespan criterion: A review. *International Journal of Production Research*, 43(14):2895–2929.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1:61–68.

Kalczynski, P. J. and Kamburowski, J. (2005). A heuristic for minimizing the makespan in no-idle permutation flow shops. *Computers & Industrial Engineering*, 49(1):146–154.

Kalczynski, P. J. and Kamburowski, J. (2007). On no-wait and no-idle flow shops with makespan criterion. *European Journal of Operational Research*, 178(3):677–685.

Kamburowski, J. (2004). More on three-machine no-idle flow shops. *Computers & Industrial Engineering*, 46(3):461–466.

Koulamas, C. (1998). A new constructive heuristic for the flowshop scheduling problem. *European Journal of Operational Research*, 105:66–71.

Liao, C. J. (1993). Minimizing the number of machine idle intervals with minimum makespan in a flowshop. *Journal of the Operational Research Society*, 44(8):817–824.

Minella, G., Ruiz, R., and Ciavotta, M. (2008). A review and evaluation of multi-objective algorithms for the flowshop scheduling problem. *In press at INFORMS Journal on Computing*.

Montgomery, D. (2005). *Design and Analysis of Experiments*. John Wiley & Sons, New York, sixth edition.

Narain, L. and Bagga, P. C. (2003). Minimizing total elapsed time subject to zero total idle time of machines in $n \times 3$ flowshop problem. *Indian Journal of Pure & Applied Mathematics*, 34(2):219–228.

Narain, L. and Bagga, P. C. (2005a). Flowshop/no-idle scheduling to minimise the mean flowtime. *Anziam Journal*, 47:265–275.

Narain, L. and Bagga, P. C. (2005b). Flowshop/no-idle scheduling to minimize total elapsed time. *Journal of Global Optimization*, 33(3):349–367.

Narasimhan, S. L. and Mangiameli, P. M. (1987). A comparison of sequencing rules for a two stage hybrid flowshop. *Decision Sciences*, 18(2):250–265.

Narasimhan, S. L. and Panwalkar, S. S. (1984). Scheduling in a two stage manufacturing process. *International Journal of Production Research*, 22(4):555–564.

Nawaz, M., Enscore, Jr, E. E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, The International Journal of Management Science*, 11(1):91–95.

Niu, Q. and Gu, X. S. (2006). An improved genetic-based particle swarm optimization for no-idle permutation flow shops with fuzzy processing time. In *PRICAI 2006: Trends in Artificial Intelligence. 9th Pacific Rim International Conference on Artificial Intelligence. Lecture Notes in Computer Science*, volume 4099, pages 757–766, Guilin, China. Springer.

Osman, I. and Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. *OMEGA, The International Journal of Management Science*, 17(6):551–557.

Page, E. S. (1961). An approach to the scheduling of jobs on machines. *Journal of the Royal Statistical Society, B Series*, 23(2):484–492.

Palmer, D. S. (1965). Sequencing jobs through a multi-stage process in the minimum total time - a quick method of obtaining a near optimum. *Operational Research Quarterly*, 16(1):101–107.

Pan, Q.-K. and Wang, L. (2008a). No-idle permutation flow shop scheduling based on a hybrid discrete particle swarm optimization algorithm. *In press at International Journal of Advanced Manufacturing Technology*.

Pan, Q.-K. and Wang, L. (2008b). A novel differential evolution algorithm for no-idle permutation flow-shop scheduling problems. *European Journal of Industrial Engineering*, 2(3):279–297.

Pan, Q.-K., Wang, L., and Zhao, B.-H. (2008). An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. *In press at International Journal of Advanced Manufacturing Technology*.

Rad, S. F., Ruiz, R., and Boroojerdian, N. (2009). New high performing heuristics for minimizing makespan in permutation flowshops. *OMEGA, the International Journal of Management Science*, 37:331–345.

Reeves, C. R. (1995). A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1):5–13.

Ruiz, R. and Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165:479–494.

Ruiz, R., Maroto, C., and Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *OMEGA, the International Journal of Management Science*, 34:461–476.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

Ruiz, R. and Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143–1159.

Saadani, N. E. H. and Baptiste, P. (2002). Relaxation of the no-idle constraint in the flow-shop problem. In *Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM'97*, pages 305–309, Lyon, France.

Saadani, N. E. H., Guinet, A., and Moalla, M. (2001). A travelling salesman approach to solve the $F/no-idle/C_{max}$ problem. In *Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM'01*, volume 2, pages 880–888, Quebec, Canada.

Saadani, N. E. H., Guinet, A., and Moalla, M. (2003). Three stage no-idle flow-shops. *Computers & Industrial Engineering*, 44(3):425–434.

Saadani, N. E. H., Guinet, A., and Moalla, M. (2005). A travelling salesman approach to solve the $F/no-idle/C_{max}$ problem. *European Journal of Operational Research*, 161(1):11–20.

Salveson, M. E. (1952). On a quantitative method in production planning and scheduling. *Econometrica*, 20(4):554–590.

Suliman, S. M. A. (2000). A two-phase heuristic approach to the permutation flow-shop scheduling problem. *International Journal of Production Economics*, 64:143–152.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47:67–74.

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

Tanaev, V. S., Sotskov, Y. N., and Strusevich, V. A. (1994). *Scheduling Theory. Multi-Stage Systems*. Kluwer Academic Publishers, Dordrecht.

Turner, S. and Booth, D. (1987). Comparison of heuristics for flow shop sequencing. *OMEGA, The International Journal of Management Science*, 15(1):75–78.

Vachajitpan, P. (1982). Job sequencing with continuous machine operation. *Computers & Industrial Engineering*, 6(3):255–259.

Vallada, E. and Ruiz, R. (2008). Cooperative metaheuristics for the permutation flowshop scheduling problem. *In press at European Journal of Operational Research*.

Vallada, E., Ruiz, R., and Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research*, 35(4):1350–1373.

Čepek, O., Okada, M., and Vlach, M. (2000). Note: On the two-machine no-idle flowshop problem. *Naval Research Logistics*, 47(4):353–358.

Wang, Z. B., Xing, W. X., and Bai, F. S. (2005). No-wait flexible flowshop scheduling with no-idle machines. *Operations Research Letters*, 33(6):609–614.

Woollam, C. R. (1986). Flowshop with no idle machine time allowed. *Computers & Industrial Engineering*, 10(1):69–76.

Ying, K.-C. (2008). An iterated greedy heuristic for multistage hybrid flowshop scheduling problems with multiprocessor tasks. *In press at Journal of the Operational Research Society*.

Index