Available online at www.sciencedirect.com

ScienceDirect

Computers & Operations Research 35 (2008) 1350–1373

computers & operations research

www.elsevier.com/locate/cor

Minimising total tardiness in the *m*-machine flowshop problem: A review and evaluation of heuristics and metaheuristics

Eva Vallada*, Rubén Ruiz, Gerardo Minella

Dpto. de Estadística e Investigación Operativa Aplicadas y Calidad, Universidad Politécnica de Valencia, Edificio I-3, Camino de Vera S/N, 46021 Valencia, Spain

Available online 10 October 2006

Abstract

In this work, a review and comprehensive evaluation of heuristics and metaheuristics for the *m*-machine flowshop scheduling problem with the objective of minimising total tardiness is presented. Published reviews about this objective usually deal with a single machine or parallel machines and no recent methods are compared. Moreover, the existing reviews do not use the same benchmark of instances and the results are difficult to reproduce and generalise. We have implemented a total of 40 different heuristics and metaheuristics and we have analysed their performance under the same benchmark of instances in order to make a global and fair comparison. In this comparison, we study from the classical priority rules to the most recent tabu search, simulated annealing and genetic algorithms. In the evaluations we use the experimental design approach and careful statistical analyses to validate the effectiveness of the different methods tested. The results allow us to clearly identify the state-of-the-art methods.
© 2006 Elsevier Ltd. All rights reserved.

Keywords: Flowshop; Scheduling; Tardiness; Heuristics; Metaheuristics

1. Introduction

The flowshop scheduling problem is one of the most thoroughly studied problems in combinatorial optimisation. Its simple definition conceals a very difficult problem which has attracted the attention of numerous researchers in the past decades. In a flowshop there is a set $N = \{1, \dots, n\}$ of n jobs that have to be processed on a set $M = \{1, \dots, m\}$ of m machines. All jobs follow the same route in the machines and $p_{ij} \geqslant 0$ denotes the fixed processing time of job j, $j \in N$, on machine i, $i \in M$. Among the existing regular criteria, the minimisation of the maximum completion time or makespan ($C_{\max}$) is widely used. The simplification of considering only permutation schedules (the processing order of the jobs is the same for all machines) is very common and the resulting problem is called the permutation flowshop problem (PFSP) and denoted as $F/prmu/C_{\max}$ [1]. This problem was shown to be $\mathcal{NP}$-complete by [2] for $m \geqslant 3$. We can find in [3] an extensive comparison and evaluation of different methods for this objective. Recently, research has also focused on the development of algorithms with the objective of minimising total tardiness of jobs owing its importance to real-life considerations. In industrial plants, criteria based on due dates for delivery have become more important than those based on makespan. Let d_j, $j \in N$, be the *due date* for the job j, the job's tardiness is defined as $T_j = \max\{C_j - d_j, 0\}$, where C_j is the completion time of job j. The objective is to minimise the tardiness over all jobs or

* Corresponding author. Tel.: +34 96 387 70 07x74911; fax: +34 96 387 74 99.
E-mail addresses: evallada@eio.upv.es (E. Vallada), rruiz@eio.upv.es (R. Ruiz), mgerar@eio.upv.es (G. Minella).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.08.016

total tardiness: $\sum_{j=1}^{n} T_j$. This problem is denoted as $F/prmu/\sum T_j$ [1] and it is known to be $\mathcal{NP}$-hard in the ordinary sense even when there is only one machine and $\mathcal{NP}$-hard in the strong sense for $m \geqslant 2$. For more details see [4].

In the literature of scheduling to minimise total tardiness of jobs we can find a few review papers, most of them dealing only with one machine. In [5] the authors presented a review focused on one machine and branch and bound methods where the scheduling problems are classified according to different objectives involving due dates. In [6], a review and evaluation of heuristics for several problems is presented. The authors survey different heuristics in the single machine, parallel machine, flowshop and jobshop settings. Several methods are proposed and evaluated for both the single machine and the parallel machine tardiness problems. In [7], a comparison of heuristics for the single machine total weighted tardiness problem is carried out. Finally, [8] reviews research on the total tardiness and total weighted tardiness problems in the single machine environment, where some extensions of these problems for multi-machine environments are also presented.

As we can see, all these reviews and evaluations are mainly focused on the single machine case and no comparison deals with the most recent heuristics and metaheuristics available for the multi-machine flowshop scheduling problem. Moreover, it is very difficult to compare the methods proposed in the literature because the evaluations are partial and the benchmarks used are not standard, therefore the results are many times difficult to reproduce.

The objective of this paper is to give an up-to-date review and evaluation of many existing heuristics and metaheuristics for the m-machine PFSP to minimise the total tardiness of the jobs. We evaluate 40 methods, from the classical despatching rules to the most recent heuristic and metaheuristic methods such as tabu search, genetic algorithms and differential evolution algorithms. Furthermore, we propose a benchmark to evaluate all the methods under a common data set, so that the results can be generalised.

The remainder of this paper is organised as follows: In Section 2 we review some exact methods for the PFSP with the objective to minimise the total tardiness of the jobs. In Section 3, most well-known heuristics for the same problem are surveyed. Section 4 deals with the metaheuristics and a comparison of all methods can be seen in Section 5. Finally, some conclusions are given in Section 6.

2. Exact methods

Due to the complexity of flowshop scheduling problems, using exact methods to solve them is impracticable for instances of more than a few jobs and/or machines. In [9], a branch and bound algorithm is proposed for the two-machine flowshop problem. The authors proposed a theorem based on the interchange of jobs which should appear consecutively if they satisfy some specific rules. This theorem is used to prune some branches in the search. The authors presented a comparison against the earliest due date (EDD), shortest processing time (SPT) and minimum slack (SLACK) despatching rules. The experiments were carried out using a set of 640 problems where the processing times for both machines are randomly generated from a uniform distribution over the values 1 and 10. The due dates are also randomly generated from a uniform distribution between $P(1 - T - R/2)$ and $P(1 - T + R/2)$ following the procedure presented in [10] where T and R are two parameters called *tardiness factor* and *due date range*. The P is commonly a lower bound on the makespan but in this case is the sum of the processing times in machine two plus the smallest processing time in machine one. Several problems were proposed where $T = \{0.25, 0.5, 0.75, 1\}$, $R = \{0.25, 0.5, 0.75, 1\}$ and $n = \{6, 8, 10, 12\}$. Therefore, 64 combinations are generated each of which is repeated 10 times. The results show that the SPT rule performs very well for large T values and the number of nodes processed by the branch and bound to find an optimal solution tends to increase with increasing values of T and R.

Another branch and bound algorithm for the two-machine case is that of [11]. The authors proposed a lower bound based on the sum of two lower bounds computed from the set of jobs in the partial sequence and the set of jobs not included in it, respectively. Dominance rules to prune sequences are also presented. A total of 240 problems were randomly generated to test the performance of the method. Processing times were uniformly distributed between 1 and 30 and due dates of the jobs were computed using the method previously commented where P in this case is the sum of the processing times of all operations divided by two. Several combinations of the parameters T, R and the number of jobs were considered where $T = \{0.1, 0.2, 0.3, 0.4, 0.5\}$, $R = \{0.8, 1, 1.2, 1.4, 1.6, 1.8\}$ and $n = \{10, 11, 12, 13, 14, 15\}$. The author compares the proposed branch and bound using the lower bounds and dominance rules presented in the paper against the branch and bound proposed in [9] and a similar branch and bound but adding the dominance rule presented in [9]. The results show that the branch and bound of [9] was outperformed by the other two proposed algorithms which showed a very similar performance.

In [12], a branch and bound is also presented for the same problem. The authors proposed dominance properties to reduce the size of the problem and a lower bound. The performance of the proposed algorithm is compared with the method proposed in [11]. A total of 600 problems were randomly generated such that the processing times were uniformly distributed over the range 1–10. Due dates were computed from another uniform distribution as aforementioned where P is the sum of the processing times of all the jobs on the second machine plus the minimum processing time among all the jobs on first machine. The methods were tested over several problem sizes and combinations of the parameters T and R and n where $T=\{0.25, 0.5, 0.75\}$, $R=\{0.25, 0.5, 0.75, 1\}$ and $n=\{10, 12, 14, 16, 18\}$. A total of 60 sets of problems were considered and for each one 10 replicates were generated randomly. The results showed that the branch and bound proposed outperformed that presented in [11] in terms of average time and number of problems solved. The proposed method was able to solve all the problems up to 16 jobs and those of 18 jobs up to $T=0.5$.

In [13] the authors proposed another branch and bound algorithm also for the two-machine case. Several procedures to establish precedence constraints between jobs were presented. The initial solution is given by the EDD rule. The computational experiments were carried out using 600 randomly generated problems where the processing times are uniformly distributed between 1 and 10. The due dates are also uniformly distributed as in the previous studies where P is the sum of the processing times in machine two plus the smallest processing time in machine one. Different problem sizes were tested according to the parameters T, R and n where $T=\{0.25, 0.5, 0.75\}$, $R=\{0.25, 0.5, 0.75, 1\}$ and $n=\{16, 18, 20, 22, 24\}$. Therefore, 60 combinations are generated, each one is repeated 10 times. The proposed branch and bound solved optimally all the replicates up to 24 jobs with $T=0.5$ and $R=0.5$. A problem was considered not solved if an algorithm needed more than 3600 s of elapsed time to obtain the optimal solution. The results are compared against the branch and bound presented in [12] which was able to solve optimally all the 10 replicates up to 20 jobs where $T=0.25$ and $R=0.75$. Therefore, the proposed method outperformed that proposed in [12] in terms of average CPU time and number of problems solved.

In a more recent work [14], three branch and bound algorithms are developed also for the two-machine case. The authors improved the lower bound and the dominance conditions presented in [13] and a new dominance rule was also proposed. Three branch and bound algorithms were presented combining the different dominance conditions and lower bounds. A depth first strategy is used and the algorithms were tested on randomly generated problems where the processing time of the jobs were computed using a uniform distribution over the integers 1 and 10. The due dates are generated in the same way explained above. In total, 12 sets were generated where $T=\{0.5, 0.75\}$, $R=\{0.5, 1\}$ and $n=\{16, 18, 20\}$ and each one consists of 10 problems. The CPU time limit was set to 20 min, if a method was not able to solve the problem within this time limit, the algorithm was terminated and the best objective value found was returned. The comparison is carried out between the three algorithms and the results showed that two of the proposed methods solved 80 problems where $n=16$ and 18 while the third method solved only 78 of the 80 problems.

Only a few articles deal with optimal algorithms for multi-machine flowshop problems. In [15], a branch and bound is proposed where some properties to compute a lower bound for the total tardiness problem are presented. The depth first rule is used and the lower bound is computed for each node following the properties presented. A procedure to check the existence of dominated sequences is applied in the root node to reduce the size of the problem. The algorithm was tested using 480 randomly generated instances where the processing time of the jobs are uniformly distributed between 1 and 30 and the due dates were randomly generated as usual. Several combinations of number of machines m and number of jobs n are considered where $n=\{10, 11, 12\}$ and $m=\{4, 6, 8, 10\}$. Additionally, the following special pairs of (n, m) were tested: (13,4), (13,6), (13,8), (14,4), (14,6), (14,8), (15,4), (16,4), (17,4), (18,4), (19,4) and (20,4). Therefore, there are 24 sets in accordance with n and m values and each consists of 20 problems with different combinations of T and R values where $T=\{0.1, 0.2, 0.3, 0.4, 0.5\}$ and $R=\{0.8, 1, 1.2, 1.4, 1.6, 1.8\}$. The results show that the proposed algorithm solved all 20 instances of each group until $n=13$ jobs and $m=8$ machines. It was able to solve also all the instances of the group $n=14$ and $m=4$ machines. Regarding the special sets of instances, the branch and bound proposed was not able to solve all the 20 problems within the CPU limit time set to 3600 s. In order to test the results, the algorithm is compared against the branch and bound for the two-machine case presented in [9] since the authors did not find any other work to solve the m-machine problem optimally. For this case, 120 problems were generated with $n=\{10, 11, 12, 13, 14, 15\}$ jobs and 20 instances for each n value. The proposed branch and bound was able to solve optimally all the problems generated while the branch and bound proposed in [9] did not solve some of the 20 problems where the number of jobs is greater or equal to 13.

In a very recent work [16], a branch and bound algorithm is proposed for the total tardiness m-machine flowshop problem. The authors proposed a machine-based lower bound and a dominance rule for pruning nodes. The depth first rule and a backtracking strategy are used. Computational experiments were carried out using problems randomly generated. The processing times were distributed uniformly between a_{ik} and b_{ik}, where a_{ik} and b_{ik} are dependent on a trend and a correlation factor. Twelve combinations of n and m values are tested where $n = \{10, 15, 20\}$ and $m = \{2, 4, 6, 8\}$. The authors proposed a set of 45,360 problems where 19,440 are for $n = 10$, 19,440 for $n = 15$ and 6480 for $n = 20$. The proposed branch and bound is able to solve optimally all instances of up to 15 jobs and two machines. Some cases with 20 jobs and eight machines are also solved optimally, but not all of them. The results are compared against the branch and bound proposed in [15] and the former is shown to outperform the latter in terms of both CPU time and number of problems solved.

As we can see, only the most recent exact methods are able to solve problems of up to 20 jobs and eight machines. Moreover, each method was tested on a different set of randomly generated problems. So, it is not completely clear which exact algorithm produces the best results. Heuristic methods are necessary to solve larger problems of more realistic sizes and in the next sections we review most well-known heuristics and metaheuristics for this problem.

3. Heuristics for the *m*-machine total tardiness problem

Heuristic algorithms can be broadly classified into *despatching rules*, *constructive* and *improvement heuristics*. Constructive heuristics build a schedule from scratch by making a series of passes through the list of unscheduled jobs where at each pass one or more jobs are selected and added to the schedule. Contrary to constructive heuristics, improvement heuristics start from an existing solution and apply some improvement procedure. In the next subsections we present several heuristics according to the above classification.

3.1. Despatching rules

Despatching rules are the most classical and well-known methods to build a schedule. These rules are used very often in practice and as an initial sequence in some improvement heuristic and metaheuristic methods. For this reason, in this section we briefly review most common despatching rules used for the total tardiness m-machine flowshop problem.

Let s be the sequence of jobs that are scheduled so far and t be the time at which jobs are considered for selection. Also, let $C_j(s)$ be the completion time of job $j \notin s$ if it is scheduled at the end of the sequence s.

- EDD: At time t, the job with minimum d_j value is selected.
- *Earliest due date with processing times* (EDDP): At time t, the job with the minimum value of $d_j/\sum_{i=1}^{m} p_{ij}$ is selected.
- *Modified due date* (MDD): At time t, the job with the minimum value of $\max\{d_j, C_j(s)\}$ is selected.
- *Earliest weighted due date* (EWDD): At time t, the job with the minimum value of the $w_j d_j$ is selected.
- SLACK: At time t, the job with the minimum value of $d_j - C_j(s)$ is selected.
- *Slack per remaining work* (SRMWK): At time t, we select the job with the minimum value of $(d_j - C_j(s))/\sum_{i=1}^{m} p_{ij}$.
- SPT: At time t, the job with the minimum value of $\sum_{i=1}^{m} p_{ij}$ is selected.
- *Longest processing time* (LPT): At time t, the job with the maximum value of $\sum_{i=1}^{m} p_{ij}$ is selected.

There are a host of despatching rules apart of the aforementioned, more so for other criteria. A comparison and evaluation of several despatching rules can be found in [17,18]. Some of the aforementioned rules, specially EDD and SLACK, are common for total tardiness minimisation. Most papers about the m-machine flowshop total tardiness problem evaluate and compare these methods [19–21].

3.2. Constructive and improvement heuristics

In [22] four simple heuristics for the sum of weighted tardiness criterion are proposed. These heuristics are based upon the following principles:

- use a dynamic job despatching rule,
- give priority to the item which is most expensive to hold,
- fit the jobs together in such a way that the generated idle time is minimal.

The heuristics were tested on 50 problems randomly generated where the number of jobs and the number of machines were uniformly distributed between 4 and 10 and 2 and 5, respectively. The processing times were also generated using a uniform distribution over the integers 1 and 10 and similarly the due dates of the jobs were distributed uniformly between $\sum_{i=1}^{m} p_{ij}$ and $25n$. The authors compared the heuristic solution with optimum values obtained for smaller problem sizes. For larger problem sizes the comparison is made between the four heuristics proposed. The best results were obtained by the second heuristic.

One of the most known rules is [19], where the author reported a heuristic called the idle time rule (IDLE). In this case, the study deals with the *proportionate flowshop* where a constant of proportionality k_i is associated with each machine and a job has processing times with values $p, k_2 p, \ldots, k_m p$ on the respective machines. Therefore, the processing times of each operation are proportionate and the problem is described by the constants of proportionality $(1, k_2, \ldots, k_m)$. In this case, the bottleneck machine is defined as the one that forces succeeding machines to be idle because it is unable to complete jobs on time. The bottleneck machine usually is the machine at which operation times are proportionally longer than at any other machine. When the bottleneck machine is free at time t, the job with the highest priority is scheduled. In the m-machine flowshop case, the bottleneck machine is first determined and then the priority for every job is computed, scheduling the one with the highest priority. The computational experiments were carried out using three problem sets with three, four and eight machines, respectively. The processing times were uniformly distributed according to the constant of proportionality and the number of jobs. Due dates are generated from a distribution using a mean due date. The results were compared to the optimal solution obtained by a branch and bound for the 10 jobs and three machines case. For larger problems with 25 jobs, four machines and 60 jobs, eight machines, the performance of the proposed heuristic was compared against several despatching rules such as EDD and SPT. The results showed that the proposed method produced the best results.

In [20], we find a set of heuristics to minimise the mean tardiness m-machine problem. The following heuristics, originally developed to minimise other objectives, are adapted to the problem of minimising mean tardiness:

- The second heuristic of [22]: According to the results shown in [22] the authors selected the second heuristic since it was shown to be the best.
- NEH heuristic [23]: This heuristic is regarded as the best heuristic for the permutation flowshop scheduling problem with the makespan minimisation criterion [3]. In the original NEH algorithm, jobs are sorted in non-increasing order of the sum of processing times on all machines. When due dates are considered, there are several methods for sorting the jobs. In [20], jobs are sorted following the EDD rule, that is, in non-decreasing order of due dates. Furthermore, a different way to sort the jobs is proposed where they are sorted in non-increasing order of due dates, that is, latest due date (LDD) rule. These methods are called $\mathrm{NEH}_{\mathrm{edd}}$ and $\mathrm{NEH}_{\mathrm{ldd}}$, respectively.

Moreover, in [20] some of the already commented priority rules are modified for the mean tardiness objective: EDD, SLACK, SRMWK and MDD. Also, an improvement heuristic which starts from the solution given by the EDD rule is proposed. In this case, the initial solution is improved by interchanging pairs of jobs and this heuristic is called ENS. All these methods are compared on 1000 randomly generated test problems where the processing times are generated from a uniform distribution with a range from 1 to 35 and the due dates are generated following the method proposed in [10] as mentioned before. The *tardiness factor* (T) was tested from 0.1 to 0.5 in steps of 0.1 and the *due date range* (R) from 0.8 to 1.8 in steps of 0.2. Several problem sizes were proposed with $n = \{15, 20, 30, 40, 50\}$ and $m = \{5, 10, 15, 20, 25\}$. The results showed that the $\mathrm{NEH}_{\mathrm{edd}}$ heuristic showed a good performance but the best results were obtained with the ENS method. In the same work, a tabu search was also proposed which will be reviewed in the next section.

In [24] several rules and heuristic algorithms for the single machine and two-machine cases are adapted and evaluated for the multi-machine flowshop problem. The method presented in [19] is extended to consider m machines such that each machine in the flowshop is considered to be a bottleneck and a complete schedule is developed following the method presented in [19]. At the end of the process, there are m possible schedules and the best one is selected. This approach is called *Modified Focused Scheduling* (MFS) method. In a similar way, the *Botflow* rule described by [25] and originally proposed for one machine is modified to consider the m-machine problem. This method obtains one schedule for each machine and at the end the best one is selected. This heuristic is also called *Botflow* rule. An improvement heuristic is also proposed which starts from the *Botflow* solution and improves it iteratively until there is no improvement in the tardiness value in two consecutive iterations. This heuristic is called *Flowshop Decomposition* (FSD) and the method used to improve the initial solution is based on the interchange of adjacent jobs. In the same work, we can find two approaches based on the shifting bottleneck method [26]. The methods were tested on problems randomly generated such that the processing times were distributed uniformly over the integers 1 and 20. Due dates were computed using a uniform distribution over the range $P(1 - R/2)$ and $P(1 + R/2)$ where $P = C_{\max}(1 - T)$ is the average job due date and $C_{\max}$ is the makespan of the sequence according to the FCFS rule. The T and R are the aforementioned parameters often used to obtain due dates where $T = \{0.05, 0.25, 0.5, 0.75\}$ and $R = \{0.5, 1.5\}$. Different problem sizes were tested where $n = \{25, 50\}$ and $m = \{4, 8\}$. For each scenario, 30 problems were generated, in total, 960 problems were solved for each heuristic method. Results showed that the *Botflow* method performs well at low T values and increasing T results in better performance for the MFS method. However, the FSD rule outperformed the remaining methods in all cases.

In [21] the authors proposed several improvement heuristics. First, two local search algorithms called ENS1 and ENS2 were presented. Both algorithms start from the NEH_{edd} solution and an improvement procedure based on insertion and interchange of jobs is applied, respectively. In the same work, a new algorithm called *rolling block optimisation* (RBO) is presented which starts from the NEH_{edd} solution. In this case, the improvement method is based on the permutation of several consecutive jobs (called a block). The number of jobs to be permuted in the block (b) is a parameter to be considered. The RBO algorithm improves the initial sequence by examining all the alternatives for the order of the b jobs in the block while maintaining the remaining jobs in their original places. Once the best permutation is selected, the first c positions of the block are fixed and the next b jobs after the c fixed jobs form the new block. The procedure is repeated until no more blocks have to be considered. Then, a backward rolling method is applied in a similar way but in this case the blocks of jobs are selected from the last block to the first one in the sequence. The method can be repeated until no improvement is found which results in another algorithm referred to as MRBO by the original authors. Four versions of the MRBO method are proposed attending to the values of the parameters b and c (MRBO1, MRBO2, MRBO3 and MRBO4). For the comparison of the algorithms, the authors randomly generated 480 test instances. In these problems, the number of jobs ranges from 15 to 40 and the number of machines from 10 to 16. Due dates were generated using the two parameters T and R where $T = \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and $R = \{0.8, 1, 1.2, 1.4, 1.6\}$ and P is a lower bound of the makespan of the sequence. The authors compared ENS1, ENS2 and the MRBO algorithms against the NEH_{edd} heuristic and the EDD rule. The results showed that the insertion method (ENS1) gave better results than the interchange method (ENS2) but required more computation time. The performance of the MRBO method was not as good as expected. The best results were obtained with $b = 6$ and $c = 1$ (MRBO1) and this method produced the best solution in several problems but bad solutions in others. Several metaheuristics were also proposed and compared in the same work which will be presented in the next section.

4. Metaheuristics for the *m*-machine total tardiness problem

Metaheuristics are general methods that guide the search through the solution space, using as surrogate algorithms some form of heuristics and usually local search. Starting from an initial solution built by some heuristic, metaheuristics improve it iteratively until a stopping criterion is met. The stopping criterion can be elapsed time, number of iterations, number of evaluations of the objective function and so on. A first classification of metaheuristic methods can be made according to the type they are based on: *simulated annealing*, *tabu search*, *genetic algorithms*, etc.

We can find in [27] a modified version of the tabu search of [28] originally proposed for the makespan objective. The initial solution is given by the heuristic proposed in [19] and the author modifies the tabu search algorithm reducing the size of the neighbourhood. The algorithm was tested on 480 randomly generated problems. These comprehend 24 sets of instances where $n = \{10, 15, 20, 25, 30, 35\}$ and $m = \{5, 10, 15, 20\}$. For each set, there are 20 problems and the authors do not give details about how the processing or the due dates are generated. The results obtained are better than with the original version of [28] and the computational performance is improved due to the restricted neighbourhood.

In [20], another modified version of the tabu search of [28] is proposed. In this case, the initial solution is given by the EDD rule. The author compares the proposed tabu search against some despatching rules and heuristics like EDD, SLACK, SRMWK, MDD, NEH_{edd} and ENS which were outperformed by the former tabu search in all the test problems.

In [29], a hybrid algorithm based on simulated annealing and tabu search are presented. Starting from the solution of [19], the original parameters of the method are changed after some computational experiments and a simulated annealing algorithm is applied to improve it. This solution is used as the initial one by a tabu search method where the size of the neighbourhood is restricted. The author compares the solution obtained against a tabu search starting from the same initial solution without any limitation in the size of the neighbourhood. The comparison is carried out with 720 randomly generated problems where $n = \{10, 15, \ldots, 50\}$ and $m = \{5, 10, 15, 20\}$. The results show that the same final solution is obtained but the number of iterations is smaller in the case of the hybrid algorithm using a restricted neighbourhood.

In [21] several algorithms were proposed, including tabu search and simulated annealing methods with the objective of minimising mean tardiness. After an experimental study to set the best values for the parameters, four tabu search methods and four simulated annealing methods, all of them starting from the solution given by the heuristic NEH_{edd}, were presented:

- TS1: Moves are based on job insertion (insertion neighbourhood) and a full evaluation of the neighbourhood is carried out. The tabu list consists of the objective values obtained in previous iterations, that is, it is not allowed to come back to an objective value which is in the tabu list. The tabu list size is set to 10.
- TS2: Similar to TS1 but moves are based on job interchange (interchange neighbourhood).
- TS3: Moves are based on job insertion (insertion neighbourhood) as in TS1. A sequence is represented by a graph such that nodes denote jobs and directed arcs denote the order of two jobs, that is, an arc (i, j) exists only when job i comes before job j in the sequence. The tabu list is defined by these arcs and a job i cannot precede a job j if the arc (i, j) is in the tabu list. The tabu tenure of the arcs is defined by a number of iterations set according to the number of jobs in the instance. Moreover, a procedure to penalise some movements according to the frequency that an arc belongs to a sequence is presented.
- TS4: Similar to TS3 but the number of iterations that an arc stays in the tabu list takes a different value and a different function to penalise the movements is applied.
- SA1, SA2: Simulated annealing methods where moves are based on insertion neighbourhood. A full evaluation of the neighbourhood is carried out. The difference between both methods relies on the cooling scheme.
- SA3, SA4: Similar to SA1 and SA2 but considering the interchange neighbourhood.

The performance of the algorithms was compared against several heuristics proposed in the same work and that have been reviewed in the previous section. Among the tabu search methods, TS3 and TS4 produced the best results and these were very similar. The TS2 showed a very poor performance but it took less computation time. The simulated annealing SA1 gave the best results outperforming the remaining heuristics and metaheuristics. The SA2 method showed a good performance too and the results obtained with SA3 and SA4 algorithms were the worst among all metaheuristics.

Two very similar simulated annealing methods were presented in [30,31] where the objective is to minimise the mean weighted tardiness in a flowshop with sequence-dependent set-up times. The algorithm starts from a solution given by the EWDD rule and a local search procedure is applied to improve it. The algorithms were tested on problems with 95 jobs and processing times from 6 to 600 min. Due dates of the jobs are generated as multiples of the sum of processing times of a job over all machines. The authors compare the results against the tabu search presented in [20] and the heuristics proposed in [22] adapted to this problem obtaining better results.

Two simulated annealing algorithms are proposed in [32]. The initial solution for both methods is given by a specific rule proposed in the same paper. One of the simulated annealing algorithms uses a perturbation scheme similar to that presented in [30,31] and the other one uses a perturbation scheme slightly different from the former. The algorithms are tested on different problem sizes such that $n = \{10, 15, 20, 25, 30\}$ and $m = \{5, 10, 15, 20, 25, 30\}$. For each $n \times m$, 30 problems are generated. Processing times are distributed uniformly from 1 to 99 and the due dates of the jobs are set following the method proposed in [22]. Results are compared against the tabu search presented in [20] and the heuristics reported in [22] and the two simulated annealing methods produced the best results.

In [33], a genetic algorithm to minimise tardiness is presented. Three different objective functions are considered: minimising total tardiness, minimising number of tardy jobs and minimising both objectives at the same time. The initial population is randomly obtained and the two-point crossover and swap mutation operators are employed. The algorithm uses a diversity measure; when the diversity falls below a predetermined value, the population is diversified using the mutation operator. The performance of the algorithm is tested on randomly generated problems where $n = \{5, 10, 20, 30\}$ and $m = \{5, 10, 15\}$. Due dates are generated randomly. Results are compared against the heuristic proposed in [34] and the proposed genetic algorithm is shown to improve results only in some cases.

Another tabu search method is that of [35]. The authors proposed a basic procedure and then they included diversification, intensification and neighbourhood restriction strategies as options. The method starts from the solution given by the MDD rule and a local search in the insertion neighbourhood is applied to improve the solution. The attributes of the tabu list are the jobs selected for insertion and they cannot be chosen for insertion again until their tabu tenure ends. The aspiration criterion is met when a solution is better than the best solution found by the search. The tabu tenure is set dynamically every 20 iterations. The diversification strategy is based on the frequency of each job appearing at each position and a penalty is applied to the value of each candidate move. Diversification is activated after k iterations without improvement in the objective function. The intensification approximation is based on storing a list with the five best solutions found in the process. When the solution is not improved during a period of time, search is reinitialised from one of these five solutions. Finally, the neighbourhood restriction strategy is based on reducing the size of the insertion neighbourhood in order to reduce the computational effort. Computational experiments were carried out using 240 randomly generated test problems where processing times were distributed uniformly over the integers 1 and 99. Due dates were generated following the method proposed in [10] where P is a lower bound of the makespan, $T = \{0.2, 0.4\}$, $R = \{0.6, 1.2\}$, $n = \{20, 50\}$ and $m = \{5, 10, 20\}$. For each combination 10 problems were generated. The basic tabu search (TSB) is compared to the NEH$_{\text{edd}}$ algorithm obtaining the former better results. Regarding the three strategies proposed, the comparison is carried out against the TSB method. The tabu search versions with the different options are shown to improve the results, albeit only slightly.

In [36] a new heuristic and metaheuristic method to minimise the sum of weighted flowtime and weighted tardiness of jobs considering set-up times are proposed. Two sequences are generated, the first one based on the processing times and set-up times and the second sequence is based on due dates of the jobs. The better of these two sequences according to the minimum sum of weighted flowtime and weighted tardiness of jobs is chosen and improved by job insertion. The metaheuristic method consists of an improvement scheme applied once or twice. The authors compared the performance of the heuristic method against the best sequence obtained by the first two heuristics proposed in [22]. Moreover, the comparison is extended applying the improvement scheme once and twice to this heuristic. Computational experience was carried out using 480 small-sized problems and 1200 large-sized problems. For the former problems, the number of jobs is seven and eight and the number of jobs in the latter case are from 10 to 30 in steps of five. In both cases the number of machines are from 5 to 20 in steps of five. The processing times are uniformly distributed over the range 1–99 and the due dates are computed as a function of the mean set-up time, the sum of the processing times and the mean processing time. Results show that the proposed heuristics outperform by far the heuristics proposed in [22]. With respect to the application of the improvement scheme once or twice, results show that two applications to the proposed heuristic are the best in most cases.

In [37], a simulated annealing method to minimise total tardiness is presented. The initial solution is given by a specific rule proposed in [32] which is improved through a perturbation scheme. Then, the simulated annealing is applied to improve this initial solution where two perturbation schemes are applied, the first is very similar to that proposed in [32] and the second one is based on jobs interchange. The performance of the proposed algorithm is evaluated against the tabu search proposed in [35] and the simulated annealing presented in [32]. The test problems to evaluate the performance of the methods were based on the benchmark of [38] adapted to compute the due dates in the following way: let $T_j = \sum_{i=1}^{m} p_{ij}$ the sum of the processing times of job j, then the due date is obtained by $d_j = T_j \times [1 + u \times 3]$ where u is a uniform random number over the range [0, 1]. Several combinations of n and m were considered where $n = \{20, 50, 100\}$ and $m = \{5, 10, 20\}$. The results show that the simulated annealing algorithm obtains better results than the other two methods.

More recently, in [39] a differential evolution algorithm is proposed to minimise the makespan, the flowtime and the total tardiness in a flowshop. This type of method was originally designed to work with continuous variables so equations to transform numbers from real to integer values and vice versa are necessary. In this case, the initial population is composed of vector parameters and is randomly obtained. Then, a mutation mechanism is applied after the conversion to real values. The mutated arrays are transformed back into discrete values to be evaluated and finally the selection of the new mutated vectors to replace the current ones is carried out. Results are compared against the genetic algorithm of [33] considering the following combinations of n and m: (4, 4), (10, 5), (15, 8), (25, 10), (25, 15), (50, 20), (75, 25), (100, 30). In the case of the total tardiness objective function, the differential algorithm is outperformed by the genetic algorithm presented in [33] as shown by the authors.

As we can see from the methods reviewed, all of them were evaluated using different randomly generated test problems. In the next section, we propose a common benchmark of instances in order to make a global and fair comparison of all the methods.

Table 1
Despatching rules and heuristics reviewed

Year	Author(s)	Objective	Type	Comments
1978	Gelders and Sambandam	Weighted tardiness	*C*	Four heuristics based on a dynamic despatching rule
1985	Ow	Total tardiness	*C*	Heuristic based on the idle time rule for the proportionate flowshop
1993	Kim	Mean tardiness	*C*/*D*	Heuristics originally developed for the makespan objective adapted to the mean tardiness objective
1995	Raman	Total tardiness	*C*/*I*	Heuristics based on priority rules: MFS, Botflow, FSD
1996	Kim	Mean tardiness	*I*	Heuristic based on the permutation of several consecutive jobs
2003	Rajendran and Ziegler	Weighted flowtime and weighted tardiness	*C*	New heuristic method
Despatching	rules			
		Total tardiness	*D*	Earliest due date with processing times
		Total tardiness	*D*	Modified due date
		Total tardiness	*D*	Earliest weighted due date
		Total tardiness	*D*	Minimum slack
		Total tardiness	*D*	Slack per remaining work
		Total tardiness	*D*	Shortest processing time
		Total tardiness	*D*	Longest processing time

C = constructive heuristic, *I* = improvement heuristic, *D* = despatching rule.

Table 2
Metaheuristics reviewed

Year	Author(s)	Objective	Type	Comments
1992	Adenso-Díaz	Total weighted tardiness	TS	Based on the Widmer and Hertz TS Initial solution given by Ow heuristic
1993	Kim	Mean tardiness	TS	Modified version of the Widmer and Hertz TS Initial solution given by the EDD rule
1996	Adenso-Díaz	Total weighted tardiness	SA/TS	Hybrid algorithm based on SA and TS starting from the solution of the Ow heuristic
1996	Kim et al.	Mean tardiness	SA/TS	Four TSs and four SAs proposed starting from the NEH_{edd} heuristic
1997	Parthasarathy and Rajendran	Mean weighted tardiness	SA	SA starting from the EWDD rule and local search applied to improve the solution
1997	Parthasarathy and Rajendran	Mean weighted tardiness	SA	SA starting from the EWDD rule and local search applied to improve the solution
1998	Parthasarathy and Rajendran	Mean weighted tardiness Mean tardiness	SA	Two SAs starting from a rule proposed in the same work Perturbation schemes are applied to improve the solution
1999	Onwubolu and Mutingi	Total tardiness	GA	Random initialisation, two-point crossover and swap mutation
1999	Armentano and Ronconi	Total tardiness	TS	Starts from the MDD solution. Basic tabu search Diversification, intensification and restricted neighbourhood
2003	Rajendran and Ziegler	Weighted flowtime and weighted tardiness	LS	Improvement of a heuristic method
2004	Hasija and Rajendran	Total tardiness	SA	Initial solution given by a despatching rule Local search procedures
2006	Onwubolu and Davendra	Total tardiness	DE	Initial population is generated randomly Conversion from discrete to real values and vice-versa

TS = tabu search, SA = simulated annealing, GA = genetic algorithm, DE = differential evolution, LS = local search.

5. Evaluation and comparison of heuristics and metaheuristics

Table 1 shows a summary of the different despatching rules and heuristics reviewed in this paper and in Table 2 we can find information about the metaheuristics.

Table 3
Constructive and improvement heuristics implemented

Year	Author(s)	Acronym	Type	max ($n \times m$)	Due dates
1970	Campbell et al. [34]	CDS	C		–
1978	Gelders and Sambandam [22]	GS1	C	10×3	S
		GS2	C	10×3	S
		GS3	C	10×3	S
		GS4	C	10×3	S
1985	Ow [19]	IDLE	C	60×8	S
1993	Kim [20]	NEH_{edd}	C	50×25	G
		ENS	I	50×25	G
1995	Raman [15]	MFS	C	50×8	S
		Botflow	C	50×8	S
		FSD	I	50×8	S
1996	Kim et al. [21]	ENS1	I	40×16	G
		ENS2	I	40×16	G
		MRBO1	I	40×16	G
		MRBO2	I	40×16	G
		MRBO3	I	40×16	G
		MRBO4	I	40×16	G
2003	Rajendran and Ziegler [36]	HA	C	30×20	S
Despatching rules		EDD	D		
		MDD	D		
		EDDP	D		
		SLACK	D		
		SRMWK	D		

C = constructive heuristic, I = improvement heuristic, D = despatching rule, S = due dates generated following a specific method, G = due dates generated following the general method proposed in [10].

Table 4
Metaheuristics implemented

Year	Author(s)	Acronym	Type	max($n \times m$)	Due dates
1989	Osman and Potts [40]	SA_OP	SA	35×20	–
1992	Adenso-Díaz [27]	H	TS	35×20	?
1993	Kim [20]	TS	TS	50×25	G
1996	Adenso-Díaz [29]	SA_TS	SA/TS	50×20	?
1996	Kim et al. [21]	TS1	TS	40×16	G
		TS2	TS	40×16	G
		TS3	TS	40×16	G
		SA1	SA	40×16	G
		SA3	SA	40×16	G
1997	Parthasarathy and Rajendran [30]	SAH	SA	95×9	S
1999	Onwubolu and Mutingi [33]	GA	GA	30×15	R
1999	Armentano and Ronconi [35]	TSB	TS	50×20	G
		TSD	TS	50×20	G
		TSR	TS	50×20	G
2003	Rajendran and Ziegler [36]	HA+IS2	LS	30×20	S
2004	Hasija and Rajendran [37]	SRH	SA	100×20	S
2006	Onwubolu and Davendra [39]	DE	DE	100×30	$?/R$

TS = tabu search, SA = simulated annealing, GA = genetic algorithm, DE = differential evolution, LS = local search, S = due dates generated following a specific method, G = due dates generated following the general method proposed in [10], R = due dates randomly generated.

The heuristics and despatching rules that are coded and evaluated are summarised in Table 3 and regarding the metaheuristics, the algorithms implemented can be seen in Table 4. We can also see in both tables the maximum size of the instances solved by the different methods and the way the due dates were generated in the original papers. Almost all the methods reviewed are selected and implemented. Nevertheless, implementation of the genetic algorithm presented in [33] is not exactly as stated in the original paper due to the absence of some details. Moreover, the following methods are discarded from the comparison due to different causes:

- NEH_{ldd} [20]: Since the NEH_{edd} heuristic proposed in the same work performs better in all cases, we decided not to implement NEH_{ldd}.
- TS4 [21]: Four tabu search methods were presented and the third one (TS3) and the last one (TS4) result in a very similar performance. For this reason, we decided to implement TS3 only.
- SA2, SA4 [21]: Four simulated annealing methods were presented, and the only differences between SA1 and SA2 and SA3 and SA4 were the stopping criteria. Since in this work the stopping criterion is the same for all the metaheuristics (elapsed time), we only needed to implement two of the four methods (SA1 and SA3).
- SA proposed in [32]: Due to the resemblance of this simulated annealing with those presented in [30,31], we did not code this method.
- Tabu search presented in [35] (intensification version): We did not code this version of the tabu search due to the absence of details in the original paper.

The remaining methods are implemented following the explanations and implementation details given in the original papers. All algorithms are coded in Delphi 2006 and run on a Pentium IV 3.0 GHz with 1 GB of main memory. In order to make a fair comparison, the stopping criterion for all the metaheuristics is set to a maximum CPU elapsed time of $n \cdot (m/2) \cdot 90$ ms. Setting the time limit in this way allows more computation effort as the number of jobs and/or the number of machines increases. Moreover, all the metaheuristics are run five independent times to obtain a final average of the results. We also include in the comparison the simulated annealing of [40] and the method presented in [34], both proposed originally for the makespan criterion. Therefore, we coded and evaluated a total of 40 methods, 23 heuristics and despatching rules and 17 metaheuristics.

5.1. Benchmark instances

As we can see from the methods reviewed, it is close to impossible to make a global comparison of the different methods. Furthermore, without the instance sets given in each paper, results are hard to reproduce. Moreover, in Tables 3 and 4 we can see that the size of most of the generated problems is small, up to 100 jobs and 20 machines maximum. For all these reasons, in this work we propose a common benchmark of instances to enable comparisons between the different methods and algorithms.

We have generated nine different sets of instances each containing 60 problems. The processing times of the jobs in the machines are uniformly distributed between 1 and 99 as it is common in the literature. The due dates are generated according to the *tardiness factor* (T) and the *due date range* (R) with a uniform distribution between $P(1-T-R/2)$ and $P(1-T+R/2)$ following the method presented in [10] which is often used in the literature as has been shown. The P is a tight lower bound of the makespan proposed in [38]. Taking into account the combinations of the parameters T and R considered in the papers reviewed in previous sections, the following combinations are proposed: $T = \{0.2, 0.4, 0.6\}$ and $R = \{0.2, 0.6, 1\}$. This results in nine combinations. In some cases, specially for low values of T and large values of R some due dates can be negative. In such cases we replace the negative due dates by zero. Regarding the number of jobs and machines, the following values are selected: $n = \{50, 150, 250, 350\}$ and $m = \{10, 30, 50\}$.

In total, we have 12 combinations for n and m and we generate five replicates for each one, that is, 60 instances for each T and R combination. Therefore, we have 540 test problems in total according to the configuration of T, R, n and m explained above. It is important to remark that all selected values are equidistant, which facilitates the statistical analyses. Also, the size of the problems is larger, by far, than that of the problems used in the reviewed papers. These benchmarks are available from http://www.upv.es/gio/rruiz and also upon request from the authors.

5.2. Computational evaluation

The evaluation of all the methods implemented is carried out using the benchmark defined in the previous section. The most common performance measure used in the literature to compare all the methods is the relative percentage

Table 5
Relative deviation index (*RDI*) for the heuristic methods

Instance	CDS	GS1	GS2	GS3	GS4	IDLE	NEH_{edd}	ENS	MFS	Botflow	FSD	ENS1
50 × 10	76.52	62.95	64.35	54.35	50.01	37.50	19.66	41.58	25.78	38.71	38.12	4.89
50 × 30	60.16	58.33	58.23	77.18	47.45	60.00	11.26	60.40	37.33	56.94	56.63	2.90
50 × 50	52.57	54.56	53.81	78.13	40.16	58.05	8.18	62.64	35.45	61.76	59.85	1.61
150 × 10	89.07	70.73	70.52	58.04	48.80	32.54	26.31	35.62	24.75	33.34	33.92	6.65
150 × 30	81.79	86.52	86.79	78.76	70.17	54.18	16.90	54.22	38.49	48.17	46.94	3.34
150 × 50	73.55	84.77	86.04	82.13	68.11	57.97	12.61	61.16	42.45	53.81	52.04	2.21
250 × 10	91.49	69.25	68.94	56.19	48.93	26.80	28.86	28.91	21.78	26.34	29.00	9.36
250 × 30	85.14	91.62	92.57	73.40	75.45	46.56	20.54	45.69	34.90	40.46	39.83	5.27
250 × 50	81.68	91.85	92.43	84.02	79.44	52.00	16.21	56.35	41.47	47.82	46.78	4.03
350 × 10	92.63	65.38	67.04	53.23	48.32	24.90	28.56	26.67	20.76	29.57	28.86	9.07
350 × 30	87.99	94.78	95.15	70.93	80.80	43.67	21.26	42.85	34.45	37.40	36.35	5.32
350 × 50	83.61	94.24	93.98	80.06	81.82	47.11	17.98	46.31	37.49	40.63	39.43	4.39
Average	79.68	77.08	77.49	70.54	61.62	45.11	19.03	46.87	32.92	42.91	42.31	4.92

	ENS2	MRBO1	MRBO2	MRBO3	MRBO4	HA	EDD	EDDP	MDD	SLACK	SRMWK
50 × 10	2.01	3.83	3.37	5.29	6.35	39.53	55.58	82.33	45.45	59.59	52.06
50 × 30	3.21	1.93	2.09	3.26	5.33	46.44	80.93	93.12	63.05	86.29	77.84
50 × 50	3.67	1.67	2.25	3.32	4.17	40.38	85.19	94.81	66.10	89.06	79.22
150 × 10	0.00	15.20	13.56	11.92	13.56	31.99	40.68	71.43	35.73	42.37	38.67
150 × 30	0.64	8.07	7.58	7.97	9.29	47.59	60.51	75.76	57.84	63.38	58.79
150 × 50	1.07	5.73	5.95	6.88	7.80	52.99	66.94	77.77	64.15	69.83	65.21
250 × 10	0.53	20.97	20.55	17.65	18.78	23.27	31.84	62.44	29.03	32.24	27.79
250 × 30	0.21	13.84	12.59	12.59	13.18	45.33	49.29	64.68	49.38	49.92	47.21
250 × 50	0.29	10.45	9.85	9.72	10.56	54.23	59.55	70.20	59.18	61.08	56.66
350 × 10	0.41	22.25	22.92	19.72	20.05	22.31	28.56	59.54	26.48	29.27	26.26
350 × 30	0.05	15.29	14.79	13.35	14.08	41.95	45.63	59.11	49.38	45.95	42.20
350 × 50	0.18	12.64	12.27	11.69	12.27	46.25	49.03	60.23	54.16	50.06	47.23
Average	1.02	10.99	10.65	10.28	11.29	41.02	54.48	72.62	49.99	56.59	51.60

deviation (*RPD*) which is computed in the following way:

$$\text{Relative percentage deviation } (RPD) = \frac{Method_{\text{sol}} - Best_{\text{sol}}}{Best_{\text{sol}}} \cdot 100, \qquad (1)$$

where $Method_{\text{sol}}$ is the solution obtained by a given method and $Best_{\text{sol}}$ is the best solution obtained among all the methods or the best known solution, possibly optimal. However, in the case of the total tardiness flowshop problem, the best solution could be zero (and therefore optimal), so the above equation gives a division by zero. Moreover, if the best solution is a small value, the performance measure underestimates an algorithm which obtains a solution slightly worse than the best one. Therefore, a different performance ratio is used in this work to avoid this problem and following that proposed in [41] and used in [20,21]:

$$\text{Relative deviation index } (RDI) = \frac{Method_{\text{sol}} - Best_{\text{sol}}}{Worst_{\text{sol}} - Best_{\text{sol}}} \cdot 100, \qquad (2)$$

where $Best_{\text{sol}}$ and $Worst_{\text{sol}}$ are the best and the worst solutions obtained among all the methods, respectively. With this measure, an index between 0 and 100 is obtained for each method such that a good solution will have an index very close to 0. Note that if the worst and the best solutions take the same value, all the methods provide the best (same) solution and, hence, the index value will be 0 (best index value) for all methods.

The results of the heuristic methods are shown in Table 5 where we have averaged the 45 instances of each $n \times m$ group. As we can see, the worst results are given by the methods proposed in [34], the EDDP rule and the four heuristics presented in [22]. This is an interesting result since the former and the latter methods are commonly used to compare the performance of some of the algorithms presented in the previous sections. Regarding the four heuristics proposed

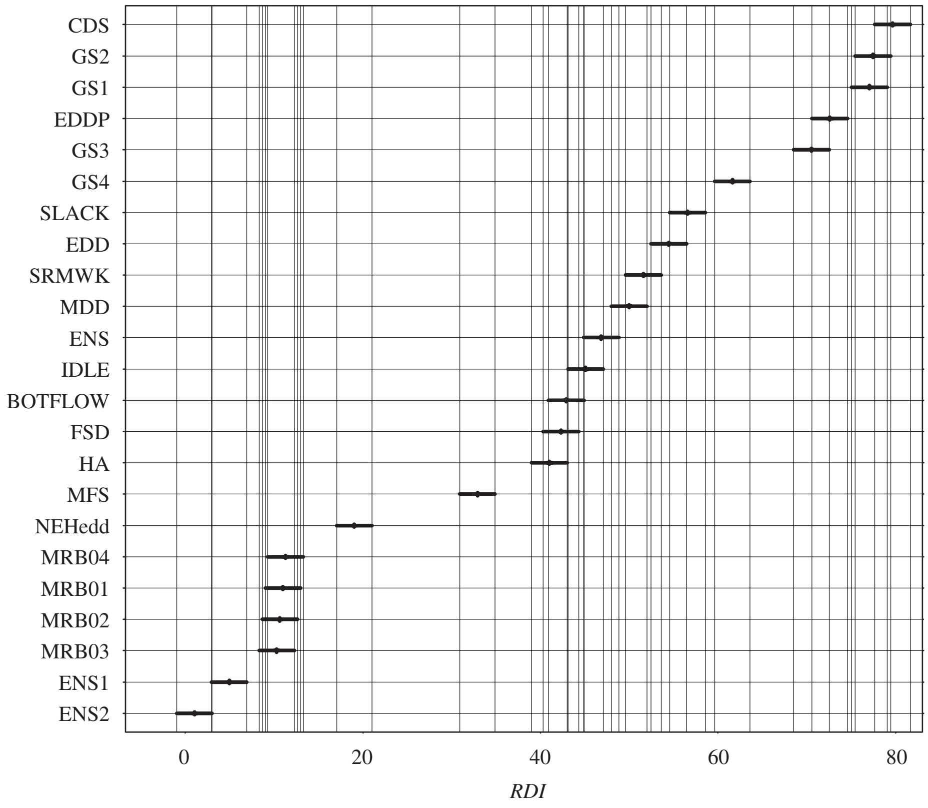

Fig. 1. Means plot and Tukey HSD intervals for the heuristic algorithm factor and the relative deviation index (*RDI*) (ANOVA parametric test).

in [22] we have to remark that the second one is the most used in the literature to compare algorithms since in the original paper the best results were obtained with this heuristic. We have to take into account that these heuristics were tested in the seventies using computers of that time and we can see that using more modern computers the best results are obtained with the fourth heuristic. Regarding the best heuristics, we can see that the two methods proposed in [21] based on interchange and insertion of jobs (ENS1 and ENS2) show the best results. In order to obtain a better picture of the results we have performed different statistical analysis where we consider the different algorithms as a factor and all the instances solved as treatments. We carried out an analysis of variance (ANOVA) [42] of heuristic methods, where we have 23 different heuristics and 12,420 treatments in total. We can see the means plot with Tukey intervals in Fig. 1. Results indicate that there are statistically significant differences with a *p*-value very close to zero for the algorithm factor. It is important to remark that in the ANOVA analysis there are three hypotheses which have to be checked: homogeneity of the variance (homoscedasticity), normality and independence of the residuals. Since in this experiment the hypotheses are slightly not fulfilled, the ANOVA loses power. For a more suitable analysis we performed a *rank-based test* where a transformation of data is carried out such that a rank is assigned to each original value. We use the R language environment for statistical computing (http://www.r-project.org/) [43]. In this case, we have 23 heuristic methods and, for every instance, each algorithm is ranked from 1 to 23 depending on the *RDI* obtained. We performed the rank-based Friedman test [44] according to [45] to compute the minimal significant difference between rank means of any two algorithms. The results of the different rank means of the heuristic methods can be seen in Table 6 and the means plot for the different algorithms is shown in Fig. 2. The results indicate that there are statistically significant differences between the different algorithms. There are 17 homogeneous groups. The two best heuristic methods in both analyses are the ENS2 and ENS1 presented in [21]. After these two improvement heuristics we can see the four MRBO methods presented in the same paper which show a good and very similar performance. In this case it is important to remark that these improvement heuristics start from the NEH_{edd} solution and we can see the effect of the improvement scheme. It is shown in Fig. 2 that these improvement methods show statistically significant differences

Table 6
Average rank for the heuristic methods

Instance	CDS	GS1	GS2	GS3	GS4	IDLE	NEH$_{edd}$	ENS	MFS	Botflow	FSD	ENS1
50 × 10	20.20	17.00	17.36	15.64	13.93	12.42	9.04	12.69	8.62	11.73	12.42	4.36
50 × 30	14.36	14.36	14.33	18.80	11.87	14.71	7.02	14.82	9.09	14.04	13.60	3.58
50 × 50	13.07	13.69	13.49	19.02	10.58	14.29	6.98	15.53	9.07	15.40	14.51	2.71
150 × 10	22.29	19.31	19.11	16.84	14.51	10.29	11.24	11.18	6.87	10.04	11.58	2.53
150 × 30	18.47	19.69	19.84	18.33	15.56	13.27	9.29	12.58	7.76	10.31	10.09	2.96
150 × 50	16.04	19.11	19.60	19.09	14.69	13.47	7.73	14.04	8.27	11.51	10.42	2.22
250 × 10	22.44	19.33	19.40	17.96	14.56	9.29	12.44	10.29	6.47	9.13	11.84	3.33
250 × 30	20.44	21.33	21.44	17.60	17.09	12.40	9.69	11.07	7.29	8.89	8.98	3.73
250 × 50	18.24	20.71	21.11	19.29	17.20	12.00	8.98	12.84	7.73	9.82	8.84	3.09
350 × 10	22.62	19.42	19.93	17.64	14.40	8.82	12.73	8.91	5.98	12.69	11.60	2.91
350 × 30	20.20	21.38	21.44	17.11	18.73	11.58	10.38	10.73	7.13	9.47	8.24	3.11
350 × 50	19.18	21.29	21.29	18.78	18.24	12.42	9.44	11.16	7.47	9.64	8.27	3.56
Average	18.96	18.89	19.03	18.01	15.11	12.08	9.58	12.15	7.64	11.06	10.87	3.17
	ENS2	MRBO1	MRBO2	MRBO3	MRBO4	HA	EDD	EDDP	MDD	SLACK	SRMWK	
50 × 10	2.69	3.62	3.00	5.02	5.51	12.80	16.87	21.11	13.69	17.36	16.49	
50 × 30	3.51	2.27	2.76	3.60	5.11	11.53	19.80	21.36	15.16	20.56	19.07	
50 × 50	4.29	2.22	2.84	3.80	4.73	10.20	20.20	21.76	16.27	21.27	19.40	
150 × 10	1.00	8.11	6.49	5.58	6.76	11.00	13.91	20.27	11.13	14.58	13.13	
150 × 30	2.07	5.18	4.49	5.02	6.98	11.64	15.71	19.62	13.80	17.16	15.24	
150 × 50	2.13	3.53	3.91	4.93	5.80	12.60	16.93	19.87	15.49	18.02	16.00	
250 × 10	1.87	10.00	9.64	7.51	8.71	9.49	12.36	19.64	10.02	12.89	9.67	
250 × 30	2.40	7.40	5.49	5.82	6.64	12.09	13.76	18.56	13.20	14.13	12.89	
250 × 50	1.69	6.02	5.00	5.00	6.56	13.04	15.38	18.76	14.47	16.13	13.62	
350 × 10	1.76	10.07	10.87	8.18	8.47	8.67	11.09	19.31	8.87	11.76	9.20	
350 × 30	1.51	8.40	7.51	5.91	6.82	11.58	12.96	18.29	13.51	13.22	11.09	
350 × 50	2.02	7.16	6.42	5.49	6.82	12.49	13.62	18.09	15.36	14.20	12.56	
Average	2.24	6.16	5.70	5.49	6.58	11.43	15.21	19.72	13.41	15.94	14.03	

with respect to the initial solution provided by the NEH$_{edd}$ heuristic. Regarding the worst methods, we can see that the despatching rules and the classical methods proposed in [34,22] give the worst results. The MFS method proposed in [24] which is an extension of the IDLE rule [19] to the *m*-machine case shows good results. However, the FSD method presented in the same work which applied an improvement scheme to the Botflow rule does not show statistically significant differences with respect to the latter. Nevertheless, the FSD method starts from the solution of a slightly modified Botflow rule. Since the CPU time was impracticable for most of the instances (close to 1000 s for instances with 350 jobs and 50 machines), the number of iterations applied to the Botflow rule until there is no improvement in the tardiness value was reduced from two to one. It is important to remark that the NEH$_{edd}$ heuristic shows a good position in the ranking since it is used as an initial solution for several metaheuristics. Another interesting result is the average performance of the recent heuristic HA method proposed in [36], the results of which are statistically similar to those obtained by the IDLE rule [19] and the Botflow rule [24].

In order to analyse the efficiency of the different heuristics, we measured the CPU time, in seconds, that a given method needs to provide a solution. The results can be seen in Table 7 and a scatter plot which represents the average *RDI* versus the time needed for each method is shown in Fig. 3. We can observe that the despatching rules and the methods proposed in [34,36] and the first and fourth heuristics presented in [22] are very fast and all needed less than 0.5 s on average for all problem sizes. The Botflow rule [24] and the second heuristic reported in [34] are the slowest methods, taking more than a minute on average although the number of iterations applied to the Botflow rule until there is no improvement in the tardiness value was reduced from two to one for instances with 350 jobs. Regarding the four improvement methods MRBO1, MRBO2, MRBO3, MRBO4 [21] and the MFS and FSD rules [24], the time needed to provide a solution is greater than 30 s. Note that the improvement heuristic FSD needs less CPU time than the Botflow rule since the changes introduced in the latter clearly affect its efficiency as we explained above. The remaining

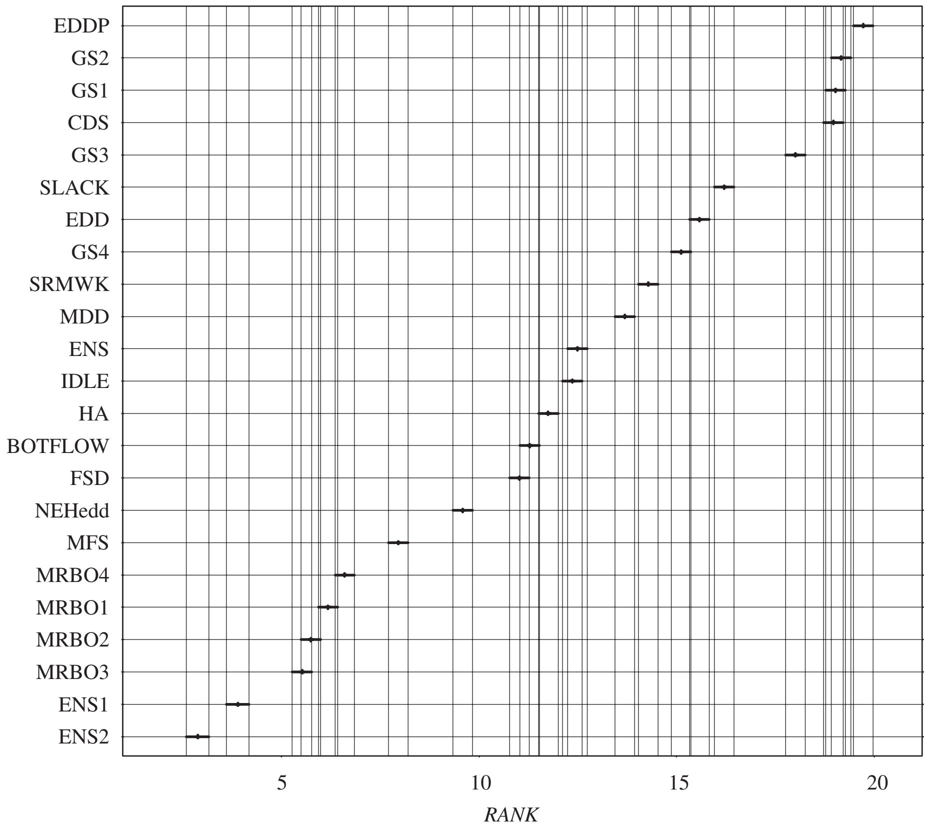

Fig. 2. Means plot for the heuristic algorithm factor and the average rank value (non-parametric test).

methods do not need more than 12 s to obtain the solution including the two best methods in terms of effectiveness, ENS2 and ENS1 which take less than 8 s on average. Therefore, ENS1 and ENS2 methods can be regarded as the best heuristic methods according to the results obtained with the statistical tests and the scatter plot.

Now we are going to analyse the results of the different metaheuristics reviewed which can be seen in Table 8. As before, we apply an ANOVA analysis to evaluate the differences between the algorithms and the means plot with Tukey intervals can be seen in Fig. 4. We can see that there are statistically significant differences but, as in the heuristic case, when we check the hypotheses of the analysis, these are slightly not satisfied. Therefore, we also evaluate the metaheuristic algorithms using the Friedman test [44] according to the minimal significant difference value computed following the formula proposed by Conover [45] extended to the several runs case [43], that is, all the metaheuristics are run five times and we analyse all the results. The average ranks are shown in Table 9 and the means plot can be seen in Fig. 5. Note that in this case, we have 17 metaheuristics but they are ranked from 1 to 85 due to the five replicates of the experiment. We can see from the results that 13 homogeneous groups are formed. The first interesting outcome is the poor performance of the DE method proposed recently in [39]. This differential algorithm seems not to be suitable for this problem and in the original paper the authors conclude that the performance is not good for the total tardiness objective. The best performing methods are the simulated annealing algorithms SAH and SRH proposed in [30,37], respectively. After these algorithms, the basic TSB and its diversification version reported in [35] show a good but very similar performance. The restriction version of the same tabu search performs slightly worse than the other ones. The tabu search and simulated annealing methods presented in [21] are ranked on an average position, the simulated annealing methods being slightly better. Regarding the GA proposed in [33] its performance is not so good and it is outperformed by the simulated annealing presented in [40] originally proposed for the makespan objective. The two adaptations of the tabu search of [28], TS and H1, proposed in [20,27] do not show a good behavior, especially the former which is the second worst algorithm. The hybrid algorithm SA_TS reported in[29] which used a

Table 7
Average CPU time, in seconds, used by the heuristic methods

Instance	CDS	GS1	GS2	GS3	GS4	IDLE	NEH$_{edd}$	ENS	MFS	Botflow	FSD	ENS1
50 × 10	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	0.11	< 0.5	< 0.5
50 × 30	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	1.04	< 0.5	< 0.5
50 × 50	< 0.5	< 0.5	1.08	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	0.62	2.51	0.69	< 0.5
150 × 10	< 0.5	< 0.5	1.53	< 0.5	< 0.5	< 0.5	< 0.5	0.79	0.70	3.01	0.74	0.63
150 × 30	< 0.5	< 0.5	10.95	1.26	< 0.5	< 0.5	< 0.5	2.31	5.97	23.63	6.35	1.66
150 × 50	< 0.5	< 0.5	29.01	2.24	< 0.5	< 0.5	0.67	3.93	16.08	57.32	17.08	2.67
250 × 10	< 0.5	< 0.5	6.99	1.86	< 0.5	< 0.5	0.71	3.67	3.18	15.54	3.29	2.84
250 × 30	< 0.5	< 0.5	51.05	5.92	< 0.5	0.95	1.98	11.19	28.19	109.64	29.57	7.99
250 × 50	< 0.5	< 0.5	134.54	10.53	< 0.5	1.55	3.15	18.09	76.58	283.01	79.54	12.81
350 × 10	< 0.5	< 0.5	19.10	5.05	0.54	0.89	1.96	10.28	8.75	8.83	8.96	7.85
350 × 30	< 0.5	< 0.5	139.85	16.45	0.63	2.66	5.56	29.52	79.57	82.00	82.53	22.38
350 × 50	< 0.5	< 0.5	370.01	28.95	0.71	4.27	8.72	52.42	212.95	217.47	219.21	35.22
Average	< 0.5	< 0.5	63.71	6.07	< 0.5	0.94	1.95	11.04	36.07	67.01	37.35	7.85

Instance	ENS2	MRBO1	MRBO2	MRBO3	MRBO4	HA	EDD	EDDP	MDD	SLACK	SRMWK
50 × 10	< 0.5	2.62	2.64	2.52	2.51	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
50 × 30	< 0.5	8.38	7.92	7.55	7.52	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
50 × 50	< 0.5	13.54	13.02	12.57	12.54	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
150 × 10	0.66	8.80	8.77	7.74	7.59	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
150 × 30	1.73	23.98	24.01	22.93	22.68	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
150 × 50	2.83	45.66	40.20	38.52	37.91	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
250 × 10	3.05	20.13	15.20	12.99	12.78	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
250 × 30	8.17	42.65	42.76	39.29	38.00	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
250 × 50	12.54	63.58	63.58	66.08	64.20	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
350 × 10	7.77	35.98	19.03	18.72	17.81	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
350 × 30	22.19	83.48	63.72	57.14	54.44	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
350 × 50	34.79	125.17	96.23	89.56	89.68	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5
Average	7.83	39.50	33.09	31.30	30.64	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5	< 0.5

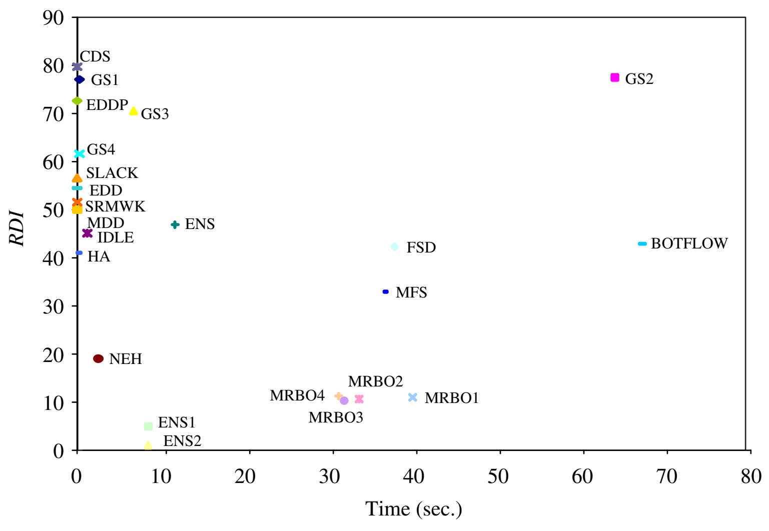

Fig. 3. Scatter plot for the heuristic methods evaluated.

Table 8
Relative deviation index (*RDI*) for the metaheuristic methods

Instance	SA_OP	H1	TS	SA_TS	TS1	TS2	TS3	SA1	SA3
50 × 10	14.57	19.55	24.14	19.32	23.50	15.98	18.76	26.75	19.44
50 × 30	30.71	57.52	61.38	53.06	41.37	45.09	36.43	50.88	53.21
50 × 50	29.62	59.01	65.35	57.69	42.39	51.32	38.99	53.91	60.56
150 × 10	5.40	5.57	5.77	5.15	9.04	4.93	9.00	9.94	5.54
150 × 30	5.37	12.97	13.22	11.64	10.91	10.58	9.74	11.89	12.20
150 × 50	5.31	16.66	16.53	14.99	11.19	12.44	10.54	12.36	14.09
250 × 10	4.26	3.22	9.17	10.17	9.29	5.64	9.34	9.38	5.61
250 × 30	4.42	9.10	13.80	15.46	8.80	9.68	8.71	9.01	10.01
250 × 50	4.05	12.45	15.27	17.47	9.97	10.77	9.74	10.31	11.41
350 × 10	4.70	9.09	30.70	36.31	13.81	13.03	13.88	13.81	13.18
350 × 30	4.93	16.83	36.29	44.56	13.30	15.20	13.22	13.36	15.54
350 × 50	3.57	19.14	35.66	46.70	12.60	14.00	12.55	12.77	14.37
Average	9.74	20.09	27.27	27.71	17.18	17.39	15.91	19.53	19.60
	SAH	GA	TSB	TSD	TSR	HA + IS2	SRH	DE	
50 × 10	7.21	12.55	6.42	6.75	9.67	15.21	2.12	99.78	
50 × 30	21.27	35.00	26.41	25.37	30.63	26.31	2.37	97.96	
50 × 50	21.50	35.77	25.73	25.96	30.86	25.67	2.67	96.86	
150 × 10	1.04	3.53	0.65	0.69	1.88	4.49	2.52	100.00	
150 × 30	2.29	8.44	5.37	5.36	6.25	4.04	1.76	100.00	
150 × 50	4.11	10.82	8.97	8.97	10.13	4.57	2.08	100.00	
250 × 10	0.80	3.70	0.52	0.52	0.59	3.54	2.56	100.00	
250 × 30	0.82	8.22	2.34	2.36	2.80	3.28	2.81	100.00	
250 × 50	1.69	10.06	5.91	5.91	6.48	2.84	3.01	100.00	
350 × 10	0.72	5.16	0.78	0.77	0.91	3.71	3.03	100.00	
350 × 30	0.75	9.77	1.63	1.63	1.52	2.93	2.97	100.00	
350 × 50	0.99	10.72	3.77	3.78	3.84	1.92	2.76	100.00	
Average	5.27	12.81	7.38	7.34	8.80	8.21	2.55	99.55	

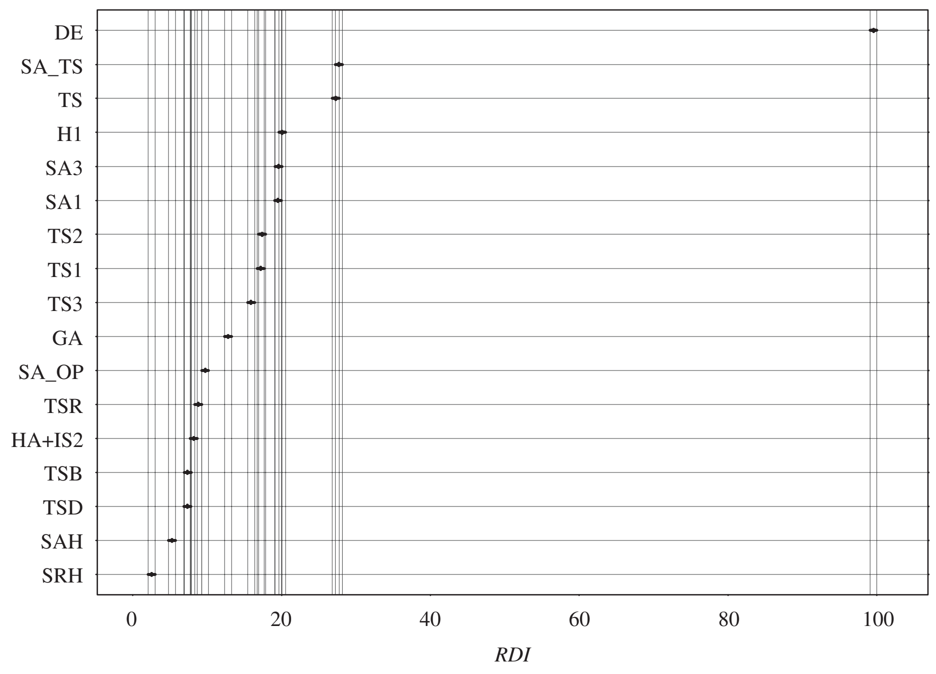

Fig. 4. Means plot and Tukey HSD intervals for the metaheuristic algorithm factor and the relative deviation index (*RDI*) (ANOVA parametric test).

Table 9
Average rank for the metaheuristic methods

Instance	SA_OP	H1	TS	SA_TS	TS1	TS2	TS3	SA1	SA3
50 × 10	7.38	10.09	11.50	9.52	12.43	7.52	8.48	12.32	10.87
50 × 30	6.73	12.72	13.74	12.22	9.49	10.64	8.00	12.15	12.85
50 × 50	6.40	12.82	13.61	12.68	8.73	11.37	8.00	11.99	13.61
150 × 10	7.46	8.34	8.74	7.75	13.98	7.50	10.23	11.70	8.58
150 × 30	5.17	12.22	13.14	11.05	11.02	10.43	8.84	11.98	12.99
150 × 50	4.04	13.78	13.85	12.61	8.88	10.44	8.07	10.92	12.24
250 × 10	6.06	5.47	10.88	11.09	13.85	8.12	10.90	10.36	7.89
250 × 30	5.25	9.69	14.21	14.93	10.87	10.94	9.06	10.32	12.12
250 × 50	3.80	11.74	14.40	15.32	9.52	10.10	8.74	10.76	11.67
350 × 10	4.86	7.49	15.18	15.64	11.40	11.16	10.18	9.69	11.66
350 × 30	5.27	11.15	15.09	15.91	10.43	11.24	9.75	10.72	12.35
350 × 50	4.21	11.63	14.99	15.98	10.24	10.88	9.78	11.17	11.70
Average	5.55	10.60	13.28	12.89	10.90	10.03	9.17	11.17	11.54
	SAH	GA	TSB	TSD	TSR	HA + IS2	SRH	DE	
50 × 10	3.57	6.20	3.25	3.66	5.37	6.84	2.01	16.96	
50 × 30	4.48	8.11	5.44	4.85	6.91	6.04	1.40	16.76	
50 × 50	4.63	8.08	5.04	5.11	6.99	5.44	1.50	16.57	
150 × 10	2.65	5.67	1.80	1.83	4.19	6.14	4.39	17.00	
150 × 30	2.57	8.28	4.84	4.84	6.86	3.96	2.11	17.00	
150 × 50	3.36	9.22	6.27	6.29	8.60	3.53	2.17	17.00	
250 × 10	2.45	5.75	1.81	1.80	2.48	5.18	4.40	17.00	
250 × 30	1.94	9.07	2.92	2.95	4.16	4.04	3.70	17.00	
250 × 50	2.01	10.05	5.33	5.33	6.86	3.08	2.97	17.00	
350 × 10	2.05	5.53	2.14	2.11	2.54	4.20	3.88	17.00	
350 × 30	1.91	7.88	2.35	2.35	2.85	3.64	4.13	17.00	
350 × 50	2.06	9.04	3.86	3.88	4.73	2.57	3.68	17.00	
Average	2.81	7.74	3.75	3.75	5.21	4.55	3.03	16.94	

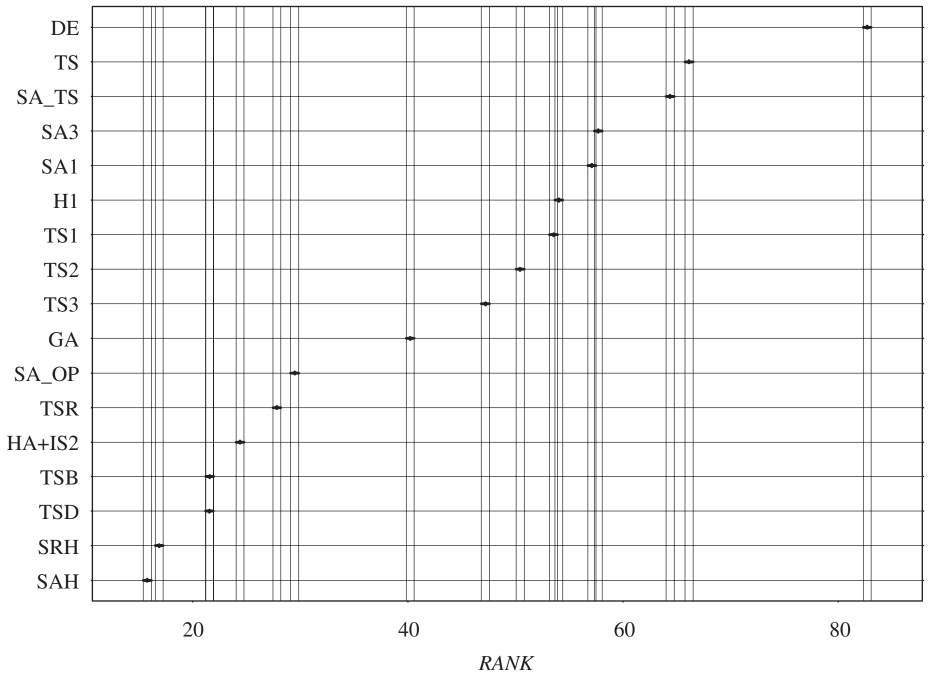

Fig. 5. Means plot for the metaheuristic algorithm factor and the average rank value (non-parametric test).

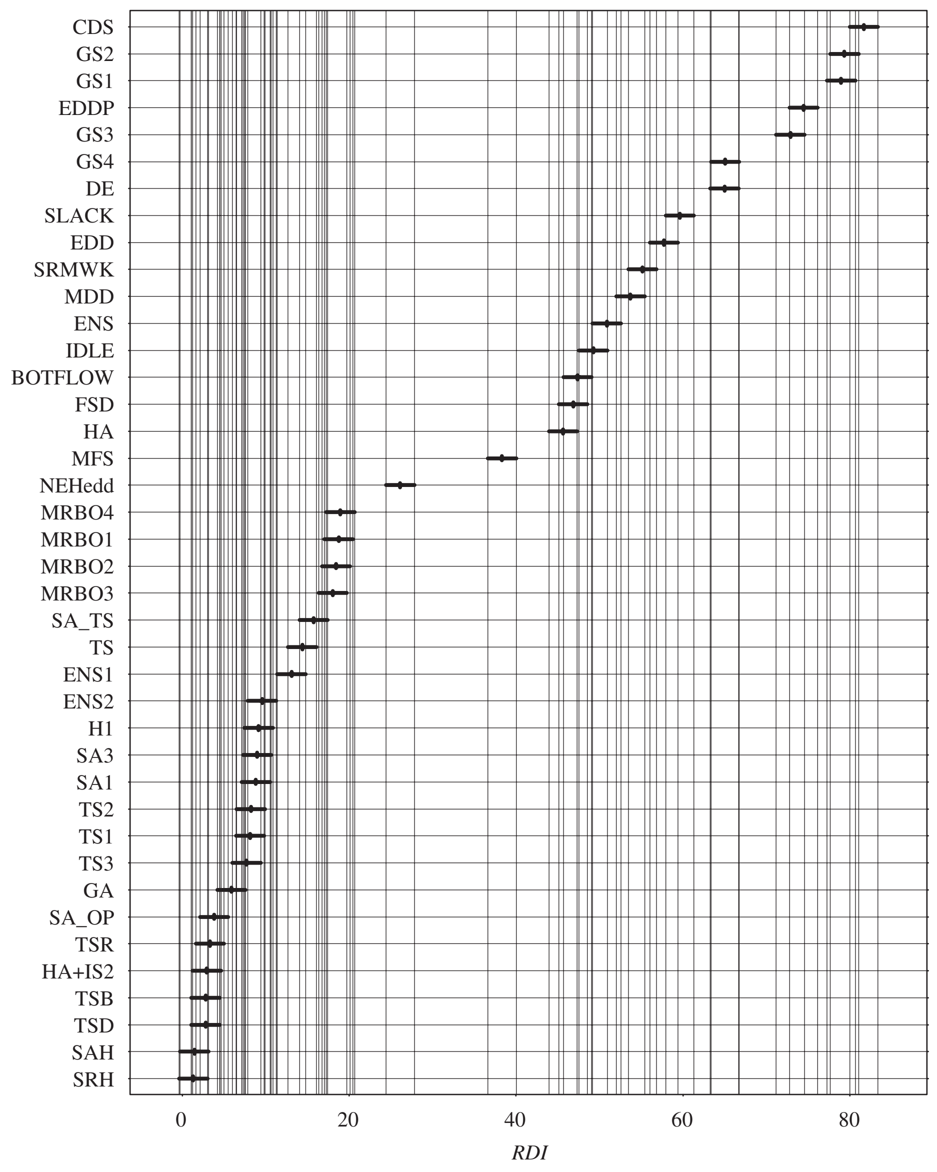

Fig. 6. Means plot and Tukey HSD intervals for the metaheuristic and heuristic algorithm factors and the relative deviation index (*RDI*) (ANOVA parametric test).

restricted neighbourhood is less effective than the tabu search presented by the same author in [27] despite the author's statements about being superior (the author used a different stopping criterion and benchmark from us). Finally, the method proposed in [36] which applies an improvement scheme twice shows good performance, specially when the number of jobs is greater than 50 with an average *RDI* of 0.082.

In order to check the effect of the initial starting sequence over the performance of the metaheuristic methods, we carried out some experiments starting from different initial sequences. We chose the differential evolution algorithm presented in [39] since, because of its poor performance, we should expect a good initialisation to improve results. We initialised the population with one individual provided by the $\mathrm{NEH}_{\mathrm{edd}}$ heuristic and the remaining individuals as in the original paper (random). The results show that this initial population improves the results obtained with the original version, where the whole population is obtained randomly, by about a 50%. While this is a good result, such improved algorithm is still statistically worse than SA_TS, which confirms our previous experiments in that the DE algorithm is

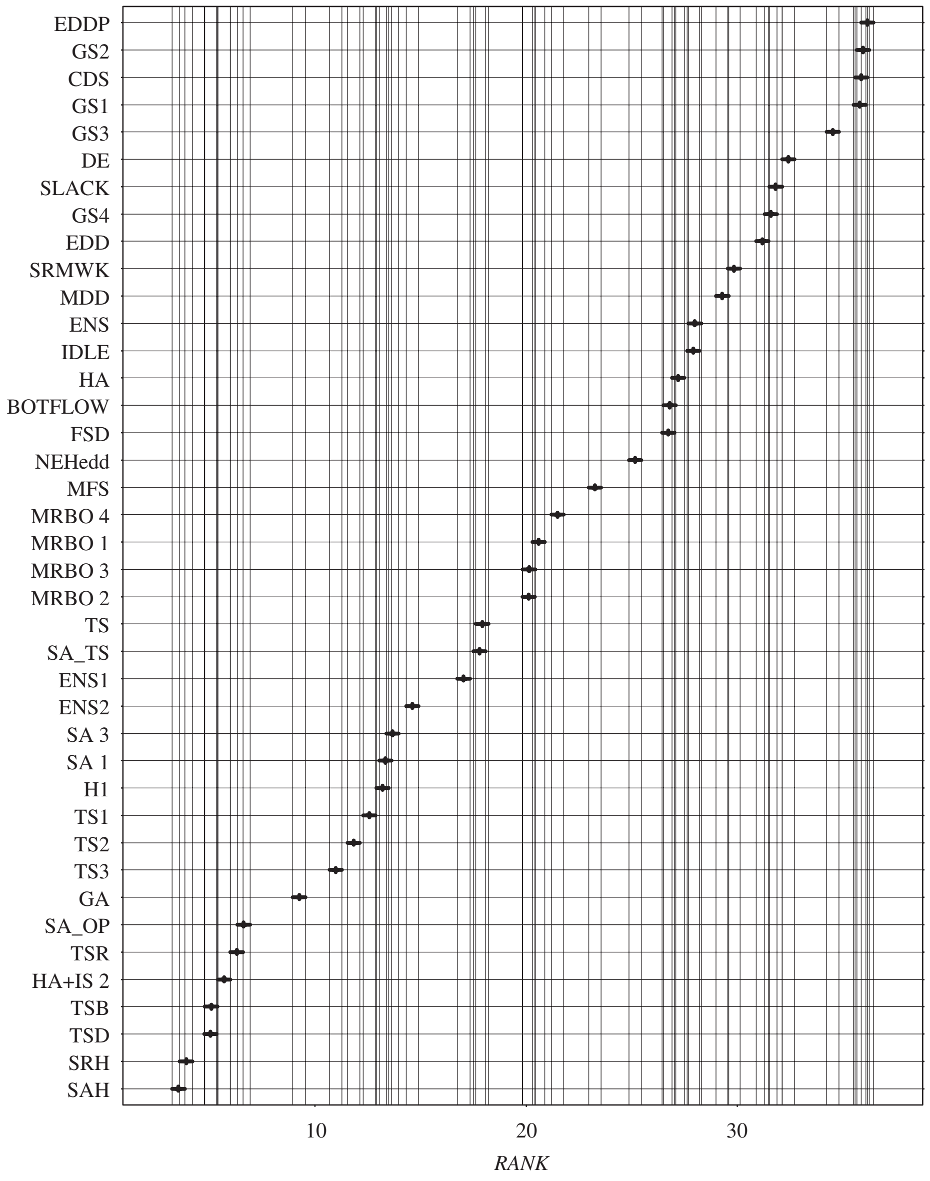

Fig. 7. Means plot for the metaheuristic and heuristic algorithm factors and the average rank value (non-parametric test).

not suited for this problem. We did a very similar experiment with the SRH simulated annealing proposed in [37] which is one of the best performing algorithms as our results have shown. In this case, we started from a random solution instead of the solution provided by the specific rule reported in the same paper. The outcome is a slight deterioration in solution quality of around 1%. Finally, we carried out an experiment in which we changed the starting initial solution of the TSB presented in [35]. In this case, the original algorithm started from the solution given by the MDD rule and we modified it using the solution provided by the NEH_{edd} heuristic as initial sequence. The results show that the performance of the algorithm improves by less than 1%. The conclusions from these experiments are clear; with a sufficiently long CPU time (as the one we have used), the choice of initialisation is not overly important. Furthermore, a good algorithm will yield good solutions regardless of the initialisation used. For poor performing algorithms, a better initialisation surely improves results, but, as has been shown, just the initialisation is not enough to obtain competitive results.

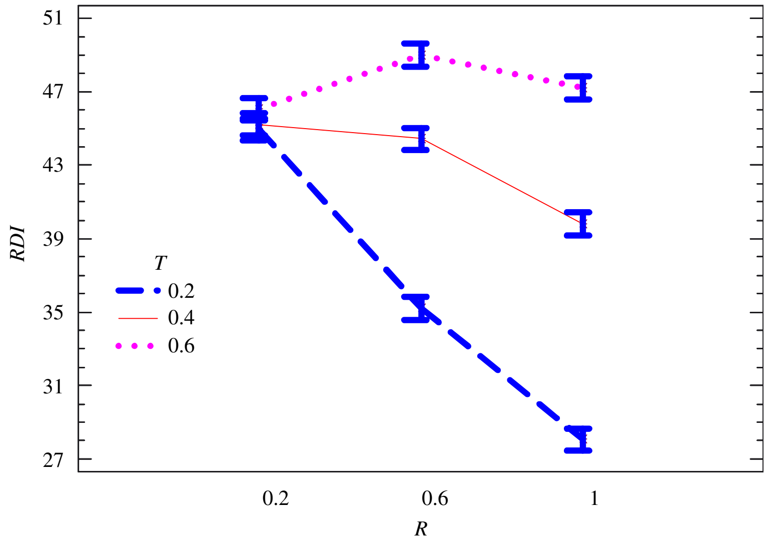

Fig. 8. Interaction plot for the tardiness factor (T) and the due date range factor (R) for the heuristic methods.

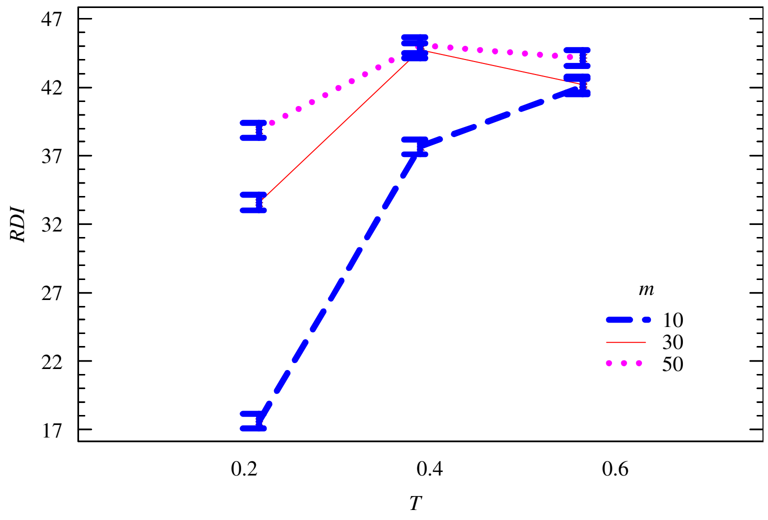

Fig. 9. Interaction plot for the tardiness factor (T) and the number of machines (m) for the metaheuristic methods.

In order to analyse the effectiveness of both heuristics and metaheuristics as a whole, we jointly evaluate all the methods in the same way as before. First, we applied an ANOVA analysis computing the *RDI* value taking into account all the methods. It is important to remark that the results of the five runs of the metaheuristic methods have been averaged in order to obtain a balanced analysis (note that the heuristic methods are run only once). The results can be seen in the means plot with Tukey intervals of Fig. 6. We applied a rank-based test and all methods are ranged from 1 to 40 depending on the *RDI* obtained. We can see the means plot of the rank analysis in Fig. 7. We can expect from the results that the metaheuristics outperform all the heuristics. However, there are three metaheuristic methods which perform worse than some heuristics, those which were the worst methods in the independent analysis of the metaheuristics. The differential evolution proposed in [33] shows a very poor performance and it is outperformed even by some despatching rules. The hybrid algorithm reported in [29] and the tabu search presented in [20] are outperformed only by the two best heuristic methods ENS1 and ENS2. The remaining metaheuristic methods show a better performance than the heuristic algorithms.

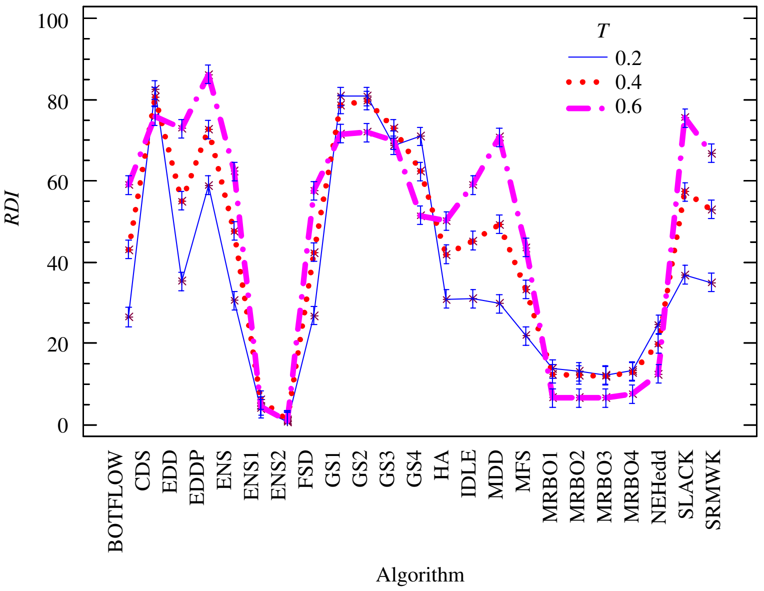

Fig. 10. Interaction plot for the tardiness factor (T) and the algorithm factor (R) for the heuristic methods.

Finally, we are going to analyse the effect of the number of jobs (n), number of machines (m), tardiness factor (T) and due date range (R) in the effectiveness of all the algorithms. An experimental design where n, m, T and R are also considered as factors was carried out. After the analysis, all the factors become statistically significant in both heuristics and metaheuristics. It is also interesting to note that all the different interactions of order two are statistically significant. In Fig. 8 we can see the effect that the *tardiness factor* (T) has over the *due date range* (R) in the heuristic methods. The same profile for this interaction is also obtained in the metaheuristic algorithms. It is clear that the most difficult instances are those with $T = 0.6$ and when the value of R is increased the instances are easier to solve, specially for $T = 0.2$. In Fig. 9 it is shown the effect that the *tardiness factor* has over the number of machines (m) in the metaheuristic algorithms. We can observe that the highest values of T lead to the worst results regardless of the number of machines. When the value of T is decreased, the hardest instances are those with the highest number of machines. Similar plots can be obtained for the n factor. We can also study the effect of the different factors over the algorithm. In Fig. 10 we can see the effect that the *tardiness factor* has over the heuristic algorithms. We can observe that the best methods ENS1 and ENS2 show the best performance regardless of the *tardiness factor* value. The same profile is obtained for the *due date range* factor in both heuristics and metaheuristics.

6. Conclusions

In this paper we have given an extensive and comprehensive review of heuristic and metaheuristic methods for the permutation flowshop scheduling problem with the objective of minimising the total tardiness. We have coded and evaluated 40 algorithms in order to test their performance under the same conditions. Moreover, a benchmark of instances is proposed to evaluate all the methods under a common data set of 540 problems of up to 350 jobs and 50 machines. This benchmark, along with the best solutions for each instance, are available from http://www.upv.es/gio/rruiz and also upon request from the authors. Different statistical tests are applied to the results in order to analyse the observed differences among the methods.

From the heuristic algorithms we can conclude that those which are based on the insertion or interchange of jobs show the best performance. We can also say that these methods perform good in terms of efficiency taking less than 8 s of CPU time on average. The MRBO methods presented in [21] perform very well but need more CPU time. An interesting result is that obtained with the heuristic method presented in [34] since this algorithm was proposed originally for the makespan objective and is often adapted to the tardiness objective despite its poor performance, in fact, it is the worst heuristic of the comparison. It is also interesting that the four heuristic methods presented in [22] showed a poor performance, specially the second one which is frequently used as a benchmark in many of the papers reviewed.

From the metaheuristics tested, the two simulated annealing algorithms proposed by [30,37] outperform all the other methods evaluated. We have also shown that the tabu search methods are good metaheuristics for this problem, mainly those presented in [35]. We have also proved that results obtained with the recent differential evolution reported in [39] are poor and outperformed even by some despatching rules.

From the computational and statistical analyses, it seems that simple adaptations of methods proposed for other objectives, mainly makespan, results in good performance for total tardiness. Therefore, an interesting line of research would be to compare a set of adapted methods against the ones reviewed here and that were specifically proposed for the total tardiness objective.

Acknowledgements

This work is partly funded by the Spanish Department of Science and Technology (research project ref. DPI2004-06366-C03-01). The authors are indebted to Dr. Marco Chiarandini for his help and the R code provided which was used in the non-parametric statistical analyses in this paper.

References

[1] Pinedo M. Scheduling: theory, algorithms and systems. 2nd ed., Englewood Cliffs, NJ: Prentice-Hall; 2002.
[2] Rinnooy Kan AHG. Machine scheduling problems: classification, complexity and computations. The Hague: Martinus Nijhoff; 1976.
[3] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 2005;165:479–94.
[4] Du J, Leung JYT. Minimizing total tardiness on one machine is NP-hard. Operations Research 1990;38:22–36.
[5] Sen T, Gupta SK. A state-of-art survey of static scheduling research involving due dates. OMEGA, The International Journal of Management Science 1984;12:63–76.
[6] Koulamas K. The total tardiness problem: review and extensions. Operations Research 1994;42:1025–41.
[7] Huegler PA, Vasko FJ. A performance comparison of heuristics for the total weighted tardiness problem. Computers and Industrial Engineering 1997;32:753–67.
[8] Sen T, Sulek JM, Dileepan P. Static scheduling research to minimize weighted and unweighted tardiness: a state-of-the-art survey. International Journal of Production Economics 2003;83:1–12.
[9] Sen T, Dileepan P, Gupta JND. The two-machine flowshop scheduling problem with total tardiness. Computers and Operations Research 1989;16:333–40.
[10] Potts CN, Van Wassenhove LN. A decomposition algorithm for the single machine total tardiness problem. Operations Research Letters 1982;1:177–81.
[11] Kim YD. A new branch and bound algorithm for minimizing mean tardiness in 2-machine flowshops. Computers and Operations Research 1993;20:391–401.
[12] Pan JCH, Fan ET. Two-machine flowshop scheduling to minimize total tardiness. International Journal of Systems Science 1997;28:405–14.
[13] Pan JCH, Chen JS, Chao CM. Minimizing tardiness in a two-machine flow-shop. Computers and Operations Research 2002;29:869–85.
[14] Schaller J. Note on minimizing total tardiness in a two-machine flowshop. Computers and Operations Research 2005;32:3273–81.
[15] Kim YD. Minimizing total tardiness in permutation flowshops. European Journal of Operational Research 1995;85:541–55.
[16] Chung CS, Flynn J, Kirca O. A branch and bound algorithm to minimize the total tardiness for *m*-machine permutation flowshop problems. European Journal of Operational Research 2006;174:1–10.
[17] Panwalkar SS, Iskander W. A survey of scheduling rules. Operations Research 1977;25:45–61.
[18] Blackstone Jr JH, Phillips DT, Hogg GL. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. International Journal of Production Research 1982;20:27–45.
[19] Ow PS. Focused scheduling in proportionate flowshops. Management Science 1985;31:852–69.
[20] Kim YD. Heuristics for flowshop scheduling problems minimizing mean tardiness. Journal of Operational Research Society 1993;44:19–28.
[21] Kim YD, Lim HG, Park MW. Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process. European Journal of Operational Research 1996;91:124–43.
[22] Gelders LF, Sambandam N. Four simple heuristics for scheduling a flow-shop. International Journal of Production Research 1978;16:221–31.
[23] Nawaz M, Enscore Jr. EE, Ham I. A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem. OMEGA, The International Journal of Management Science 1983;11:91–5.
[24] Raman N. Minimum tardiness scheduling in flow shops: construction and evaluation of alternative solution approaches. Journal of Operations Management 1995;85:131–51.
[25] Morton TE, Pentico DW. Heuristic scheduling systems with applications to production systems and project management. New York: Wiley; 1993.
[26] Adams J, Balas E, Zawack D. The shifting bottleneck procedure in job shop scheduling. Management Science 1988;34:391–401.

[27] Adenso-Díaz B. Restricted neighbourhood in the tabu search for the flowshop problem. European Journal of Operational Research 1992;62: 27–37.

[28] Widmer M, Hertz A. A new heuristic method for the flow shop scheduling problem. European Journal of Operations Research 1989;41: 186–93.

[29] Adenso-Díaz B. An SA/TS mixture algorithm for the scheduling tardiness problem. European Journal of Operational Research 1996;88: 516–24.

[30] Parthasarathy S, Rajendran C. A simulated annealing heuristic for scheduling to minimize mean weighted tardiness in a flowshop with sequence-dependent setup times of jobs—a case study. Production Planning and Control 1997;8:475–83.

[31] Parthasarathy S, Rajendran C. An experimental evaluation of heuristics for scheduling in a real-life flowshop with sequence-dependent setup times of jobs. International Journal of Production Economics 1997;49:255–63.

[32] Parthasarathy S, Rajendran C. Scheduling to minimize mean tardiness and weighted mean tardiness in flowshop and flowline-based manufacturing cell. Computers and Industrial Engineering 1998;34:531–46.

[33] Onwubolu GC, Mutingi M. Genetic algorithm for minimizing tardiness in flow-shop scheduling. Production Planning and Control 1999;10: 462–71.

[34] Campbell HG, Dudek RA, Smith ML. A heuristic algorithm for the n job m machine sequencing problem. Management Science 1970;16: B-630–7.

[35] Armentano VA, Ronconi DP. Tabu search for total tardiness minimization in flow-shop scheduling problems. Computers and Operations Research 1999;26:219–35.

[36] Rajendran C, Ziegler H. Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times. European Journal of Operational Research 2003;149:513–22.

[37] Hasija S, Rajendran C. Scheduling in flowshops to minimize total tardiness of jobs. International Journal of Production Research 2004;42: 2289–301.

[38] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64:278–85.

[39] Onwubolu G, Davendra D. Scheduling flow shops using differential evolution algorithm. European Journal of Operational Research 2006;171:674–92.

[40] Osman I, Potts C. Simulated annealing for permutation flow-shop scheduling. OMEGA, The International Journal of Management Science 1989;17:551–7.

[41] Zemel E. Measuring the quality of approximate solutions to zero–one programming problems. Mathematics of Operations Research 1981;6: 319–32.

[42] Montgomery DC. Design and analysis of experiments. 5th ed., New York: Wiley; 2000.

[43] Chiarandini M. Stochastic local search methods for highly constrained combinatorial optimisation problems. PhD thesis, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany; 2005.

[44] Friedman M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association 1937;32:675–701.

[45] Conover W. Practical nonparametric statistics. 3rd ed., New York: Wiley; 1999.