Computers & Operations Research 37 (2010) 236 – 246

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

Algorithms for a realistic variant of flowshop scheduling

B. Naderi[a,*], Rubén Ruiz[b], M. Zandieh[c]

[a]Department of Industrial Engineering, Amirkabir University of Technology, 424 Hafez Avenue, Tehran, Iran
[b]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
[c]Department of Industrial Management, Management and Accounting Faculty, Shahid Beheshti University, Tehran, Iran

ARTICLE INFO

Available online 12 May 2009

Keywords:
Scheduling
Hybrid flexible flowshops
Sequence dependent setup times
Dynamic dispatching rule heuristic
Iterated local search metaheuristic

ABSTRACT

This paper deals with a realistic variant of flowshop scheduling, namely the hybrid flexible flowshop. A hybrid flowshop mixes the characteristics of regular flowshops and parallel machine problems by considering stages with parallel machines instead of having one single machine per stage. We also investigate the flexible version where stage skipping might occur, i.e., not all stages must be visited by all jobs. Lastly, we also consider job sequence dependent setup times per stage. The optimization criterion considered is makespan minimization. While many approaches for hybrid flowshops have been proposed, hybrid flexible flowshops have been rarely studied. The situation is even worse with the addition of sequence dependent setups. In this study, we propose two advanced algorithms that specifically deal with the flexible and setup characteristics of this problem. The first algorithm is a dynamic dispatching rule heuristic, and the second is an iterated local search metaheuristic. The proposed algorithms are evaluated by comparison against seven other high performing existing algorithms. The statistically sound results support the idea that the proposed algorithms are very competitive for the studied problem.

1. Introduction

Scheduling problems deal with the allocation of resources to perform a set of activities over a period of time [1]. Flowshop scheduling is among the most studied scheduling settings. In this problem, a set of n jobs need to be processed in a set of m machines disposed in series. All jobs follow the same processing route through the machines, i.e., they are first processed on machine 1, then on machine 2 and so on until machine m. However, flowshops have been frequently criticized because of their excessive simplicity. In the literature we find direct critics to the flowshop model as in [2]. Real production floors rarely employ a single machine for each operation. As a result, the regular flowshop problem is many times extended with a set of–usually identical–parallel machines at each stage, i.e., instead of having a series of machines, we have a series of stages. The purpose of duplicating machines in parallel is to increase the throughput and capacity of the shop floor, to balance the speed of the stages, or to either eliminate or to reduce the impact of bottleneck stages on the overall shop efficiency. It is also frequent in practice to have optional treatments for products, like polishing or additional decorations in ceramic tiles to name just an example [3]. The possibility of not executing these optional treatments or operations is known as stage skipping.

Another frequent situation faced in production floors are setup times. These are all non-productive operations carried out on machines in order to prepare them for the production of the next product in the sequence. Often, these setup times are sequence dependent, something referred to as SDST in short. According to the reviews of Allahverdi et al. [4,5], research in setup times is attracting more and more interest over time. In any case, and also according to these reviews, research is mostly confined to regular scheduling problems and research on complex and realistic scheduling settings is scarce. Sequence dependent setup times significantly increase the difficulty of scheduling problems. For example, for the regular permutation flowshop problem and makespan criterion, problem denoted as $F/prmu/C_{max}$ according to the well known three field notation of Graham et al. [6], Haouari and Ladhari [7] proposed a Branch and Bound algorithm capable of solving large instances of up to 200 jobs and 10 machines (200×10). However, just adding sequence dependent setup times–the F/S_{ijk}, $prmu/C_{max}$ problem–results in a significantly more complex setting. After much research efforts, like those reported in [8,9], instances of a maximum size of only 10×9 are solved to optimality. Obviously, setup times have to be explicitly considered in solution procedures if high quality results are to be obtained. This seems obvious, but there is a great difference between

* Corresponding author.
E-mail address: bahman.naderi@aut.ac.ir (B. Naderi).

0305-0548/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2009.04.017

accounting for these setups in the solutions and devising methods that specially consider setups. Allahverdi and Soroush [10] have recently studied the effect and importance of considering setups in scheduling research.

More formally, we define the sequence dependent setup times hybrid flexible flowshop (SDST/HFFS) as follows: a set N of jobs, $N=\{1,\ldots,n\}$ have to be processed in a set M of stages, $M=\{1,\ldots,m\}$. At every stage i, $i \in M$ we have a set $M_i=\{1,\ldots,m_i\}$ of identical parallel machines. Every machine at each stage is capable of processing all the jobs. Each job has to be processed in exactly one out of the m_i identical parallel machines at each stage i. However, not every job has to visit each stage. We denote by F_j the set of stages that job j, $j \in N$ has to visit. Obviously, $1 \leq |F_j| \leq m$. Furthermore, since we are dealing with a flowshop, only stage skipping is allowed, for example, it is not possible for a given job to visit stages {1, 2, 3} and another one to visit stages {3, 2, 1}. The processing time of job j at stage i is denoted by p_{ij}. Lastly, S_{ijk} denotes the setup time between jobs j and k, $k \in N$ at stage i. The optimization criterion is the minimization of the maximum completion time or makespan. C_j denotes the completion time of job j, which is basically when the job is finished at its last visited stage. The makespan is calculated as $C_{max}=\max_{j\in N}\{C_j\}$. Vignier et al. [11] extended the three field notation for hybrid flowshops. With this in mind, the SDST/HFFS problem studied in this paper can be denoted as $HFFSm$, $((PM^{(i)})_{i=1}^{m})/F_j$, S_{ijk}/C_{max}. Notice that makespan criterion has been sometimes criticized for not being realistic. However, in a make-to-stock environment, where products are manufactured to refill stocks, increasing throughput (which is related to minimizing makespan) is a very important measure. Furthermore, in short term finite capacity scheduling, due dates might be distant in the future and it might be easy to fulfill all of them. In all these scenarios, minimizing makespan is preferable. Realistic scheduling with makespan criterion has been also studied and motivated by Ruiz and Maroto [3] and Ruiz et al. [12].

Additionally, the following usual assumptions are also considered in the SDST/HFFS:

- All jobs are independent and available for processing at time 0.
- All machines are continuously available.
- Each machine can only process a job at a time. Jobs can only be processed by one machine at a time.
- The processing of a job on a machine cannot be interrupted, i.e., no preemption is allowed.
- There are infinite buffers in between all stages, if a job needs a machine that is occupied, it waits indefinitely until it is available.
- There are no transportation times between stages.

Gupta [13] showed the flowshop with multiple processors (FSMP) problem with only two stages ($m=2$) to be NP-hard even when one of the two stages contains a single machine. The FSMP can be considered as a special case of the SDST/HFFS problem studied in this paper and therefore, the SDST/HFFS is also NP-hard.

As we will see, the SDST/HFFS problem has been sparsely studied in the literature and no exact approaches have been proposed for its solution. In any case, it is unlikely that any such approaches will be able to solve large instances, given the abundant literature on exact methods for simpler problems with setup times and the limited results obtained so far. However, some heuristics and metaheuristics have been proposed for this problem like random keys genetic algorithms and heuristics [14,15], genetic algorithms [3], and artificial immune systems [16]. In this paper we exploit the flexible and setup time characteristics of the SDST/HFFS in order to obtain competitive results.

The rest of the paper is organized as follows: Section 2 provides a brief literature review of the SDST/HFFS problem. Section 3 details the proposed algorithms. Section 4 presents the experimental design and results. Finally, Section 5 concludes the paper.

2. Literature review of SDST hybrid flexible flowshops

Since Johnson's pioneering work [17] on the two-machine regular permutation flowshop, a large number of studies have been published about scheduling. Citing only the review papers that have appeared in the last years will need probably several references. Recall that the SDST/HFFS problem considered in this paper has three main characteristics that are jointly considered: hybrid setting, where there are parallel machines at each stage; flexibility, where stages might be skipped; and sequence dependent setup times. In the literature, we find reviews of each one of these three characteristics and we refer the reader to them for more details. Hybrid flowshops are reviewed in [11,18]. More recently, Kis and Pesch [19] have published a review about exact techniques for hybrid flowshops. Notice that the literature on hybrid flowshops, with identical parallel machines per stage and no setups, is plenty. Some recent references are the study of Haouari and Hidri [20] in lower bounds or the metaheuristics proposed by Jin et al. [21]. Flexible flowshops, often referred to as flexible flow line problems, are reviewed in [22], among others. Lastly, and as commented, setup time scheduling is excellently reviewed by Allahverdi et al. [4,5].

However, from all these reviews and to the best of our knowledge, the studies about SDST/HFFS–which is, in reality, a generalization of all the previous problems–are really scarce. Kurz and Askin [14] consider dispatching rules for the SDST/HFFS. They investigate three classes of heuristics based on simple greedy methods, insertion heuristics and adaptations of Johnson's rule. Later, Kurz and Askin [15] formulate the SDST/HFFS as an integer programming (IP) model. Because of the difficulty in solving the IP model directly, they develop a random keys genetic algorithm (RKGA). Problem data are generated to compare the RKGA against other heuristic rules. Zandieh et al. [16] propose an immune algorithm, and compare it favorably against the RKGA of Kurz and Askin [15].

In the literature, we find studies about related problems, often considering even more complex settings and/or slight variations of the SDST/HFFS problem. Ruiz and Maroto [3] consider SDST hybrid flowshops with machine eligibility to minimize makespan. The difference is that they consider unrelated parallel machines at each stage where the setup times are job sequence and machine dependent. Additionally, all jobs must visit all stages as they do not deal with flexibility. They propose a calibrated genetic algorithm (GA). The authors demonstrate the high performance of this GA by comparing it against several other algorithms. Recently, Jungwattanakit et al. [23] have proposed an IP model, dispatching rules and a genetic algorithm for a SDST hybrid flowshop with a weighted sum of makespan and number of tardy jobs as an optimization criterion. The authors consider the unrelated parallel machines case, and compare all the methods they propose against a rather limited benchmark of a maximum size of 50 jobs and 20 stages (however, they never state how many uniform parallel machines there are per stage). The same authors study the same problem in [24]. This time, they add some simulated annealing and tabu search algorithms to the comparison. Naderi et al. [25] study hybrid flowshops with setup times where no flexibility is considered. They study several simulated annealing operators using the Taguchi method. Also recently, Ruiz et al. [12] consider a realistic case of hybrid flexible flowshops with unrelated machines, setup times and many other additional characteristics like overlaps, precedence relationships and others. They present a mixed integer programming model and some heuristics.

3. Proposed algorithms

Before going into details, we remind and discuss two important decisions that have to be taken when scheduling in hybrid flowshops:

(1) Determination of the job sequence at the beginning of each stage.
(2) Assignment of jobs to machines at each stage.

In regular flowshops, a permutation of jobs for each machine is the only thing needed, at least for completion time-based criteria. Furthermore, regular flowshops are most of the time simplified to permutation flowshops where a single permutation sequence is used for all machines. In the HFFS problem, a single permutation of jobs, where this permutation is maintained through all stages, results in poor performance. The permutation simplification in flowshops is effective because the first job in stage/machine 1 has the lowest completion time and consequently the lowest ready time for stage/machine 2. In the HFFS problem, it is not unlikely for the first job in stage 1 to have a greater completion time than other succeeding jobs in the sequence. As a result, a better method is needed to determine the job sequence at the beginning of each stage. For the HFFS problem, the job sequence at stage 1 is normally determined by the outcome of the scheduling algorithms. For subsequent stages, the jobs are sorted in increasing value of their completion times in the previous stage. This is a common approach in the literature, see for example [15,23]. Notice that the ready times of the jobs at the beginning of each stage t, $t=\{2,\ldots,m\}$ are equal to the completion times at stage $t-1$.

In regular flowshop problems, there is no need to assign jobs to machines since there is only one machine per stage. In the HFFS, with potentially more than one machine per stage, we have to make this additional decision. Actually, an HFFS can be seen as a series of parallel machine problems. In the case of HFFS with identical parallel machines per stage, a possibility is to assign the jobs at each stage to the first available machine (FAM). FAM technique in HFFS with no setups results in the earliest completion time for the jobs in that stage. However, in the case of the SDST/HFFS, the FAM rule is very unlikely to be an effective technique because it is possible that the setup time on the first available machine for a given job is large, and this machine can result in a later completion time compared with other machines. The reason behind this is that the machines at each stage, despite being identical, would have different previous jobs assigned and the new job to be assigned incurs in a job-sequence dependent setup time. Therefore, each job is assigned to the machine that is able to complete the job at the earliest time in the given stage. This is referred to as the earliest completion time (ECT) rule. A similar approach is followed by Ruiz and Maroto [3]. Notice that the completion time of a given job j at stage i, denoted as C_{ij} takes into account the sequence dependent setup time, i.e., $C_{ij}=\max\{C_{i,j-1};C_{i-1,j}\}+S_{i,j-1,j}+p_{ij}$, where $C_{i,j-1}$ is the completion time of the previous job in the sequence that was assigned to the same machine as job j at stage i. Similarly, $C_{i-1,j}$ is the completion time of job j at the previous stage that this job visited.

As reviewed, having effective, yet clear and simple strategies for each one of the two previous decisions according to the features of the SDST/HFFS problem plays a key role in our proposed algorithms. These algorithms are described in the following sections.

3.1. Modified dynamic dispatching rule heuristic

One possible criticism of some of the previous reviewed algorithms from the literature is that their approach to setup times is rather reactive. Setup times are only realized once jobs are assigned, instead of actively searching for small setups. Clear examples are the heuristics and the RKGA of Kurz and Askin [14,15].

The proposed modified dynamic dispatching rule (MDDR) heuristic does not separate the decisions of job sequencing and machine assignment. Beginning with stage 1, all jobs visiting this first stage are considered. For these jobs, the minimum ECT is calculated for all available parallel machines at this stage. The job with the minimum ECT is scheduled and assigned to the machine producing this minimum ECT value. The process is repeated until all jobs have been assigned to machines. The process continues with stage 2, considering now the ready times for jobs at stage 2. This is repeated for all stages. Notice that it is a dynamic dispatching rule since the ECT values might change after scheduling each job. An outline of the MDDR heuristic is given in Fig. 1.

MDDR is a very fast method. At each stage, all n jobs are considered in the first step. Once the job with the minimum ECT is scheduled, the remaining $n-1$ jobs are considered again. Therefore we have $n(n+1)/2$ steps. At each step, the ECT values for all parallel machines have to be calculated. This process is repeated for all stages. As a result, the computational complexity of MDDR is $(n^2\sum_{i=1}^{m}m_i)$. In practice, after scheduling a job inside a stage, only the ECT values of the pending jobs have to be calculated. Furthermore, only the ECT values of the machine to which the job was assigned need recalculation. This does not change the theoretical worst case computational complexity as all ECT times have to be calculated at least once at the beginning of each stage, but still the empirical running time of the MDDR heuristic is expected to be low. As a last note, the previous computational complexity can only be easily derived when all jobs visit all stages. With flexibility, some jobs will be skipping stages and the complexity will be much lower. However, this complexity can only be derived for specific problem instances once the F_j sets are given. As a result, the given computational complexity is actually an upper bound on the complexity.

3.2. Iterated local search metaheuristic

Iterated local search (ILS) is a simple and effective type of metaheuristic. As the name implies, ILS iteratively applies local search in a special way. According to Hoos and Stützle [26], ILS is one of the simplest local search-based methods that prevents stagnation around local optima. ILS was applied with great success to regular flowshop problems with makespan criterion as early as in [27]. As a matter of fact, ILS has produced state-of-the-art results for regular flowshop problems as shown in the review of Ruiz and Maroto [28]. Furthermore, a related method referred to as Iterated Greedy (IG) has been recently proposed with excellent results as shown in [29,30]. As with many metaheuristics, ILS starts from a heuristically obtained solution, preferably from a high quality one. Usually, this initial solution undergoes a strong local search until a local optimum is found according to a given local search neighborhood definition. The main loop of ILS starts and three main operators are iteratively applied until a given stopping criterion is met. The first operator is referred to as *perturbation* and consists of changing the current solution in order to escape from the previous local optimum. The second operator is actually the local search. After perturbation and local search, hopefully a new–and possibly better–local optimum is found. At this stage, the third operator is applied. This operator is an acceptance criterion, which accepts or discards the new local optimum. Generally, the new solution is accepted if it is better than the previous one. Otherwise, it is usually discarded. Some ILS implementations accept worse solutions temporarily as in simulated annealing. For more details about ILS, the reader is referred to [31], among others.

It is very common to find in the related literature of hybrid flowshops complex and elaborated methodologies like genetic

```
Procedure MDDR_heuristic
    At the first stage all the ready times of jobs at machines are zero
    For i := 1 to m do
        N = all n jobs
        Ready times of jobs at stage i = completion times at stage i − 1
        While |N| > 0 do
            Find the ECT after assigning each pending job in N to each machine at stage i
            Assign job j to machine l resulting in the lowest ECT
            Extract job j from N
            Update the ready time of machine l at stage i
        endwhile
    endfor
```

Fig. 1. Pseudo algorithm of the MDDR heuristic.

```
Procedure Local_search
    i = 1
    While i ≤ n do
        x′= the sequence produced by relocating x_i to a different random position
        if C_max(x′) < C_max(x) then
            x = x′
            break
        endif
        i = i + 1
    endwhile
```

Fig. 2. Pseudo algorithm of the local search.

algorithms, simulated annealing, tabu search, and others where local search is only used as a surrogate method. However, the simple application of local search-based methods alone can produce state-of-the-art results. Some examples are the aforementioned IG method for the regular flowshop problem as shown in [29,30]. ILS and IG are very simple to code and have less operators and parameters, yet surprisingly show good performance in many scenarios.

Our proposed ILS algorithm starts from an initial solution, which will be detailed later, and then conducts the search in the solution space by applying two of the main mechanisms explained above: local search and *perturbation*. A simple local search is applied to a candidate solution x. This search stops as soon as a first improvement is found. When no improvements are found, a counter is increased. Obviously, each time the local search improves the current solution x, the counter is reset. The local search itself is now described: the job in the first position of the current sequence x, referred to as x_1 is relocated to a different randomly chosen position in the sequence. If this new sequence x' results in better makespan, the current solution x is replaced by x' and the local search ends. Otherwise, the procedure continues with x_2. The search iterates at most for all jobs, without repetition. The rationale behind this procedure is to have a very fast local search procedure. The general outline of the applied local search is detailed in Fig. 2.

Note that the stochastic nature of the local search impedes a precise computational complexity analysis. However, in the worst case in which n jobs are relocated with no improvements, the complexity is $O(n^2 \times m)$.

The second mechanism employed is the ILS *perturbation*. We carry out this procedure whenever the local search counter exceeds a threshold value referred to as *no_change*, i.e., *perturbation* is triggered when a number of unsuccessful local search iterations have been carried out. As noted in [31], *perturbation* should be large enough so to escape from a local optimum found in the local search phase but at the same time not too large so as not to convert the ILS in a random search procedure. Usually, and as explained in [27], the *perturbation* is just a collection of "moves" that complement those carried out by the local search. A weak *perturbation* is likely to result in a local search that gets rapidly stuck in a deep local optimum, whereas a strong *perturbation* results in a local search that is prone to be slow in convergence and similar to a randomized search. Achieving such a delicate balance is a challenge and certainly it is the key to success in ILS.

We propose an elaborated *perturbation* operator which tries to excel in achieving the aforementioned balance. Our intention is also to direct our proposed ILS towards more promising regions in the search space. Our *perturbation* operator is just a collection of insertion movements. Namely, we randomly select a number d of jobs and relocate them to different random positions in the sequence. We refer to this *perturbation* operator as "giant leap" or GL. Note that this type of movement is not easily reachable by the proposed local search procedure. Notice that a collection of random "moves" will change the current solution in an unpredictable way. The result will be, most likely, a worse solution, and many times, much worse. In order to increase the chance of moving to a more promising region of the search space, we augment the GL operator as follows: we generate a number *nu_move* of perturbed solutions through GL, all of the perturbations of the incumbent solution. All these *nu_move* solutions are evaluated and the best one is accepted as the new perturbed solution. The rationale behind this operator is simple. Instead of accepting a new perturbed solution blindly, a solution that could be much worse and a possible waste of time, we carry out a random sampling and the best perturbed candidate is chosen. Notice that this perturbed solution is accepted even if it is worse than the incumbent solution, i.e., our GL *perturbation* already includes the acceptance operator of ILS. With these two main operators, the whole ILS procedure is given in detail in Fig. 3.

3.3. Encoding and initial solution

Two commonly used approaches for the representation of solutions in hybrid flowshop methods are direct and indirect encodings as shown, among others, in [32]. In a direct encoding, the representation string contains the complete solution to the problem while in an indirect encoding, the representation string contains data which is later used to obtain a solution. During the last decade, many encodings have been employed. However, for hybrid flowshops, the most common are the job-based representation (JBR) and random keys representation (RKR).

In JBR, the job sequence at stage 1 is represented by a simple permutation of jobs. This permutation indicates the order in which jobs are launched to the shop. Using a machine assignment rule (like for example ECT), jobs are assigned to the different machines. The job sequence for stage 1 can be maintained for all subsequent

```
Procedure Iterated_Local_Search
    counter = 0
    Initialize x (from NEHH)
    x_best = x
    while stopping criterion not met do
        x' = Local search(x)                          % local search
        if f(x') < f(x) do
            counter = 0
            x = x'
            if f(x') < f(x_best) do
                x_best = x'
            endif
        else
            counter = counter +1
            if counter > no_change do
                counter = 0
                for r := 1 to nu_move do              % perturbation
                    s̃(r) = GL_operator(x)
                endfor
                x = s̃_best
            endif
        endif
    endwhile
```

Fig. 3. General outline of proposed ILS.

2.86	1.48	2.12

Fig. 4. Representation of a candidate solution in ORK.

2.25	1.67	2.71	1.24	1.81	1.39

Fig. 5. Representation of a candidate solution in TRK.

stages as in [3]. However, as explained before, a better approach is to change the sequence of jobs for subsequent stages according to their earliest ready times in previous stages.

RKR was first proposed by Bean [33] and later used by Norman and Bean [34] for identical parallel machine problems. Each job is assigned a real number. The integer part of this number indicates the machine to which the job has been assigned. The fractional part is used to order the jobs assigned to each machine. This representation is extended to the SDST/HFFS problem by having real numbers for all stages (TRK) as in [15], or just for the first stage (ORK) [16]. In TRK all stages are sequenced and all jobs assigned to machines with the real numbers. ORK follows the previously commented approach, i.e., n real numbers are just used for assigning and sequencing jobs to machines at the first stage. For subsequent stages, the sequence of jobs is determined by their ready times and assignment is done with the ECT rule.

We illustrate these two representations with the following example with three jobs ($n=3$), two stages ($m=2$), two machines at stage one, and one machine at stage two ($m_1=2$ and $m_2=1$, respectively). ORK requires three random numbers as shown in Fig. 4. From this example, jobs 1 and 3 are assigned to machine 2 whereas job 2 is assigned to machine 1. Furthermore, the fractional part of job 3 is lower than that of job 1, therefore, job 3 is processed first on machine 2. For the second stage, jobs will be processed according to their earliest ready times and assigned to machines according to the ECT rule. TRK needs $n \times m$ random numbers as shown in Fig. 5 and does not need reordering of jobs or machine assignment rules.

A priori, JBR is the most indirect representation and TRK the most direct one. It is not clear which one will perform best in the proposed ILS algorithm. As a result, we first aim at comparing and analyzing the behavior of ILS with these three different encoding schemes. Note that changes in the encoding scheme also result in adaptations in all the previous operators. ILS with JBR encoding is referred to as ILS-P. The other two ILS methods are called ILS-TRK and ILS-ORK, respectively. JBR and ORK encodings have a potential drawback, which is redundancy, i.e., different permutations might result in the same solution after decoding. Furthermore, with stage skipping, this possibility is even greater. For example, suppose an instance with 10 jobs and 2 stages. If the probability for a job to skip a stage is 0.4, we expect around 4 jobs to skip stage 1. Again suppose that jobs 2, 4, 7, and 9 are those skipping the first stage. Therefore, they all have a ready time of zero at the beginning of stage 2 since they are not processed on stage 1. Consequently, the permutations {2, 8, 4, 1, 9, 3, 7, 10, 6, 5}, {8, 9, 1, 4, 3, 10, 6, 5, 2, 7}, {8, 1, 7, 3, 9, 10, 6, 4, 2, 5}, and all other permutations in which jobs 1, 3, 5, 6, 8, and 10 have the same relative sequence, represent the same solution as the positions of jobs 2, 4, 7, and 9 are irrelevant. In fact, from the total 10! possible solutions with the JBR encoding, (10!–6!) represent the same solution for this example with two stages. As a result, a large number of actually different permutations potentially represent the same solution.

The technique that we propose to overcome this redundancy is the following: if some jobs have the same ready times at the beginning of stage t, $t=\{2,\ldots,m\}$, we arrange them in the same relative order as in stage $t-1$, instead of permitting different orders that ultimately result in the same solution. This technique seems to be very effective in the presence of stage skipping. We refer to this technique as FECT and we will later show, in Section 4.4, its effectiveness.

Research on scheduling contains many papers that stress the significance of a good initial solution for metaheuristics. Nowadays, it is hardly common to see metaheuristics initialized from a random solution, at least in the scheduling field. It is well known that random initial solutions result in slow converging methods. Therefore, one should give careful consideration for the choice of the initialization procedure in order to achieve competitive levels of performance in hard scheduling problems. According to the study of Ruiz and Maroto [3], an adaptation of the well known NEH heuristic of Nawaz et al. [35], referred to as NEHH, resulted in top heuristic performance for hybrid flowshops. As a result of this, we employ the NEHH method for the initialization of the proposed ILS algorithm.

4. Experimental evaluation

In this section, the performance of both, the MDDR heuristic and the proposed ILS, is evaluated. Statistical analyses are also conducted in order to calibrate the different design options and parameters of the ILS metaheuristic. The algorithms are implemented in MATLAB 7. Despite of not being a fast environment, MATLAB permits a fast and seamless coding of the different methods. All experiments are run on a PC with a 2.0 GHz Intel Core 2 Duo processor and 1 GB of RAM memory. No multithreading or multicore programming is employed. Therefore, only one core in the processor is utilized. We use the relative percentage deviation (RPD) over the best solutions known as a performance measure to compare methods and algorithmic parameters:

$$RPD = \frac{Alg_{sol} - Min_{sol}}{Min_{sol}} 100 \quad (1)$$

where Alg_{sol} is the C_{max} obtained by an algorithm for a given problem instance. Obviously, lower values of RPD are preferred. Min_{sol} is the best solution known for a given problem. All instances and best solutions are available from http://soa.iti.es.

The stopping criterion is set to $n^2 \times m \times 1.5$ ms elapsed CPU time. This applies only for the algorithms that need a stopping criterion. Setting a time limit like this allows for a much better statistical analysis. The reason behind is that if a fixed CPU time is used, the effect of the instance size cannot be easily studied. More specifically, if the same number of iterations (or same time) is given to all instances, regardless of instance size, worse results for large instance cannot be attributed only to instance size as they could be the result of insufficient CPU time.

4.1. Parameter tuning

We analyze now the effectiveness of the different design options and parameters of the proposed ILS method. We have only calibrated the ILS-P variant without the FECT technique, as this will be evaluated later. Notice that no calibration is carried out for ILS-ORK and ILS-TRK. However, short experiments (not shown here) indicate that the results of the ILS-P calibration hold for ILS-ORK and ILS-TRK.

We calibrate three parameters: *no_change* (maximum number of non-improving iterations before *perturbation*), *nu_move* (number of perturbed solutions generated in the GL *perturbation* operator) and *d* (number of jobs that are relocated in the GL *perturbation* operator). We employ the well known Design of Experiments (DOE) approach [36]. More specifically, we carry out a full factorial experiment testing the above mentioned factors at the following levels:

- *no_change*: 4 levels (5, 10, 15, and 20);
- *nu_move*: 4 levels (10, 20, 30, and 40);
- *d*: 3 levels (2, 4, and 6).

Short experiments with higher and lower levels for the factors than those tested showed worse performance. This technique of level screening is common in the experimental design literature, see [36] for more details.

All level combinations result in 48 different treatments and hence 48 ILS candidate configurations. We generate a "calibration" set of 60 instances as follows: five instances for each one of the 12 combinations between n and m, where $n = \{20, 50, 80, 120\}$ and $m = \{2, 4, 8\}$. The processing times are generated from a uniform [1,99] distribution as usual in the scheduling literature. The sequence dependent setup times are sampled also from a uniform [1,75] distribution. Finally, the number of parallel machines at each stage (m_i) is sampled from a uniform distribution in the range [1,4]. All 60 instances are solved by the 48 different ILS configurations.

The results are analyzed by means of the analysis of variance (ANOVA) technique. All three main hypotheses (normality, homogeneity of variance–or homocedasticity–and independence of residuals) have been carefully checked by the appropriate techniques. We found no evidence to question the validity of the experiment. The means plot and Tukey intervals with 95% confidence for the levels of *nu_move*, *no_change*, and *d* parameters are shown in Figs. 6–8. Recall that overlapping confidence intervals between any two means indicate that there is no statistically significant difference between them.

As seen in Fig. 6, there is no statistically significant difference between *nu_move* levels of 20, 30, and 40. Based on the average values, *nu_move* of 30 is slightly better than the other values. By applying the same reasoning, we choose the value of 15 for *no_change* (Fig. 7). Fig. 8 shows that $d = 2$ yields statistically better results than 4 or 6. This is an interesting result since a value of 2 for *d* is a very minor *perturbation*. Clearly, larger perturbations result in solutions that are really far off the incumbent one and are not desirable. This result is in concordance with previous observations as, for example, the small perturbation in the ILS of Stützle [27] or the *destruction* parameter in IG algorithms (see [29] for more details). This destruction is similar to the perturbation phase in ILS. According to the calibration experiments of Ruiz and Stutzle [29], a very low destruction value of 4 resulted best.

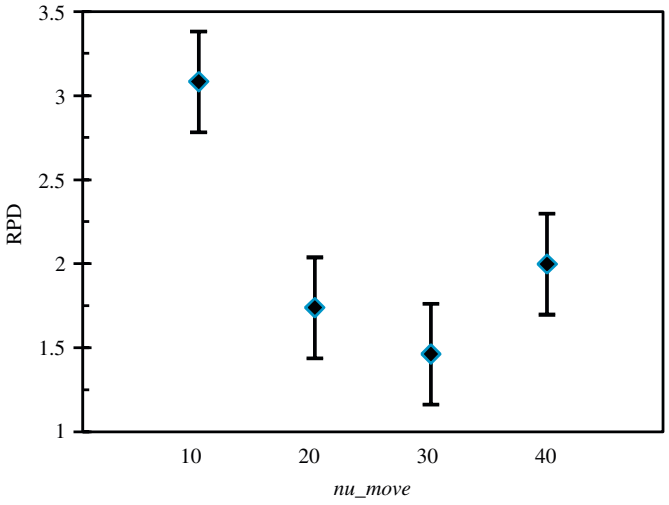

Fig. 6. Means plot and Tukey intervals for ILS *nu_move* parameter.

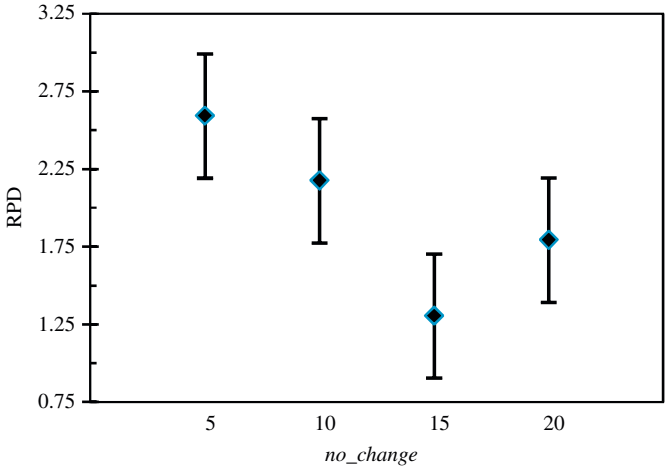

Fig. 7. Means plot and Tukey intervals for ILS *no_change* parameter.

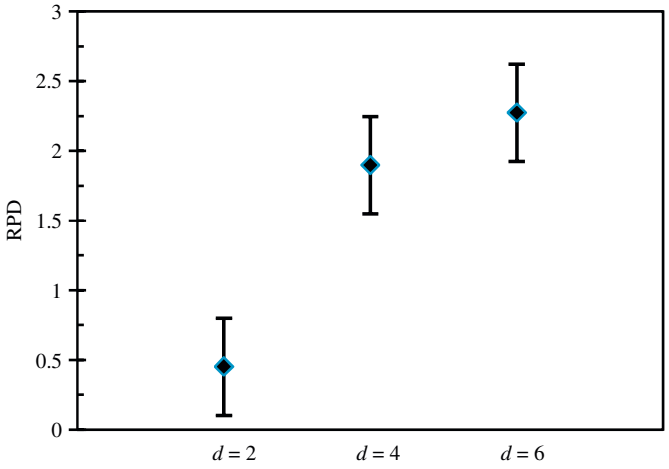

Fig. 8. Means plot and Tukey intervals for ILS *d* parameter.

Table 1
Data employed for the benchmark generation.

Factors	Levels			
n	20	50	80	120
n	2	4	8	
m_i	Constant:	2		
	Variable:	U (1, 4)		
p_{ij}	U (1, 99)			
S_{ijk}	U (1, 25)	U (1, 50)	U (1, 99)	U (1, 125)
S_p	0.10	0.40		

Note that we are calibrating single factors without actually looking into the possibly different performance of these three factors against different values of n and m. In other words, we are disregarding interactions. The reason for this is that fixing different values for the parameters as a function of the size of the instance normally results in over-calibrated algorithms. We are more interested in overall good parameters rather than in a finely tuned method. Finally, the calibration leads to the following values of the three parameters: $nu\_move = 30$, $no\_change = 15$, and $d = 2$.

4.2. Data generation

Now that we have calibrated the parameters of the proposed ILS, we need a larger data set for the computational evaluation of the different ILS encoding alternatives and existing algorithms from the literature. We generate a larger set of instances, different from the ones used for calibration, so to avoid bias in the results.

We generate test instances with the following combinations of n and m, where $n = \{20, 50, 80, 150\}$ and $m = \{2, 4, 8\}$. The processing times are generated from a uniform [1,99] distribution. In this case, we also want to study the effect of the magnitude of setup times. Following Ruiz and Stützle [30], we generate the setup times according to four distributions, [1,25], [1,50], [1,99], and [1,125]. Notice that this corresponds to a ratio between setup and processing times of 25%, 50%, 100%, and 125%, respectively. We also generate groups of instances with two parallel machines per stage and groups where the number of parallel machines at each stage is sampled from a uniform distribution in the range [1,4]. The probability of skipping a stage for each job (S_p) is set at 0.10 and 0.40. Therefore, there are 192 combinations of n, m, m_i, S_{ijk}, and S_p. For each combination we generate five different instances. Therefore, the total number of test instances is 960. All instances and best solutions are available from http://soa.iti.es. A summary of the instances is given in Table 1. Note that other instances have been proposed for similar problems. For example, Ruiz and Maroto [3] proposed a large benchmark of 1320 instances. However, they do not consider stage skipping and their data sets are much heavier since they consider unrelated parallel machines, machine eligibility at each stage and also machine-dependent set up times.

4.3. Analysis of the behavior of ILS with different encoding schemes

We now compare the performance of the proposed ILS algorithm with the three encoding schemes analyzed in Section 3.3. Namely, ILS-P, ILS-ORK, and ILS-TRK. Recall that these schemes have been used for the same SDST/HFFS problem or for similar settings in [3,16,15], respectively. We evaluate all three algorithm variants against all 960 instances introduced in the previous section. The results of the experiments, averaged for each n and m combination (80 data per average) are presented in Table 2. As can be seen, ILS-P and ILS-ORK algorithms provide better results than ILS-TRK. ILS-P results are slightly better than those of ILS-ORK in terms of the average RPD. In order to verify the statistical validity of the results presented in Table 2, and to determine which encoding scheme is best, we perform an ANOVA where we consider the different encodings as a factor and RPD as the response variable. Fig. 9 shows corresponding means plot.

Table 2
$\overline{RPD}$ for ILS-P, ILS-TRK, and ILS-ORK.

Instance	ILS-P	ILS-TRK	ILS-ORK
20×2	11.53	**5.30**	14.30
20×4	12.36	**8.49**	14.92
20×8	**4.61**	16.17	6.27
50×2	11.17	**6.06**	12.45
50×4	**4.26**	12.34	5.11
50×8	**1.02**	14.76	1.21
80×2	7.48	**3.60**	8.21
80×4	**2.96**	15.27	4.34
80×8	**0.87**	16.28	0.95
120×2	12.48	**2.83**	11.35
120×4	**1.71**	13.91	2.10
120×8	**0.68**	18.79	0.83
Average	**5.93**	11.15	6.79

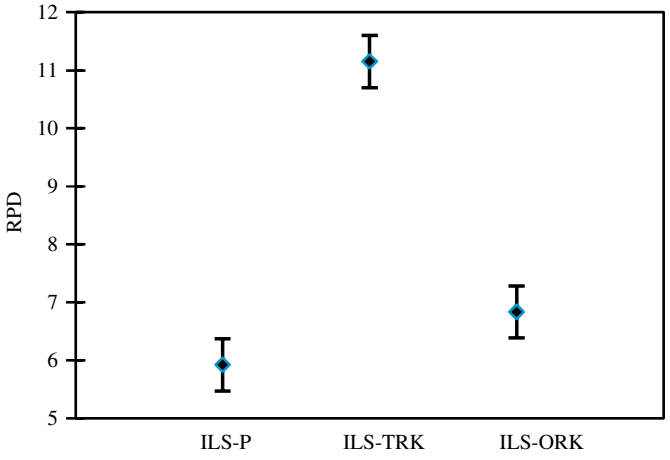

Fig. 9. Means plot and Tukey intervals for the three encoding schemes in the ILS algorithm.

The results indicate that there is no overall statistically significant difference between the performance of ILS-P and ILS-ORK, albeit by a very small margin. However, both statistically outperform ILS-TRK. In order to analyze the possible impact of n and m over the different encoding schemes we apply a two-way ANOVA. Figs. 10 and 11 report the corresponding means plots and Tukey intervals (at the 95% confidence level) for the interactions between the encoding schemes and n and m, respectively.

As seen in Fig. 10, there is a clear trend where ILS-TRK deteriorates with larger n values and specially with more than two stages. The reason is clear. For larger instances, an exact representation of a solution entails a much larger search space of solutions and the ILS shows worse performance in this case. For small instances, and specially when $m = 2$, ILS-TRK is statistically better than the other encodings, and by a wide margin. However, for a larger number of stages, the savings obtained in smaller instances do not compensate. We can conclude that the more verbose encoding scheme (direct: ILS-TRK) outperforms the other encoding schemes for small instance sizes, while job-based representation (indirect) generally provides better results as observed in Table 2.

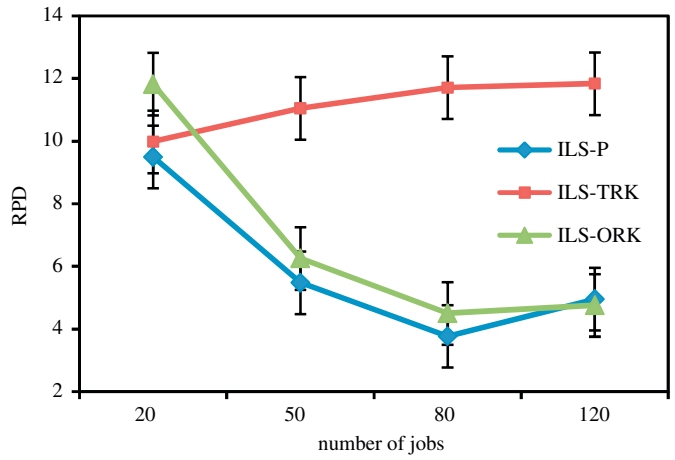

Fig. 10. Means plot and Tukey intervals for the interaction between encoding schemes of the ILS and number of jobs.

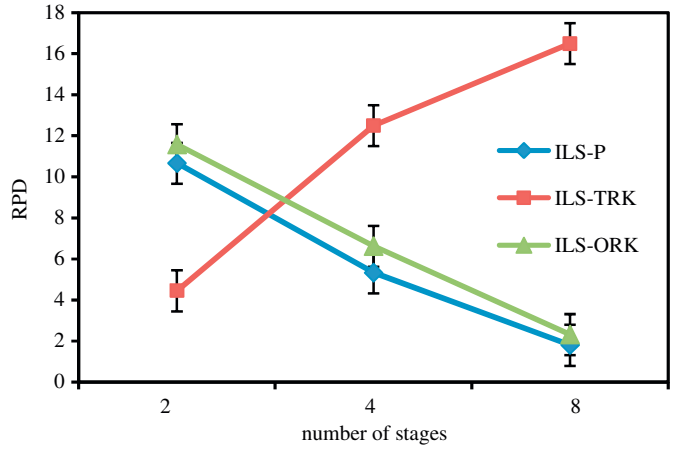

Fig. 11. Means plot and Tukey intervals for the interaction between encoding schemes of the ILS and number of stages.

Table 3
$\overline{RPD}$ for ILS-P and ILS-P_{FECT}.

Instance	ILS-P	ILS-P_{FECT}
20×2	2.14	0.40
20×4	2.25	0.73
20×8	2.33	0.48
50×2	3.02	0.76
50×4	2.13	0.62
50×8	2.07	0.73
80×2	2.35	0.51
80×4	1.94	0.44
80×8	2.27	0.52
120×2	2.42	0.49
120×4	2.09	0.70
120×8	2.23	0.54
Average	2.27	0.58

4.4. Evaluating the effectiveness of FECT technique

We now evaluate the effectiveness of the proposed FECT technique by comparing the performance of ILS-P without and with this technique (ILS-P_{FECT}). Table 3 presents the results of the experiment. This comparison strongly supports the effectiveness of the FECT technique. Again, we carry out an ANOVA to statistically check the significance of the observed differences in performance. Fig. 12 shows the means plot.

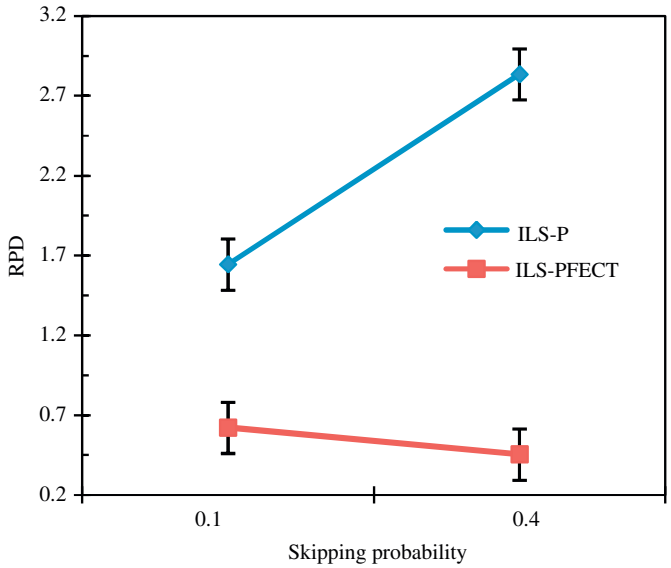

Fig. 12. Means plot and LSD intervals for the interaction between algorithm type and skipping probability.

As can be seen, the FECT technique improves results in a significant way. Furthermore, when the probability of skipping stages is high, FECT is even better. This supports our previous statement that redundancy in the JBR encoding (ILS-P) needs to be carefully accounted for. As a result of this experiment, the FECT technique is employed for ILS-P. In the following, ILS-P is used to actually denote ILS-P_{FECT}.

4.5. Experimental results

We proceed now with the comparisons of the proposed ILS-P (with the FECT technique), and the proposed MDDR heuristic against other existing algorithms from the literature. Among the existing methods, we have fully re-coded in MATLAB the following methods:

- "SPTCH", "FTMIH" and the "$g/2, g/2$ Johnson's rule" heuristics from Kurz and Askin [14].
- Random keys genetic algorithm "RKGA" proposed by Kurz and Askin [15].
- "NEHH" heuristic and the genetic algorithm "GA_R" from Ruiz and Maroto [3]. Note that these methods were proposed for a different problem and some adaptations were needed.
- Immune algorithm "IA_Z" from Zandieh et al. [16].

All these algorithms have shown high performance in the original papers in which they were proposed. For example, GA_R by Ruiz and Maroto [3] was compared against four other high performing genetic algorithms, against a simulated annealing, a tabu search and two ant colony optimization methods and showed clearly better results. Summing up, the algorithms above showed the best results in their respective studies.

We first analyze the makespan results and then analyze the effect of some instance factors, such as problem size and magnitude of setup times. All algorithms have been run on the instances of Section 4.2 and the results are now discussed. The stopping criterion for the metaheuristics (i.e., ILS-P, RKGA, IA_Z, and GA_R) is again $n^2 \times m \times 1.5$ ms elapsed CPU time. Note that all algorithms have been re-coded

Table 4
The $\overline{RPD}$ for the tested algorithms.

Instance	SPTCH	FTIMH	John.	NEHH	MDDR	ILS-P	RKGA	IA_Z	GA_R
20×2	39.38	41.82	29.29	8.03	14.43	**1.39**	6.93	4.39	2.61
20×4	29.96	37.57	21.79	10.16	15.15	**1.27**	3.89	4.25	2.94
20×8	24.35	29.95	18.83	10.21	14.44	**1.49**	3.09	4.35	3.50
50×2	34.18	38.71	30.30	7.20	10.24	**1.26**	9.01	4.82	2.51
50×4	26.60	38.60	26.60	8.93	10.53	**1.85**	5.85	4.24	4.03
50×8	24.68	29.06	19.30	9.49	8.97	**1.75**	3.79	4.28	3.20
80×2	35.65	42.22	31.66	6.13	5.84	**2.06**	8.28	7.19	2.15
80×4	27.59	36.15	25.85	9.37	7.49	**2.92**	7.39	6.70	4.64
80×8	28.07	32.02	23.92	12.20	8.18	5.46	8.53	5.95	**4.51**
120×2	36.03	40.12	34.27	6.66	5.19	**3.38**	10.58	7.06	3.42
120×4	33.92	40.24	30.79	11.25	**5.93**	6.74	12.04	8.96	5.97
120×8	28.74	35.27	25.97	14.21	6.80	9.25	11.69	7.82	**5.19**
Average	30.76	36.81	26.55	9.49	9.43	**3.24**	7.59	5.83	3.72

and are run on the same instances, on the same computer and with the same elapsed CPU time stopping criterion. Therefore, results are fully comparable.

It is necessary to address that we first implemented the algorithms from the literature exactly as they were originally proposed in their corresponding papers. However, the usage of the FECT technique in the ILS-P resulted in large differences in favor of our proposed ILS method. For the sake of a fair comparison, we decided to include the FECT technique into all re-implemented metaheuristics from the literature. It can be concluded that FECT is a powerful technique for problems with stage skipping.

4.5.1. Analysis of makespan

The results of the computational comparison, averaged for each combination of n and m (80 data per average) are presented in Table 4. As it can be seen, ILS-P and GA_R provide the best results among the tested algorithms with average RPD values of 3.24% and 3.72%, respectively. Surprisingly, MDDR obtains the lowest average RPD in case of 120×4, although it is a fast dispatching rule. The two worst performing algorithms are FTMIH and SPTCH with RPD values of 36.81% and 30.76%, respectively.

We carry out an ANOVA on the results where the type of algorithm is the single controlled factor. We exclude from this analysis the heuristics SPTCH, FTMIH, and $(g/2, g/2)$ Johnson's rule due to their poor performance. The results indicate that there are statistically significant differences between various algorithms with a p-value very close to zero in mostly all cases. Fig. 13 shows the means plot with Tukey intervals (at the 95% confidence level) for the different algorithms.

As can be seen, ILS-P and GA_R provide statistically far better results among those algorithms tested. Interestingly enough, ILS-P is statistically better, on average, than GA_R. MDDR and NEHH are statistically similar. IA_Z outperforms RKGA, which confirms previous findings of Zandieh et al. [16].

4.5.2. Analysis of instance factors

The purpose of this subsection is to assess the relation between the performance of the algorithms and the factors affecting the characteristics of the instances, like n, m, m_i, S_{ijk}, and S_p. We first start with the influence of n over the different algorithms. The best analysis option is a two-way ANOVA whose result, in form of a means interaction plot, is given in Fig. 14.

The first interesting conclusion is that the proposed MDDR heuristic is very sensitive to the number of jobs. Surprisingly, with more jobs, MDDR performs better. Actually, for more than 50 jobs, MDDR clearly and statistically outperforms the much more complex and costlier NEHH. Furthermore, for $n = 120$, MDDR is competitive

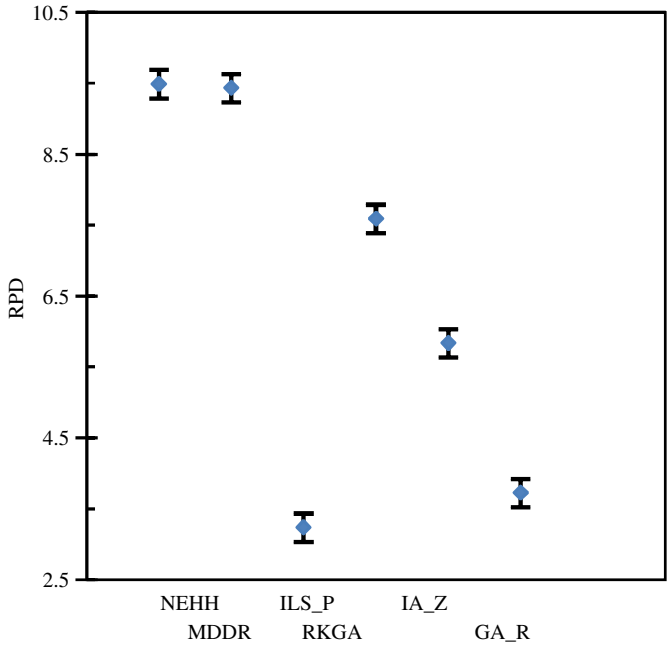

Fig. 13. Means plot and Tukey intervals for the tested algorithms.

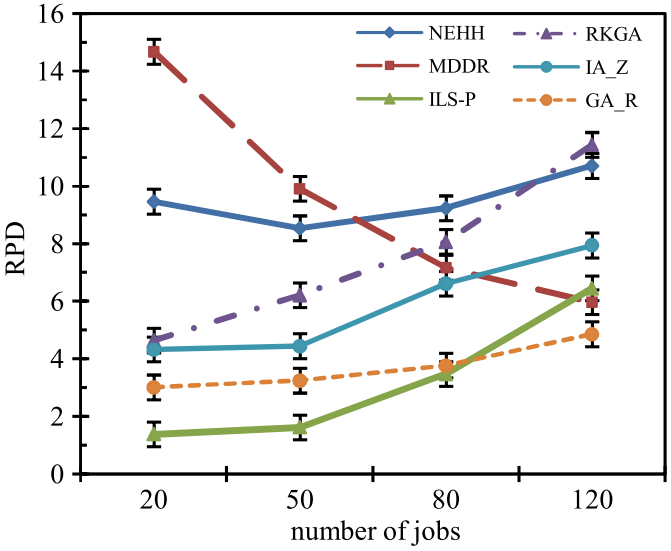

Fig. 14. Means plot and Tukey intervals for the interaction between algorithms and the number of jobs.

with the state-of-the-art methods. Probably, for more than 120 jobs, MDDR could improve the results of the state-of-the-art. In any case, the SDST/HFFS problem is a very complex realistic scheduling setting and it is unlikely to find instances with more than 120 jobs in practice. Let us remind that scheduling is solved in the short term.

Notice that although ILS-P has a lower overall RPD than GA_R, and that this difference is statistically significant, for very large instances of 120 jobs, GA_R shows superior performance. Our hypothesis for this result is that the proposed local search, albeit fast for most instances, turns out slow for the largest instances. As a matter of fact, GA_R does not include local search. This is interesting, because this genetic algorithm is the evolution of other GAs applied by the same authors to simpler problems like the SDST flowshop in [37], and regular flowshops in [38]. These previous GAs incorporate local

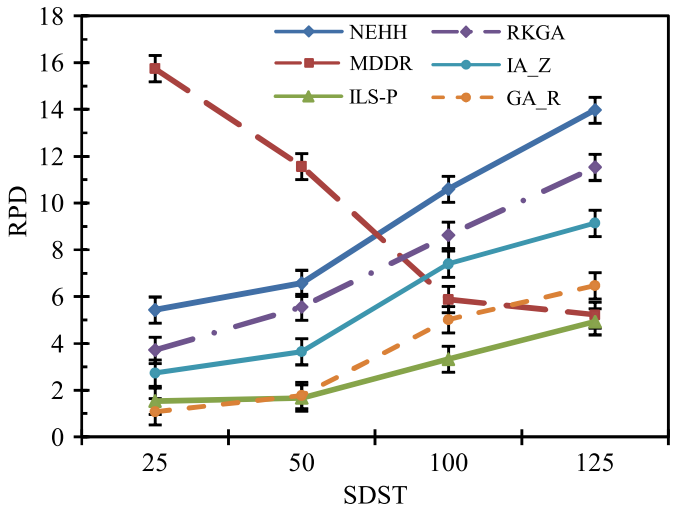

Fig. 15. Means plot and Tukey intervals for the interaction between algorithms and the magnitude of setup times.

search. For hybrid flowshops, the local search turns out slow for large instances and therefore the GA approach performs best. In any case, it has to be reminded that the original GA_R was proposed without the FECT technique and here it has been tested with it. Otherwise, ILS-P results in much better performance across all instances, including the largest ones.

It is also interesting to check the influence of the magnitude of setup times over the different algorithms. Fig. 15 shows the means plot resulting from such analysis.

Once again, there is a marked interaction between MDDR, the magnitude of the setup times, and the remaining algorithms. With increasing length of setup times, the performance of MDDR improves comparatively. Furthermore, for SDST = 125%, MDDR is statistically better than GA_R. As regards ILS-P, its performance also improves (comparatively speaking) as the magnitude of setup times increases; for SDST = 100% and 125%, it is statistically the best algorithm from the comparison. This result confirms our previous hypothesis that SDST have to be adequately considered in the solution methods.

We checked all the other instance characteristics including the number of stages, different levels of skipping probability and number of parallel machines per stage. We found no remarkable effects from these characteristics on the performance of the algorithms. To summarize, MDDR and ILS-P are very effective in environments where setup times play an important role. MDDR also turns into a competitive algorithm for instances with a large number of jobs.

4.6. Analysis of CPU time

It is also necessary to analyze the efficiency of the different algorithms. We measured the time (in seconds) that a given method needs in order to provide a solution. As mentioned earlier, the stopping criterion for all the metaheuristics (i.e., ILS-P, RKGA, IA_Z, and GA_R) is set to $n^2 \times m \times 1.5$ ms elapsed CPU time and therefore there are no differences here. However, it is interesting to compare the CPU times employed by the heuristic methods. Table 5 summarizes the CPU times of the algorithms, grouped by n and m. We omit the CPU times of SPTCH, FTMIH, and $(g/2, g/2)$ Johnson rule heuristics since they all needed less than 0.01 s on average for all instance sizes.

It is of interest to discuss the CPU times of MDDR and NEHH. The first conclusion that can be drawn is that MDDR is much faster than NEHH. Furthermore, and as Fig. 16 shows, this difference becomes more noticeable as the number of jobs increases. For the largest

Table 5
Average CPU times for the tested algorithms, measured in seconds.

Instance	NEHH	MDDR	Metaheuristics
20×2	0.05	< 0.01	1.20
20×4	0.09	0.01	2.40
20×8	0.16	0.02	4.80
50×2	0.34	0.04	7.50
50×4	0.66	0.07	15.00
50×8	1.29	0.12	30.00
80×2	1.09	0.10	19.20
80×4	2.24	0.19	38.40
80×8	4.43	0.36	76.80
120×2	3.16	0.28	43.20
120×4	7.03	0.51	86.40
120×8	14.59	1.03	172.8
Average	2.91	0.23	40.85

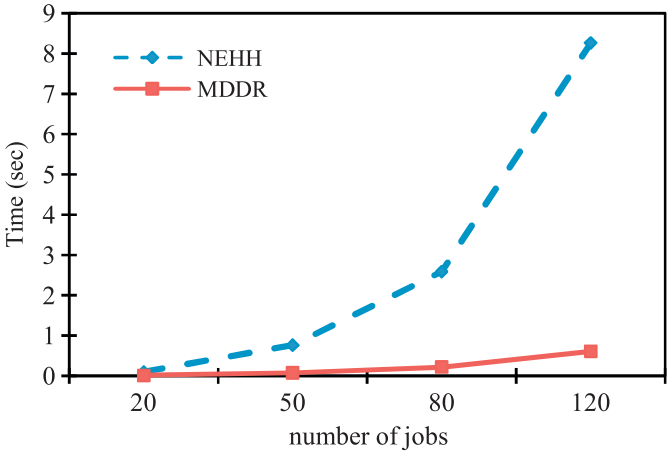

Fig. 16. CPU times of NEHH and MDDR heuristics.

size of $n = 120$ and $m = 8$, MDDR is 14 times faster than HEHH. Considering also that for those largest instances, MDDR provides much better results in terms of average RPD, we can conclude that MDDR is a very effective heuristic for large SDST/HFFS problems.

5. Conclusions

In this paper we have proposed two high performing algorithms for hybrid flexible flowshops with sequence-dependent setup times under makespan minimization criterion. The first algorithm is a very fast dynamic dispatching rule heuristic. The second method proposed is an iterated local search metaheuristic. Additionally, this paper presents effective techniques in order to deal with some specific characteristics of the considered problem, like stage skipping or setup times. These techniques allow overcoming some drawbacks identified in the literature and in previous existing algorithms. Furthermore, we have analyzed and compared the three most common encoding schemes for hybrid flowshop problems in our iterated local search method. The results indicate that the choice of the encoding scheme strongly depends on the problem size and characteristics. For example, for small instances, a verbose encoding scheme (direct encoding) performs better, while in medium and large instances, a job-based representation is recommended. We have compared the proposed algorithms against seven other algorithms from the literature. Together with the comparison, we have also assessed the effect of the instance characteristics over the algorithms.

The results support our initial hypothesis that stage skipping and setup times need to be specifically considered in the solution methods if high performance is desired. Among our proposed algorithms, the MDDR dynamic dispatching rule is both faster and more effective than all other existing heuristics for medium and large instances. This is also true for all instances with large setups to processing times ratios. Similarly, our proposed iterated local search method provides state-of-the-art results for small and medium instances or for all instances with large setup times.

References

[1] Pinedo M. Scheduling: theory, algorithms, and systems. 2nd ed, Upper Saddle, NJ: Prentice-Hall; 2002.

[2] Dudek RA, Panwalkar SS, Smith ML. The lessons of flowshop scheduling research. Operations Research 1992;40(1):7–13.

[3] Ruiz R, Maroto C. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. European Journal of Operational Research 2006;169(3):781–800.

[4] Allahverdi A, Gupta JND, Aldowaisan T. A review of scheduling research involving setup considerations. Omega—International Journal of Management Science 1999;27(2):219–39.

[5] Allahverdi A, Ng CT, Cheng TCE, Kovalyov MY. A survey of scheduling problems with setup times or costs. European Journal of Operational Research 2008;187(3):985–1032.

[6] Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 1979;5:287–326.

[7] Haouari M, Ladhari T. A branch-and-bound-based local search method for the flow shop problem. Journal of the Operational Research Society 2003;54(10):1076–84.

[8] Ríos-Mercado RZ, Bard JF. A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. IIE Transactions 1999;31(8):721–31.

[9] Ríos-Mercado RZ, Bard JF. The flow shop scheduling polyhedron with setup times. Journal of Combinatorial Optimization 2003;7(3):291–318.

[10] Allahverdi A, Soroush HM. The significance of reducing setup times/setup costs. European Journal of Operational Research 2008;187(3):978–84.

[11] Vignier A, Billaut J-C, Proust C. Les problèmes d'ordonnancement de type flow-shop hybride: État de l'art. RAIRO Recherche Operationnelle 1999;33(2):117–83.

[12] Ruiz R, Sivrikaya Serifoglu F, Urlings T. Modeling realistic hybrid flexible flowshop scheduling problems. Computers & Operations Research 2008;35(4):1151–75.

[13] Gupta JND. Two-stage, hybrid flowshop scheduling problem. Journal of the Operational Research Society 1988;39(4):359–64.

[14] Kurz ME, Askin RG. Comparing scheduling rules for flexible flow lines. International Journal of Production Economics 2003;85(3):371–88.

[15] Kurz ME, Askin RG. Scheduling flexible flow lines with sequence-dependent setup times. European Journal of Operational Research 2004;159(1):66–82.

[16] Zandieh M, Ghomi SMTF, Husseini SMM. An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. Applied Mathematics and Computation 2006;180(1):111–27.

[17] Johnson SM. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1954;1(1):61–8.

[18] Linn R, Zhang W. Hybrid flow shop scheduling: a survey. Computers & Industrial Engineering 1999;37(1–2):57–61.

[19] Kis T, Pesch E. A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. European Journal of Operational Research 2005;164(3):592–608.

[20] Haouari M, Hidri L. On the hybrid flowshop scheduling problem. International Journal of Production Economics 2008;113(1):495–7.

[21] Jin ZH, Yang Z, Ito T. Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem. International Journal of Production Economics 2006;100(2):322–34.

[22] Quadt D, Kuhn H. A taxonomy of flexible flow line scheduling procedures. European Journal of Operational Research 2007;178(3):686–98.

[23] Jungwattanakit J, Reodecha M, Chaovalitwongse P, Werner F. Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. International Journal of Advanced Manufacturing Technology 2008;37(3–4):354–70.

[24] Jungwattanakit J, Reodecha M, Chaovalitwongse P, Werner F. A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. Computers & Operations Research 2009;36:358–78.

[25] Naderi B, Zandieh M, Roshanaei V. Scheduling hybrid flowshops with sequence dependent setup times to minimize makespan and maximum tardiness. International Journal of Advanced Manufacturing Technology 2009;41:1186–98.

[26] Hoos HH, Stützle T. Stochastic local search: foundations and applications. San Francisco, CA: Morgan Kaufmann; 2005.

[27] Stützle T. Applying iterated local search to the permutation flow shop problem. Technical Report AIDA-98-04, FG Intellektik, TU Darmstadt; 1998.

[28] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 2005;165(2):479–94.

[29] Ruiz R, Stutzle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 2007;177(3):2033–49.

[30] Ruiz R, Stützle T. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European Journal of Operational Research 2008;187(3):1143–59.

[31] Lourenço HR, Martin OC, Stützle T. Iterated local search. In: Glover F, Kochenberger GA, editors. Handbook of metaheuristics. Boston: Kluwer Academic Publishers; 2003. p. 321–53.

[32] Essafi I, Mati Y, Dauzère-Pérès S. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. Computers & Operations Research 2008;35(8):2599–616.

[33] Bean JC. Genetic algorithms and random keys for sequencing and optimization. INFORMS Journal on Computing 1994;6(2):154–60.

[34] Norman BA, Bean JC. A genetic algorithm methodology for complex scheduling problems. Naval Research Logistics 1999;46(2):199–211.

[35] Nawaz M, Enscore Jr. EE, Ham I. A heuristic algorithm for the *m*-machine, *n*-job flowshop sequencing problem. Omega—International Journal of Management Science 1983;11(1):91–5.

[36] Montgomery DC. Design and analysis of experiments. 6th ed., Hoboken, NJ: Wiley; 2005.

[37] Ruiz R, Maroto C, Alcaraz J. Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. European Journal of Operational Research 2005;165(1):34–54.

[38] Ruiz R, Maroto C, Alcaraz J. Two new robust genetic algorithms for the flowshop scheduling problem. Omega—International Journal of Management Science 2006;34(5):461–76.