Omega 38 (2010) 57 – 67

Contents lists available at ScienceDirect

Omega

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

Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem☆

Eva Vallada*, Rubén Ruiz

Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Ciudad Politécnica de la Innovación, Edificio 8G, Acc. B. Camino de Vera S/N, 46021 Valencia, Spain

ARTICLE INFO

Article history:
Received 17 December 2007
Accepted 15 April 2009
Available online 5 May 2009

Keywords:
Flowshop
Tardiness
Genetic algorithm
Path relinking
Diversity

ABSTRACT

In this work three genetic algorithms are presented for the permutation flowshop scheduling problem with total tardiness minimisation criterion. The algorithms include advanced techniques like path relinking, local search and a procedure to control the diversity of the population. We also include a speed up procedure in order to reduce the computational effort needed for the local search technique, which results in large CPU time savings. A complete calibration of the different parameters and operators of the proposed algorithms by means of a design of experiments approach is also given. We carry out a comparative evaluation with the best methods that can be found in the literature for the total tardiness objective, and with adaptations of other state-of-the-art methods originally proposed for other objectives, mainly makespan. All the methods have been implemented with and without the speed up procedure in order to test its effect. The results show that the proposed algorithms are very effective, outperforming the remaining methods of the comparison by a considerable margin.

1. Introduction

The permutation flowshop scheduling problem (PFSP) is a widely studied combinatorial optimisation problem where there is a set $N=\{1,\ldots,n\}$ of n jobs that have to be processed on a set $M=\{1,\ldots,m\}$ of m machines. In the PFSP all jobs visit the machines in the same order which can be assumed to be $1,2,\ldots,m$. Therefore, each job is made up of m tasks. Furthermore, in the PFSP once the job sequence is established, it is not changed from machine to machine. This yields $n!$ job permutations and possible processing sequences. Each operation requires a given processing time denoted by p_{ij}. The most commonly studied objective in the literature is to find a minimum completion time sequence, a criteria that is known as makespan or C_{max}. According to Pinedo [1] this problem is denoted as $F/prmu/C_{max}$ and the associated decision problem was shown to be $\mathcal{NP}$-complete by Rinooy Kan [2] for $m \geq 3$.

In the literature, there is a large body of papers dealing with the PFSP and the makespan objective. Some of the most recent methods proposed for this problem are those presented by Ruiz et al. [3], Grabowski and Pempera [4] and Farahmand et al. [5]. We can find in Ruiz and Maroto [6] an updated extensive comparison and evaluation. Other recent reviews centred around makespan criterion can be found in Framinan et al. [7] or Hejazi and Saghafian [8]. Recent research has also focused on other objectives, especially those related to due dates. These objectives are important in real-life, mainly in industry where the fulfillment of due dates agreed with customers is of uttermost importance. Among due date based criteria, the minimisation of the total tardiness is probably the most common one. Therefore, the PFSP with this criterion is denoted as $F/prmu/\sum T_j$ [1] where $T_j = \max\{C_j - d_j, 0\}$ is the *tardiness* of job j, being d_j its *due date* and C_j its completion time at the last machine of the shop. Du and Leung [9] showed that the problem is $\mathcal{NP}$-hard in the ordinary sense even when there is only one machine and $\mathcal{NP}$-hard in the strong sense for $m \geq 2$.

Recently, Vallada et al. [10] reported an exhaustive review and comparative evaluation of different methods for the PFSP minimising total tardiness. In this review, it was shown that it is not common to adapt recent, high performing methods from the makespan to the total tardiness objective. As it was shown, only a few classic algorithms for the makespan objective are usually adapted to the total tardiness objective [11–13].

Given that the makespan objective is the most studied, it seems plausible to take advantage of this research when developing new algorithms for total tardiness minimisation. Like for example the advanced genetic algorithms (GAs) depicted by Ruiz et al. [3] for the makespan criterion. Working with such methods for other objectives seems promising. Furthermore, genetic algorithms have been

☆This article was processed by Associate Editor Lev.

* 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).

0305-0483/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.omega.2009.04.002

sparsely used for the total tardiness objective. Therefore, the main goal of this paper is to propose high performing genetic algorithms for the PFSP minimising total tardiness. These algorithms make extensive use of advanced techniques like local search, diversification and path relinking (PR). The resulting algorithms show excellent performance when compared against recent proposed methods under careful and comprehensive computational and statistical experiments.

The remainder of this paper is organised as follows. In Section 2 we review the literature on this problem. In Section 3 we describe in detail the genetic algorithms proposed. In Section 4, a design of experiments (DOE) approach is applied in order to calibrate the genetic algorithms. Results of a comparative computational and statistical evaluation are reported in Section 5. Finally, conclusions are given in Section 6.

2. Literature review

Vallada et al. [10] provided an extensive literature review along with a comparative evaluation about the PFSP with the objective of total tardiness minimisation. In this section we review the most important existing methods, from the classical exact approaches to the most recent and effective metaheuristic algorithms.

Exact methods for the PFSP minimising total tardiness are mainly focused in the two machine case. There exist several papers where branch and bound algorithms for this case are proposed [14–18]. The best performing algorithms among those presented in the aforementioned papers are able to solve instances of up to 18 or 20 jobs maximum. Regarding the m-machine case, we only find two papers, Kim [19] and more recently, Chung et al. [20]. The former solved all the instances of up to 13 jobs and eight machines and some instances of up to 14 jobs and four machines. The latter solved optimally all instances up to 15 jobs and two machines and some with 20 jobs and eight machines. As we can see, only the most recent exact methods are able to solve some problems of up to 20 jobs and eight machines. For this reason, research is mainly focused on the development of heuristic and metaheuristic methods to tackle this complex problem.

Some of the earlier heuristics are the four proposed by Gelders and Sambandam [12] based on priority rules. A few years later we find the paper by Ow [21] where a heuristic based on a priority rule is proposed for the proportionate flowshop problem. In [22] some adaptations of algorithms for different objectives, mainly the makespan, are presented. We want to remark the NEH method [13], which is considered the best heuristic for the makespan objective, according to Ruiz and Maroto [6] and Kalczynski and Kamburowski [23]. In the adapted version of Kim [22], jobs are initially sorted following the earliest due date (EDD) rule, that is, in non-decreasing order of due dates and according to the author we name this NEH version as NEH_{edd}. We can also find in [24] several rules and heuristics originally proposed for the one machine and two machine cases and adapted to the m-machine case. Finally, Kim et al. [25] reported several improvement heuristics which start from the NEH_{edd} method [22] and local search procedures based on insertion, interchange and permutation of jobs are applied to improve the solution.

With respect to the metaheuristic methods, some of the earliest existing papers are Adenso-Díaz [26] and Kim [22]. In both papers we find adapted versions of the well-known tabu search by Widmer and Hertz [27] originally proposed for the makespan objective. The former starts from the solution provided by Ow [21] and a restricted neighbourhood is applied. The latter starts from the *earliest due date* rule. Later, the same authors extended these results in [28,25]. In this case, Adenso-Díaz [28] proposed a hybrid algorithm based on simulated annealing and tabu search starting from the solution provided by Ow [21]. Kim et al. [25] devised four tabu search and four simulated annealing methods all of them starting from the NEH_{edd} method. In two related papers [29,30], two very similar simulated annealing methods for the flowshop problem with sequence dependent setup times and the objective to minimise the mean weighted tardiness are proposed. Another two simulated annealing algorithms are those reported by the same authors in [31] which start from a specific rule and perturbation schemes are applied to improve the initial solution. In [32], a genetic algorithm is proposed which considered three objective functions: minimising total tardiness, minimising number of tardy jobs and minimising both objectives at the same time. In [33], a basic tabu search was proposed and then diversification, intensification and restricted neighbourhood strategies are applied to form four tabu search variations. Another work is that by Rajendran and Ziegler [34] where heuristic and metaheuristic methods are proposed with the objective of minimising the sum of weighted flowtime and weighted tardiness. A simulated annealing method can be found in Hasija and Rajendran [35] which starts from the solution provided by Parthasarathy and Rajendran [31] and then applies a local search procedure to improve it. After this, the simulated annealing method itself is applied to improve this initial solution and two local search procedures are considered, the first one is very similar to that presented in [31] and the second one is based on the interchange of jobs. This simulated annealing shows the best performance for this problem up to the moment according to the results of Vallada et al. [10]. Finally, in a recent work, Onwubolu and Davendra [36] proposed a differential evolution algorithm to minimise the makespan, flowtime and total tardiness and Ronconi and Henriques [37] presented heuristics for the flowshop problem with blocking.

It is important to point out that most of the aforementioned published papers only compare results with a few classical algorithms for the same problem and the experiments were carried out using different benchmarks of instances of up to 100 jobs and 20 machines maximum in the best case. For more details about the performance of all these methods see [10].

3. Proposed genetic algorithms

Genetic algorithms are bio-inspired optimisation methods that are widely used to solve combinatorial problems such as the PFSP. There is a rich literature where GAs are successfully applied to this problem (see for example [3,38–42], among others). However, as we have mentioned in the previous section, despite showing a very good performance for the makespan objective, only one paper can be found in the literature for the total tardiness objective [32]. In our proposed algorithms, among other innovative features, we add path relinking techniques and a procedure to control the diversity of the population. We have studied all these characteristics in a series of calibration experiments in order to set the best operators and parameter values. In the next subsections a detailed description about the algorithms is reported.

3.1. Representation of solutions, initialisation of the population and selection operator

The most commonly used solution representation for the PFSP is a simple permutation of jobs that indicates the processing order by the machines. With this representation it is easy to construct an active schedule by sequencing the first job of the permutation on all m machines, then the second, and so on until all n jobs are scheduled. The GAs are formed by a population of P_{size} individuals or n job permutations.

It is also common to randomly generate the initial population in a genetic algorithm. However, a recent trend consists in including in the population some good individuals provided by some effective

heuristics, mainly the NEH heuristic [13] or by some despatching rules. This approach ensures a faster convergence to good solutions. Such "seeded" GAs are very common and can be found in most papers related to PFSP and GAs. In our case, we propose two initialisation schemes, the first one randomly generates all the individuals except one which is given by the EDD rule. The second initialisation scheme consists of seeding the population with two good individuals provided by the NEH_{edd} heuristic [22] and by the EDD despatching rule. The remaining individuals are randomly generated.

Regarding the selection mechanism, in the classical genetic algorithms, tournament and ranking-like selection operators are common. Such operators either require fitness and mapping calculations or the population to be continuously sorted. In this work, a much simpler and faster selection scheme, called *n*-tournament, is used. In this case, according to a parameter called "*pressure*", a given percentage of the population is randomly selected. The individual with the lowest total tardiness value among the randomly selected percentage of individuals wins the tournament and is finally selected. This results in a very fast GA and selection operator since no fitness calculation and/or mapping is needed, and the population does not need to be sorted. Finally, we can easily vary the selection pressure in the GA by increasing or decreasing the "*pressure*" parameter.

3.2. Crossover, mutation and generational scheme

There are several crossover operators proposed in the literature for scheduling problems. In general, the goal of the crossover operator is to generate two good individuals, called offspring, from the two selected progenitors. In [3] we can find an extensive review of eight crossover operators suitable for the PFSP. After statistical experimentation, the authors stated that the two best crossover operators are those proposed in the same paper: similar block order crossover (SBOX) and similar block 2-point order crossover (SB2OX) with a very similar performance. The one point (OP) order crossover showed a good performance as well. In this work, we pick for experimentation the SBOX and OP crossover operators which are applied with a P_c probability to two distinct individuals selected by *n*-tournament, resulting in two offspring. Regarding the mutation operator, the most common and best performing for the PFSP with the objective to minimise makespan is the shift mutation where each job in the permutation is extracted with a P_m probability and inserted in a random different position.

Another aspect to consider is the way the generated offsprings after selection, crossover and mutation are inserted in the population. This is usually known as generational scheme. It is usual that offspring directly replace the parents. In other GAs, this procedure is carried out only after having preserved the best individuals from the last generation in order to avoid loosing the best solutions (elitist GAs). Another approach is the so called steady state GAs where the offspring do not replace the parents but different individuals from the population. In the genetic algorithms proposed in this work, the offspring are accepted into the population only if they are better than the worst individuals of the population and if at the same time are unique, i.e., there are no other identical individuals already in the population. Otherwise they are rejected. As a result, population steadily evolves to better average tardiness values while at the same time it contains different solutions, which help in maintaining diversity and in avoiding premature convergence to a dominant, sub-optimal individual.

3.3. Local search

Local search procedures to improve solutions are widely used in genetic algorithms just as in other metaheuristic approaches for the PFSP regardless of the objective to minimise. This is specially important in GAs for scheduling where genetic operators are unable to carry out the fine improvement that a simple local search method can do. We test a local search based on job insertion (insertion neighbourhood) since it is the most used in flowshop problems (see [3,43], among many others). In the proposed local search, a job is removed from the sequence and inserted in all possible *n* positions. The job is finally placed at the position that results in the lowest total tardiness value. We call this a step or iteration of the local search procedure.

In order to reduce the computational effort it is possible to introduce a simple speed up. When inserting a removed job in a given position, one has to calculate the total tardiness of the sequence. This calculation has a computational complexity of $\mathcal{O}(nm)$. Therefore, a single step of the local search has a computational complexity of $\mathcal{O}(n^2m)$. However, if the insertion in all *n* positions is done in order, substantial savings can be obtained. For example, let us picture a permutation $\pi = \{1,2,3,4,5\}$ of five jobs. If we extract the job 4, the "incomplete" permutation would be $\pi_i = \{1,2,3,5\}$ and the job 4 is now tested in the first, second, third, fourth and fifth positions, i.e., the total tardiness for the following permutations needs to be calculated: $\pi_1 = \{4,1,2,3,5\}$, $\pi_2 = \{1,4,2,3,5\}$, $\pi_3 = \{1,2,4,3,5\}$, $\pi_4 = \{1,2,3,4,5\}$, $\pi_5 = \{1,2,3,5,4\}$. First, π_4 does not need to be calculated if the tardiness for the original π was known. Also, notice that job 1 is in the same position for π_2, π_3, π_4 and π_5. Similarly, job 2 is in the same position for π_3, π_4 and π_5. This means that we need to calculate the tardiness of job 1 at the first position in π_2 only once and with proper bookkeeping, savings can be obtained in subsequent permutations. In π_2 job 1 is already calculated and only job 2 needs to be sequenced. In π_3 jobs 1 and 2 are already set and so on. Unfortunately, the worstcase computational complexity is maintained with the speed up. However, a careful analysis results in a computational complexity of $\mathcal{O}(n^2m - ((n^2 - 3n + 2)/2)m)$ for the speed up which confirms the expected large savings in CPU time. The above local search is carried out in all *n* jobs of each generated offspring after crossover and mutation with a probability P_{ls}. It is also applied to the best individual after the initialisation of the population. We test the algorithms with and without the speed up procedure in order to study the effect of this improvement method.

3.4. Diversity of the population and restart mechanism

Ideally, a diverse population is more likely to evolve whereas when the diversity of the population falls, it means that the individuals are very similar and the algorithm is or will soon be stalled. Similarly, after a number of generations, the best solution in a GA will cease to improve. If this is detected, one can apply some restart mechanism in order to move away from the current population in which the GA is stuck (see [3]), hoping to find better solutions in the long run. However, these two problems (loosing diversity and getting stalled), although related, are not the same. One can prematurely converge to not so good solutions by rapidly loosing diversity. Differently, it is possible for a GA to get stalled after many generations with a sufficiently diverse population. In this latter case probably the GA has found a near optimum solution.

In this work, we deal with both problems. We propose to compute a diversity value based on the number of times that jobs appear at the different positions of the solutions that form the population. Ultimately, and as we will detail, we obtain a value between zero and one so that a value close to one indicates a very diverse population where each job occupies different positions among the individuals. A small value indicates that all individuals (although different, according to the generational scheme depicted in Section 3.2) are very similar which indicates a degenerated population with low diversity. When the diversity value falls below a given

threshold value, a restart mechanism is applied where all or part of the population is regenerated. We test three restart mechanisms: (1) regenerate the whole population randomly except for two individuals provided by the NEH heuristic and the EDD rule, (2) regenerate all the population randomly and (3) keep the 20% best individuals from the current population and regenerate the remaining 80% at random.

There are several diversity measures that can be found in the literature. The most simple and usual is the sum of the Hamming distances between all possible pairs. In our case, this is the sum of all positions where two given solutions have different jobs. To obtain the diversity value from the individuals of the population we compute two matrices called *GeneCount* and *GeneFrecuency* according to Wineberg and Oppacher [44] in the following way:

- The *GeneCount* matrix is the number of times that a job α appears at a given position k across the population

$$c_k(\alpha) = \sum_{i=1}^{P_{size}} \sum_{j=1}^{n} \delta_{i,j}(\alpha), \quad \alpha, k = 1, \ldots, n \tag{1}$$

  where $\delta_{i,k}$ is a Kronecker δ such that becomes 1 if the job at position j of individual i is the job α and at the same time $j = k$ or 0 otherwise.
- The *GeneFrecuency* matrix is the ratio of the job count to the size of the population

$$f_k(\alpha) = \frac{c_k(\alpha)}{P_{size}}, \quad \alpha, k = 1, \ldots, n \tag{2}$$

- The diversity value of the population (*Div*) is then computed in the following way:

$$Div = \frac{1}{n-1} \sum_{k=1}^{n} \sum_{\alpha=1}^{n} f_k(\alpha)(1 - f_k(\alpha)) \tag{3}$$

  Hence, *Div* gives us the diversity measure between 0 and 1.

For example, let us picture a population P formed by three individuals (permutations) of four jobs: $\pi_1 = \{1, 2, 3, 4\}$, $\pi_2 = \{2, 3, 4, 1\}$, $\pi_3 = \{1, 4, 2, 3\}$. First, we compute the *GeneCount* matrix following Eq. (1):

Position (k)	Job (α)			
	1	2	3	4
1	2	1	0	0
2	0	1	1	1
3	0	1	1	1
4	1	0	1	1

Then, we compute the *GeneFrecuency* matrix following expression (2), which is simply obtained by dividing each cell of the *GeneCount* matrix by P_{size} (3 in our example). Finally, we compute the diversity value using Eq. (3) obtaining a value of 0.815.

3.5. Path relinking

Path relinking is a search technique originally proposed by Glover and Laguna [45] where the objective is to explore the search space or "path" between a given set (usually two) of good solutions. The objective is to generate a set of new solutions in between the good solutions from the set. Path relinking is frequently used in greedy randomised adaptive search procedure (GRASP) algorithms, (see [46–48]). Regarding GAs, we find several papers where the path relinking technique is applied, like for example [42,49–51].

Table 1
Interchange movements of the *path relinking* to transform π_1 into π_2.

Movements	Permutation
	(1,2,5,3,4) $= \pi_1$
(1,1,4)	(3,2,5,1,4)
(2,2,1)	(2,3,5,1,4)
(5,3,5)	(2,3,4,1,5) $= \pi_2$

Most of them were proposed for the makespan objective and used path relinking technique as local search procedure between two good solutions. In this work, a path relinking technique is applied in order to explore the search space between two individuals of the population. We obtain all the job interchange movements that are necessary to transform one selected solution, called "origin" into another one, called "destination". In this way, path relinking based on interchange movements complements the local search procedure which is based on the insertion neighbourhood. Each time a movement is carried out, the obtained solution looks less like origin and more like destination. For each intermediate solution, the tardiness value is obtained and at the end the intermediate solution with the lowest tardiness value among all the movements is returned. Note that the set of movements is not symmetric, that is, the movements from individuals A to B are not the same that those from individuals B to A. Therefore, given two selected individuals from the population, when we apply the path relinking we obtain two new solutions, the best intermediate solution in each direction. For example, let us picture two permutations of five jobs: $\pi_1 = \{1, 2, 5, 3, 4\}$, $\pi_2 = \{2, 3, 4, 1, 5\}$. We define the set of interchange movements to transform π_1 into π_2 like for example (1,1,4), which means that the job 1 placed on position 1 of π_1 will be placed on position 4 of π_1, which is the position of the job 1 in π_2. At the same time, job 3 in π_1 (which is in position 4 of π_1) will be placed in position 1 (interchange with job 1 in π_1). The set of movements to transform π_1 into π_2 are showed in Table 1.

With respect to the selection of the two individuals to carry out the path relinking, we can select them from the current population or we can select two individuals from a pool of elite solutions. In the latter case, we maintain a separated pool of $0.4P_{size}$ individuals and after each generation, the best solution replaces the worst individual from the pool if it is better than this worst individual and also if it is not already in the pool. Note that in the first iteration of the GAs, the elitist pool is formed by the best 40% of the individuals of the current population.

Different approaches are proposed. We test the path relinking applied either as a crossover operator or after a number of generations without improvement in the best solution. The selected individuals are marked in order to not be selected again for the path relinking. We propose four different versions of the technique where the path relinking is applied:

1. between two individuals of the current population. In this case the n-tournament selection is carried out, that is, two different unmarked individuals are selected according to a *pressure* value,
2. between the two best unmarked individuals of the elite pool,
3. between two randomly selected unmarked individuals from the elite pool, and
4. between the best and a randomly selected individual. Both unmarked and from the elite pool.

We test the four approaches in both cases; path relinking as crossover and path relinking applied after a number of iterations without improving the best solution found so far. Notice that the procedure is not carried out if there are less than two unmarked individuals available.

4. Calibration of the algorithms

We propose and calibrate three GAs. In all three we apply the restart mechanism when the population diversity falls below a given threshold value as well as the steady state generational scheme. The first proposed algorithm, referred to as GAPR, substitutes the crossover operator in favor of the path relinking technique. The second algorithm, referred to as GAPR2, is slightly more complex since it maintains the crossover operator and applies the path relinking technique as an additional step whenever the best solution of the population has not improved over a given number of generations. Lastly, the third proposed GA, refereed to as GADV, does without the path relinking technique entirely as only applies the restart mechanism.

We calibrate all three proposed genetic algorithms taking into account the different choices for the operators and values for the parameters. Each algorithm is calibrated separately and we make extensive use of the design of experiments approach. A full factorial design is employed.

For the calibration of the GAPR, the following combinations of factors are tested:

- Population size (P_{size}): two levels (30 and 50).
- Population initialisation (*Inipop*): two levels (Randomly + EDD individual; Randomly + NEH_{edd} individual + EDD individual).
- Diversity threshold (*Div*): three levels (0.2, 0.4 and 0.6).
- Restart mechanism (*Restart*): three levels according to Section 3.4 (1–3).
- Path relinking: four levels according to Section 3.5 (1–4).

We obtain a total of $2 \times 2 \times 3 \times 3 \times 4 = 144$ different combinations, that is, 144 different configurations for GAPR. For the GAPR2 calibration, the following combinations of factors are tested:

- Crossover type: two levels (OP and SBOX).
- Crossover probability: two levels (0.3 and 0.5).
- Diversity threshold (*Div*): two levels (0.2, 0.4).
- Restart mechanism (*Restart*): three levels (1–3).
- Path relinking: four levels (1–4).
- Number of iterations without improvement before path relinking is applied: three levels (25, 50 and 75).

The total number of GAPR2 configurations is therefore $2 \times 2 \times 2 \times 3 \times 4 \times 3 = 288$. In the last calibration for the GADV algorithm, we test the following combinations of factors:

- Population size (P_{size}): two levels (30 and 50).
- Crossover type: two levels (OP and SBOX).
- Crossover probability: two levels (0.3 and 0.5).
- Diversity threshold (*Div*): three levels (0.2, 0.4, 0,6).
- Restart mechanism (*Restart*): three levels (1–3).

For this last experiment, $2 \times 2 \times 2 \times 3 \times 3 = 72$ different configurations are tested.

Moreover, the following parameter values and operators are fixed in all the experiments:

- Selection type: *n*-tournament.
- Pressure for the selection: 30%.
- Mutation probability: 0.02.
- Mutation type: insertion.
- Probability to apply local search (P_{ls}): 0.15.

These parameters are fixed after short runs or small calibrations in order to keep the aforementioned calibrations at a manageable level.

Table 2
Operators and parameter values used for the genetic algorithms after calibration.

Factor	GAPR	GAPR2	GADV
P_{size}	30	30	30
Inipop	2	2	2
Div	0.4	0.4	0.4
Restart	1	1	1
PR	1	3	–
Crossover	–	1 (OP)	1 (OP)
P_c	–	0.3	0.3
Iterations	–	50	–
SelType	*n*-Tournament	*n*-Tournament	*n*-Tournament
Pressure	30	30	30
Mutation	Insertion	Insertion	Insertion
P_m	0.02	0.02	0.02
P_{ls}	0.15	0.15	0.15

Each algorithm is tested with a set of 24 randomly generated test instances. These instances are generated following the procedure explained in Vallada et al. [10] which is briefly summarised as follows: two instances for each combination of n and m are generated where $n = \{50, 150, 250, 350\}$ and $m = \{10, 30, 50\}$. The processing times are uniformly distributed between 1 and 99 as usual in the scheduling literature. The due dates are also generated with a uniform distribution between $P(1 - T - R/2)$ and $P(1 - T + R/2)$ following the method of Potts and Van Wassenhove [52] where P is a lower bound of the makespan and T and R are two parameters called *tardiness factor* and *due date range* which take the following values: $T = \{0.2, 0.4, 0.6\}$, $R = \{0.2, 0.6, 1\}$.

The GAs are coded in Delphi 2007 and run on a Pentium IV 3.0 GHz with 1 GB of main memory. The stopping criterion is set to a maximum elapsed CPU time of $n \cdot (m/2) \cdot 60$ ms. Therefore, the computational effort increases as the number of jobs and/or machines increases.

Regarding the response variable for the experiments, the following performance measure is computed for each instance according to Refs. [22,25,53]:

$$\text{Relative Deviation Index (RDI)} = \frac{Method_{sol} - Best_{sol}}{Worst_{sol} - Best_{sol}} \cdot 100 \qquad (4)$$

where $Best_{sol}$ and $Worst_{sol}$ are the best and the worst solutions obtained among all the methods, respectively and $Method_{sol}$ is the solution obtained with a given algorithm configuration. An index between 0 and 100 is obtained as a result such that a good algorithm will obtain an index close to 0. Note that if the worst and the best solutions are similar, all the combinations would provide the best (same) solution and hence, the index value will be 0 (best index value) for all the similar solutions. We run five replicates of each experiment and the results are analysed by means of a multifactor analysis of variance (ANOVA) where n and m are also considered as factors. First, we check the three main hypotheses of ANOVA: normality, homocedasticity and independence of the residuals [54]. Residuals from the three experiments satisfied all three hypotheses.

We can see in Table 2 a summary with the different parameter values for the algorithms GAPR, GAPR2 and GADV after the calibration experiments (details about the statistical analysis are not shown due to space restrictions).

5. Computational results

We proceed now with the comparison between the three proposed genetic algorithms, GAPR, GAPR2 and GADV against other existing metaheuristics for the objective of minimising the total tardiness. We also compare the proposed methods with some of the most recent and effective algorithms originally proposed for the makespan and adapted to the total tardiness criterion. In the latter

Table 3
Average relative deviation index (*RDI*) and average rank (in brackets) for the evaluated methods, with and without the local search speed ups (denoted by "su"); $t = 60$ in the stopping criterion.

Instance	SAH	SRH	SRHsu	HGA	HGAsu	IG	IGsu	SGALS
50 × 10	47.29(11.30)	34.95(9.98)	35.30(9.84)	15.43(5.68)	11.98(4.09)	17.81(6.37)	11.20(3.70)	11.21(3.78)
50 × 30	67.33(14.72)	39.38(12.47)	40.45(12.48)	19.17(7.30)	15.33(5.52)	22.11(8.54)	16.49(5.94)	13.10(4.44)
50 × 50	62.72(14.63)	39.49(12.70)	41.72(13.01)	16.66(6.44)	14.37(5.25)	24.67(9.65)	17.41(6.77)	12.55(4.27)
150 × 10	37.66(8.14)	52.62(10.20)	52.50(10.33)	39.71(8.58)	28.96(6.04)	32.12(6.75)	23.06(4.10)	29.50(6.29)
150 × 30	59.07(11.36)	58.12(11.38)	59.42(11.35)	47.80(10.46)	35.70(7.52)	39.64(8.45)	29.53(5.43)	35.75(7.51)
150 × 50	69.29(13.57)	65.77(13.26)	65.41(13.22)	46.47(11.05)	35.92(8.05)	40.12(9.18)	30.08(5.99)	35.55(8.12)
250 × 10	19.82(3.80)	36.01(7.85)	36.81(7.96)	46.37(9.20)	33.00(6.95)	32.34(7.20)	22.67(4.14)	26.62(5.65)
250 × 30	37.04(6.79)	53.11(10.00)	54.20(10.10)	55.86(10.67)	42.57(8.44)	40.79(7.92)	31.11(5.22)	38.90(7.56)
250 × 50	52.21(9.59)	62.10(11.52)	60.86(11.38)	57.80(11.10)	44.04(8.40)	43.17(8.16)	34.70(5.76)	40.93(7.64)
350 × 10	9.52(2.57)	20.57(7.00)	20.93(7.30)	44.03(9.76)	28.22(8.00)	23.77(7.39)	13.86(3.36)	16.38(4.87)
350 × 30	16.73(3.80)	27.60(8.05)	27.17(7.94)	47.79(10.86)	30.84(9.21)	26.31(8.32)	18.50(4.91)	21.34(6.54)
350 × 50	28.35(6.67)	37.29(10.04)	37.53(10.23)	50.32(11.72)	34.95(10.01)	29.55(8.09)	21.98(5.08)	26.19(7.46)
Average	42.25(8.91)	43.92(10.37)	44.36(10.43)	40.62(9.40)	29.66(7.29)	31.03(8.00)	22.55(5.03)	25.67(6.18)
Instance	SGALSsu	GADV	GADVsu	GAPR	GAPRsu	GAPR2	GAPR2su	
50 × 10	7.91(2.42)	18.55(7.08)	16.10(6.12)	18.59(7.17)	17.74(6.86)	16.95(6.41)	15.05(5.56)	
50 × 30	10.75(3.45)	20.62(8.52)	16.65(6.28)	20.87(8.52)	17.92(7.16)	19.50(7.86)	16.55(6.36)	
50 × 50	11.11(3.60)	19.69(8.23)	16.44(6.37)	19.72(8.20)	17.42(6.91)	18.66(7.49)	15.84(6.12)	
150 × 10	21.48(3.71)	28.64(5.94)	22.41(3.96)	28.64(5.76)	23.37(4.22)	29.25(5.92)	22.63(4.06)	
150 × 30	26.28(4.42)	35.03(7.04)	25.34(3.87)	31.48(6.02)	26.38(4.43)	33.06(6.63)	23.68(3.38)	
150 × 50	25.20(4.43)	33.83(7.42)	24.49(3.91)	32.27(6.69)	25.04(4.04)	32.68(6.76)	24.77(3.96)	
250 × 10	20.51(3.58)	31.28(7.01)	23.06(4.24)	27.77(6.04)	20.22(3.44)	31.75(7.26)	22.81(4.09)	
250 × 30	29.00(4.46)	39.40(7.42)	27.80(4.00)	35.30(6.32)	24.91(3.20)	41.01(7.78)	27.24(3.84)	
250 × 50	31.89(4.70)	41.37(7.96)	27.53(3.70)	37.00(6.80)	25.54(3.17)	41.40(8.18)	28.59(4.12)	
350 × 10	14.20(3.46)	23.46(7.95)	18.38(5.29)	21.22(6.76)	16.64(4.47)	23.51(8.01)	18.12(5.00)	
350 × 30	17.39(4.25)	24.31(8.05)	17.20(4.24)	21.16(6.70)	15.90(3.63)	25.63(8.23)	16.54(3.98)	
350 × 50	21.45(5.00)	27.55(8.10)	19.32(4.30)	23.94(6.65)	17.49(3.39)	27.15(8.45)	19.27(4.34)	
Average	19.76(3.96)	28.64(7.56)	21.23(4.69)	26.50(6.80)	20.71(4.58)	28.38(7.42)	20.92(4.57)	

case, we choose the following methods to carry out the comparison: the hybrid genetic algorithm (HGA) by Ruiz et al. [3] and the iterated greedy method (IG) by Ruiz and Stützle [43] which were shown by the authors to outperform many other existing PFSP metaheuristics for the makespan objective. We also adapt a genetic algorithm, referred to as SGALS by Ruiz and Allahverdi [55] that was recently proposed for a specific type of PFSP problem and for the maximum lateness criterion.

Regarding the existing methods proposed for the total tardiness objective, from the comparative evaluation by Vallada et al. [10] we have chosen the best two performing methods: the simulated annealing algorithms by Parthasarathy and Rajendran [30] and Hasija and Rajendran [35], which are referred to as SAH and SRH, respectively. We want to remark that all methods have been implemented with and without the speed up procedure explained in Section 3.3. The only exception is the SAH algorithm since in this method the local search is applied in a different and unordered way and therefore the speed up is not possible unless the local search step for this algorithm is completely changed. We differentiate the methods tested with and without the speed up by adding "su" (speed up) to the end of the method's names. For example, HGAsu and HGA refer to the HGA method with and without speed up, respectively.

To test all the methods and variations (15 in total), we use the benchmark of instances available from http://soa.iti.es, which are the same 540 problems that Vallada et al. [10] proposed, ranging from 50 to 350 jobs and from 10 to 30 machines.

All methods are coded in Delphi 2007 and run five independent times on a Pentium IV 3.0 GHz with 1 GB of main memory. The stopping criterion is set to a maximum elapsed CPU time of $n \cdot (m/2) \cdot t$ ms. We tested three different values for t: 60, 90 and 120. Regarding the performance measure, we use again the relative deviation index (*RDI*).

The results are shown in Tables 3–5 where we have averaged the 45 instances of each $n \times m$ group (recall that the instance set contains also different values for the tardiness factor, T and ranges of due dates, R) for the different t values in the stopping criterion (60, 90 and 120).

We can see that the adapted methods show a very good performance, especially those proposed by Ruiz and Stützle [43] (IG) and Ruiz and Allahverdi [55] (SGALS) which shows the best performance for the shortest time ($t = 60$). We can also observe the effect of the speed up procedure: all the methods show substantially better results except for the SRH algorithm by Hasija and Rajendran [35], which seems to be stalled as the speed up provides no advantage. Regarding the proposed algorithms (GAPR, GAPR2 and GADV), we can see that they provide the best results, and significantly outperform the two best existing methods for the total tardiness objective (SAH, SRH) for most instance sizes. As a matter of fact, and on average, SAH, SRH and SRHsu obtain the highest *RDI* values on the comparison, also significantly higher than the methods adapted from other objectives. These results support our initial hypothesis that the adaptation of existing methods to other objectives is a worthwhile effort.

It is also interesting to check wether these observed differences in the *RDI* values are statistically significant. We have carried out a non-parametric test (*Rank-based test*), where a rank is assigned to each original value (*RDI*), instead of a parametric test (ANOVA) since the three necessary hypotheses to apply this test: homogeneity of the variance (homocedasticity), normality and independence of the residuals, are slightly not fulfilled. More specifically, we apply the Friedman test [56] according to Conover [57] to compute the minimal significant difference (MSD) between the rank means of any two algorithms. We use the R language environment for statistical computing (http://www.r-project.org/), and use the code provided in [58]. We can see in Tables 3–5, in brackets, the average rank for each method and the corresponding means plots with MSD intervals ($\alpha = 0.05$) are shown in Figs. 1–3, from the worst method (top of the plot) to the best one (bottom of the plot). Note that in this case, we have 15 methods ranked from 1 to 75 due to the five replicates

Table 4
Average relative deviation index (*RDI*) and average rank (in brackets) for the evaluated methods, with and without the local search speed ups (denoted by "su"); $t = 90$ in the stopping criterion.

Instance	SAH	SRH	SRHsu	HGA	HGAsu	IG	IGsu	SGALS
50 × 10	47.13(11.45)	35.26(10.22)	33.96(9.96)	13.11(4.57)	10.55(3.38)	13.30(4.77)	9.96(3.28)	17.35(6.51)
50 × 30	65.02(14.67)	39.05(12.72)	40.21(12.94)	15.59(5.88)	13.20(4.86)	18.80(7.47)	14.28(5.25)	19.58(7.99)
50 × 50	63.93(14.58)	39.92(12.90)	41.61(13.22)	16.49(6.68)	12.54(4.16)	19.17(7.68)	16.10(6.07)	19.68(8.02)
150 × 10	37.05(8.29)	50.46(10.48)	49.42(10.27)	41.27(9.04)	24.64(5.39)	32.50(7.66)	20.01(3.98)	23.88(5.40)
150 × 30	56.89(11.67)	55.26(11.52)	55.49(11.70)	42.54(10.47)	28.84(6.86)	37.01(9.08)	24.24(5.10)	28.03(6.63)
150 × 50	69.70(13.98)	63.87(13.40)	62.27(13.45)	41.94(10.82)	28.08(6.95)	39.79(10.48)	25.76(6.11)	27.38(6.72)
250 × 10	19.56(4.51)	34.99(8.45)	35.96(8.72)	45.01(9.52)	26.82(6.45)	26.19(6.74)	18.33(4.02)	26.71(7.00)
250 × 30	34.51(7.20)	50.09(10.31)	51.45(10.67)	55.24(11.36)	37.54(8.42)	39.51(9.00)	26.51(5.36)	30.75(6.80)
250 × 50	48.88(10.15)	59.12(12.04)	58.62(12.03)	53.26(11.31)	36.95(8.18)	41.84(9.52)	28.50(5.49)	33.82(7.80)
350 × 10	8.71(2.58)	19.00(7.45)	19.34(7.39)	37.88(10.07)	20.97(6.98)	16.26(6.06)	12.33(3.37)	19.84(7.45)
350 × 30	15.96(4.38)	26.00(9.04)	25.41(8.72)	40.97(11.48)	25.53(8.78)	21.10(7.56)	16.06(4.80)	20.64(7.83)
350 × 50	27.32(7.18)	35.35(11.00)	35.79(10.75)	41.21(12.08)	26.96(8.84)	25.51(8.72)	19.46(5.84)	22.07(8.04)
Average	41.22(9.22)	42.36(10.80)	42.46(10.82)	37.04(9.44)	24.38(6.60)	27.58(7.89)	19.29(4.89)	24.14(7.18)
Instance	SGALSsu	GADV	GADVsu	GAPR	GAPRsu	GAPR2	GAPR2su	
50 × 10	17.59(6.58)	17.05(6.66)	14.24(5.28)	18.29(7.18)	15.95(6.04)	15.74(5.99)	13.13(4.57)	
50 × 30	18.73(7.52)	17.94(7.76)	14.92(5.71)	18.89(8.07)	15.64(6.20)	17.06(7.00)	15.24(5.86)	
50 × 50	18.88(7.61)	17.50(7.13)	15.16(5.66)	18.51(7.70)	15.93(6.12)	17.19(7.09)	14.57(5.30)	
150 × 10	20.15(4.09)	24.10(5.49)	19.37(4.04)	25.73(5.93)	20.56(4.31)	25.16(5.84)	19.63(4.12)	
150 × 30	21.09(4.04)	27.21(6.22)	21.39(4.20)	27.00(6.26)	23.08(4.74)	28.02(6.47)	21.31(3.96)	
150 × 50	21.50(4.44)	27.15(6.65)	20.50(4.21)	27.52(6.69)	22.01(4.84)	27.66(6.84)	21.10(4.39)	
250 × 10	17.60(3.62)	24.91(6.49)	17.82(3.79)	23.14(5.78)	16.42(3.12)	24.93(6.37)	16.70(3.45)	
250 × 30	21.36(3.80)	31.76(6.94)	22.24(3.74)	29.60(6.45)	19.02(3.04)	31.68(7.09)	21.75(3.71)	
250 × 50	21.54(3.53)	33.08(7.70)	21.86(3.45)	29.21(6.45)	20.66(3.31)	33.13(7.73)	22.54(3.88)	
350 × 10	14.72(4.78)	20.84(7.55)	15.02(4.65)	18.55(6.86)	12.88(4.00)	20.71(7.94)	15.12(4.83)	
350 × 30	14.07(3.80)	20.06(7.65)	14.17(3.80)	17.49(6.06)	12.45(3.10)	20.13(7.43)	14.00(3.98)	
350 × 50	15.19(4.079)	23.00(8.04)	15.35(4.12)	19.71(6.45)	13.24(3.00)	22.20(7.84)	15.23(4.06)	
Average	18.54(4.83)	23.72(7.02)	17.67(4.39)	22.80(6.66)	17.32(4.32)	23.64(6.97)	17.53(4.34)	

Table 5
Average relative deviation index (*RDI*) and average rank (in brackets) for the evaluated methods, with and without the local search speed ups (denoted by "su"); $t = 120$ in the stopping criterion.

Instance	SAH	SRH	SRHsu	HGA	HGAsu	IG	IGsu	SGALS
50 × 10	48.95(11.38)	35.96(10.36)	34.44(10.02)	11.26(4.10)	9.31(3.12)	15.65(6.07)	9.00(3.10)	16.83(6.62)
50 × 30	66.77(14.67)	40.48(12.96)	40.49(12.88)	14.49(5.62)	12.61(4.58)	20.73(8.86)	14.40(5.76)	19.69(8.50)
50 × 50	62.23(14.59)	39.32(12.88)	40.32(13.13)	14.83(6.06)	11.66(4.47)	21.80(9.60)	14.80(5.89)	19.99(8.53)
150 × 10	37.42(9.17)	49.39(10.62)	49.66(10.67)	29.20(7.74)	20.29(5.10)	23.53(6.35)	16.92(3.94)	22.36(5.89)
150 × 30	57.05(11.93)	54.32(11.69)	54.27(11.79)	36.90(9.93)	24.13(6.37)	31.04(8.51)	21.78(5.51)	23.95(6.27)
150 × 50	69.84(14.20)	60.03(13.40)	61.76(13.56)	36.99(10.47)	25.25(7.18)	33.29(9.71)	21.54(5.64)	25.52(7.21)
250 × 10	18.87(5.26)	34.17(9.06)	33.77(9.16)	30.68(8.40)	22.63(6.41)	17.75(4.67)	15.87(4.04)	21.29(6.52)
250 × 30	34.67(8.06)	48.89(10.91)	49.67(10.91)	41.54(10.44)	31.88(8.32)	25.69(6.68)	22.67(5.59)	27.27(7.24)
250 × 50	50.76(11.10)	56.59(12.18)	55.94(12.19)	43.82(10.84)	31.35(8.00)	28.93(7.23)	25.55(6.33)	27.50(7.51)
350 × 10	8.45(3.22)	19.28(8.36)	18.53(7.99)	25.00(8.88)	18.64(7.17)	10.01(3.21)	11.20(3.84)	17.52(7.60)
350 × 30	16.67(5.64)	24.95(9.52)	24.52(9.16)	31.21(10.82)	22.38(8.71)	12.55(3.91)	14.66(5.47)	17.24(7.73)
350 × 50	26.37(7.87)	33.57(11.14)	34.28(11.29)	34.65(11.80)	24.83(9.30)	15.76(4.99)	17.57(6.41)	19.22(7.99)
Average	41.50(9.76)	41.41(11.09)	41.47(11.06)	29.21(8.76)	21.25(6.56)	21.39(6.65)	17.16(5.13)	21.53(7.30)
Instance	SGALSsu	GADV	GADVsu	GAPR	GAPRsu	GAPR2	GAPR2su	
50 × 10	15.84(6.24)	15.95(6.59)	13.83(5.52)	16.08(6.66)	14.71(6.07)	13.82(5.55)	13.22(5.11)	
50 × 30	17.98(7.49)	16.48(7.22)	14.62(5.75)	16.79(7.30)	15.01(6.24)	16.09(6.88)	13.54(5.20)	
50 × 50	17.71(7.46)	16.21(6.79)	14.18(5.60)	16.89(7.34)	14.84(6.08)	15.29(6.42)	13.47(5.09)	
150 × 10	16.53(3.94)	21.47(5.56)	16.48(4.03)	23.65(6.30)	19.35(4.96)	22.59(5.99)	17.10(4.08)	
150 × 30	18.60(4.29)	24.42(6.44)	18.53(4.40)	24.56(6.45)	18.86(4.23)	24.50(6.49)	19.06(4.55)	
150 × 50	18.75(4.55)	23.41(6.35)	18.24(4.52)	25.40(7.08)	18.92(4.66)	24.85(6.84)	19.03(4.64)	
250 × 10	15.40(3.92)	21.23(6.49)	14.99(3.75)	20.45(5.85)	14.67(3.60)	21.62(6.63)	15.65(4.05)	
250 × 30	17.64(3.92)	27.23(7.07)	17.82(3.85)	24.29(6.16)	15.96(3.35)	26.68(7.02)	18.26(4.13)	
250 × 50	18.48(4.16)	28.86(7.93)	17.51(3.56)	26.00(6.82)	15.40(2.82)	27.78(7.33)	18.05(3.88)	
350 × 10	12.64(4.61)	18.17(8.07)	11.88(4.51)	15.93(6.88)	10.73(3.89)	17.79(7.70)	12.12(4.73)	
350 × 30	12.26(4.31)	17.28(7.52)	11.82(3.88)	15.17(6.13)	10.25(3.02)	17.79(7.72)	12.12(4.04)	
350 × 50	12.34(4.01)	19.59(7.97)	12.31(4.01)	17.29(6.98)	10.60(2.91)	19.21(8.08)	13.04(4.44)	
Average	16.18(4.91)	20.86(7.00)	15.18(4.45)	20.21(6.66)	14.94(4.32)	20.67(6.89)	15.39(4.50)	

that were carried out in the computational experiment. Note that overlapping confidence intervals means that the overlapped rank means are statistical equal.

The first interesting outcome is the great improvement obtained when the speed up procedure is applied. We can observe that most methods jump several positions in the ranking when using speed

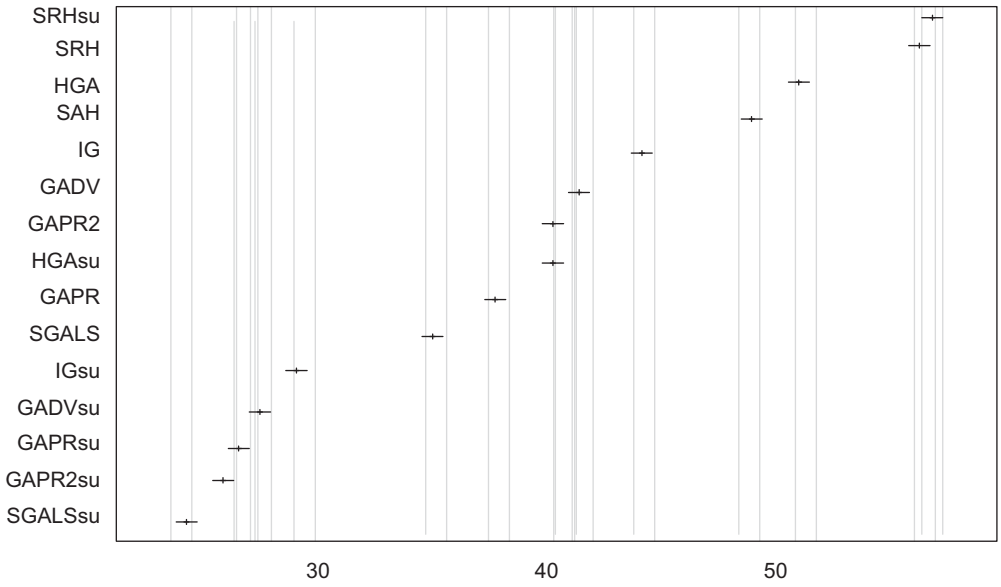

Fig. 1. Means plot and MSD intervals ($\alpha = 0.05$) for the mean rank of the methods evaluated; $t = 60$ in the stopping criterion.

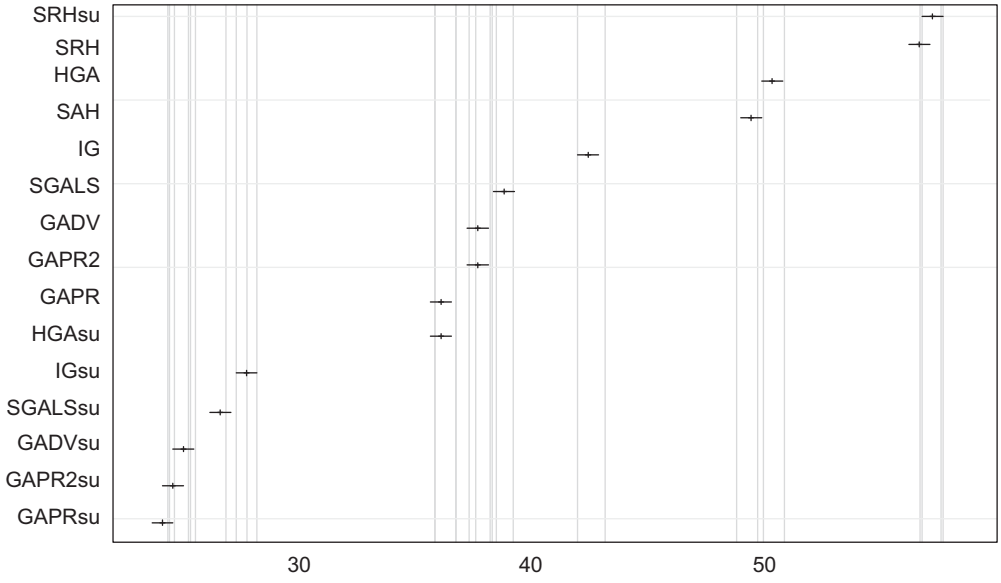

Fig. 2. Means plot and MSD intervals ($\alpha = 0.05$) for the mean rank of the methods evaluated; $t = 90$ in the stopping criterion.

ups. In Fig. 1 we can observe that the algorithm proposed by Ruiz and Allahverdi [55] shows the best performance due to the small amount of time available (from 15 s for smallest instances to 8.75 min for largest ones) to run the methods. Notice that the three proposed algorithms apply diversity techniques and path relinking which need more computation time. Figs. 2 and 3 confirm the previous findings that the best performing methods are those proposed in this paper: GAPR, GAPR2 and GADV. Since the local search method used in the three new proposed methods is similar to that of the IG and HGA algorithms, and all three proposed methods are GAs similar to HGA and SGALS, the better performance observed can only be attributed to the new diversity and path relinking techniques. These two simple additional techniques allow for much better solutions as the results indicate.

From the remaining methods, we can observe that the SGALS algorithm proposed by Ruiz and Allahverdi [55] and the IG reported by Ruiz and Stützle [43] show a very good performance, specially the speed up variants, despite being adapted methods from different objectives; maximum lateness and makespan, respectively. The HGA algorithm by Ruiz et al. [3] shows an average performance when the speed up procedure is applied (HGAsu), but when it is not applied the results are quite poor. Finally, if we compare the results of the existing methods from the total tardiness literature: the SAH by Parthasarathy and Rajendran [30] and the SRH by Hasija and Rajendran [35], we can see that these algorithms show a poor performance and are outperformed by almost all other algorithms. This is an interesting result since these two simulated annealing algorithms have been recently shown to outperform all other existing methods for the same criterion [10]. As a result, we can safely conclude that the three proposed methods are new state-of-the-art algorithms for the PFSP and total tardiness criterion.

A final analysis comes from the fact that in Fig. 3, there are no statistically significant differences between the two speeded up versions of the best proposed algorithms: GAPR and GADV. In order

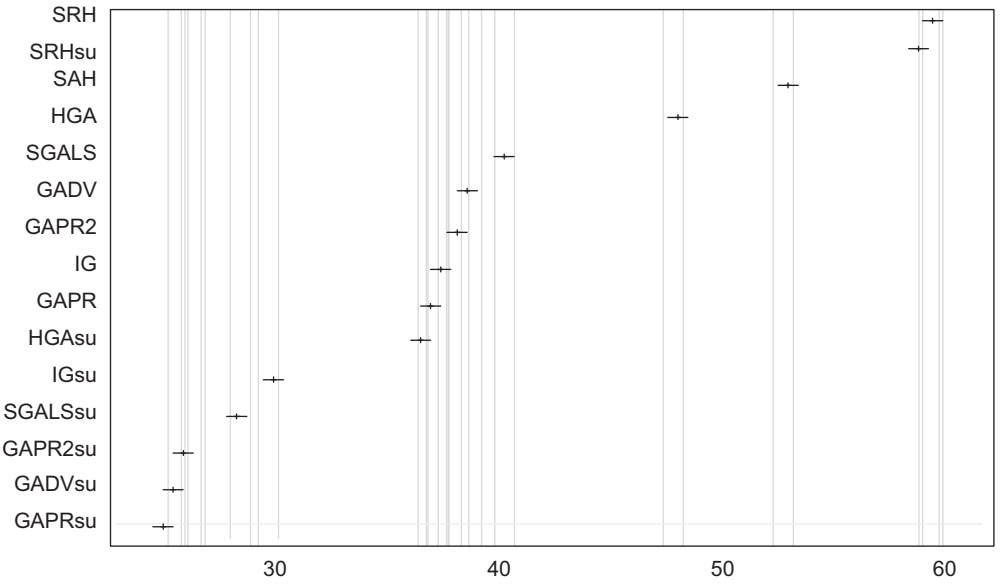

Fig. 3. Means plot and MSD intervals ($\alpha = 0.05$) for the mean rank of the methods evaluated; $t = 120$ in the stopping criterion.

Table 6
Average relative percentage deviation (*RPD*) for GADV and GAPR proposed algorithms.

Instance	GADV	GAPR
50 × 10	1.75	2.03
50 × 30	1.51	1.61
50 × 50	1.01	1.08
150 × 10	3.55	3.81
150 × 30	12.40	14.32
150 × 50	7.96	8.29
250 × 10	5.14	4.88
250 × 30	38.12	35.25
250 × 50	3.38	2.99
350 × 10	95.00	39.29
350 × 30	3.65	3.27
350 × 50	13.69	9.25
Average	15.60	10.51

to closely study this behaviour, we need a more detailed analysis. We carry out another experiment in which GAPR and GADV are run 20 independent times with a stopping criterion set to a maximum elapsed CPU time of $n(m/2)120$ ms. We increase the number of observations in order to allow for a more exhaustive analysis.

In this experiment, the *RDI* is not advisable as a performance measure. The reason is that with two methods alone, we will end up with a method with index 0 (the best) and another with index 100 (the worst). Such a performance measure cannot be analysed from a statistical point of view and negates most differences among algorithms. Therefore, we use the relative percentage deviation (*RPD*) over the best solution instead

$$\text{Relative Percentage Deviation } (RPD) = \frac{Method_{sol} - Best_{sol}}{Best_{sol}} \cdot 100, \quad (5)$$

In this case, we avoid the division by zero because in the few cases were a given best solution was found to be zero, both methods did also obtain this optimal solution. Therefore, the *RPD* will be zero for all the algorithms in these few cases.

In Table 6 we can see the averaged results of the 45 instances for each $n \times m$ group. We can observe that on average the algorithm GAPR seems to be the best. However, a closer analysis indicates that this is not consistent across all $n \times m$ groups.

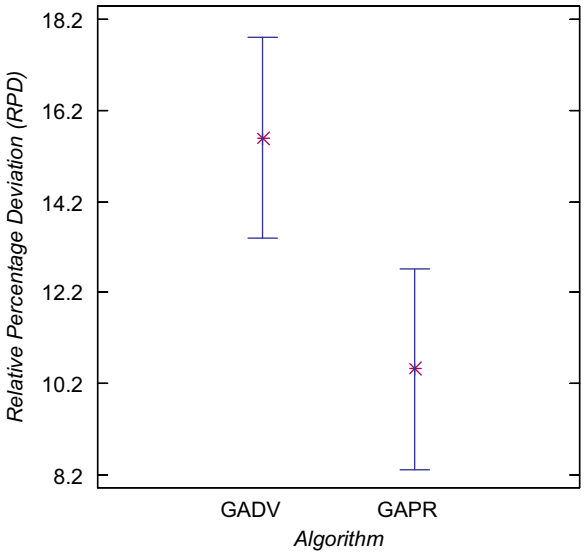

Fig. 4. Means plot and Tukey HSD intervals at the 95% confidence level for the algorithm (*Alg*) factor.

In order to obtain a better picture of the results, we carry out an ANOVA analysis. However, the three hypotheses for the test are not strictly satisfied. This is mainly due to the total tardiness values, that can be very different for the same instance and both methods. Non-parametric methods are also not suitable since we are interested in the interactions between the different values of n and m and non-parametric methods are limited in this regards. We can see in Fig. 4 the means plot with HSD intervals for the algorithm factor (α=0.05).

We can observe that there are statistically significant differences, being the p-value 0.02. According to the means plot, GADV and GAPR are statistically different. Furthermore, if some replicates are removed from the experiment, the observed differences diminish. This indicates that with more and more replicates, the HSD intervals will most probably shrink and more differences would be then observed. Nevertheless, we have to interpret the results with a little scepticism due to the ANOVA assumptions being violated. Additionally, and as can be deducted from Table 6, there seem to be

different outcomes depending on n and m. Other plots (not shown), like that for the algorithm factor and $n = 50$ shows that GADV is the best method with narrow intervals and considerable differences in *RPD* values. However, the means plot (not shown) for the algorithm factor when $n = 350$, results GAPR being statistically better than the other method by a considerable margin. As a conclusion, it seems that GAPR is the best proposed algorithm but there is no clear recommendation about which algorithm, among the proposed, performs best in all situations.

6. Conclusions and future research

In this work we have proposed three new genetic algorithms for the permutation flowshop scheduling problem with the objective of minimising the total tardiness. The algorithms include advanced techniques like local search, diversification and path relinking. We have also adapted some of the most effective algorithms that we can find in the literature for different objectives, mainly the makespan.

The proposed algorithms have been calibrated by means of a design of experiments approach that involves the evaluation of many different alternatives. After this calibration we have obtained the best combination of parameters for each proposed algorithm.

We have carried out an extensive comparison of the three proposed algorithms against the best existing methods for the tardiness objective, as well as with several adapted algorithms originally proposed for other objectives. Moreover, we have applied a speed up procedure to all of them in order to reduce the computational effort needed for the local search technique. The effect of this speed up is also assessed. In total, we have compared 15 algorithms and the results show that the speed up procedure improves the results substantially for almost all compared methods. The results also indicate that most adapted methods from other objectives show a better performance than existing algorithms for the total tardiness criterion. According to the extensive experimental and statistical analysis, the three proposed genetic algorithms clearly outperform, by a considerable margin, the adapted methods and specially the existing methods for the tardiness objective.

Future research directions involve the consideration of more complex scheduling problems with due date based criteria. Since these criteria are important in practice, it seems worthwhile to study more realistic flowshop problems like those with setup times [59], parallel machines [60], no-idle considerations [61] or other interesting extensions with practical applications.

Acknowledgements

The authors are partly funded by the Spanish Ministry of Science and Innovation, under the projects “SMPA—Advanced Parallel Multiobjective Sequencing: Practical and Theoretical Advances” with Reference DPI2008-03511/DPI and “OACS—Advanced Optimization of the Supply Chain” with Reference IAP-020100-2008-11. 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. 3rd ed, Berlin: Springer; 2008.
[2] Rinnooy Kan AHG. Machine scheduling problems: classification, complexity and computations. The Hague: Martinus Nijhoff; 1976.
[3] Ruiz R, Maroto C, Alcaraz J. Two new robust genetic algorithms for the flowshop scheduling problem. OMEGA, The International Journal of Management Science 2006;34:461–76.
[4] Grabowski J, Pempera J. The permutation flow shop problem with blocking. A tabu search approach. OMEGA, The International Journal of Management Science 2007;35:302–11.
[5] Farahmand S, Ruiz R, Boroojerdian N. New high performing heuristics for minimizing makespan in permutation flowshops. OMEGA, The International Journal of Management Science 2009;37(2):331–45.
[6] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 2005;165: 479–94.
[7] Framinan JM, Gupta JND, Leisten R. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. Journal of the Operational Research Society 2004;55(12):1243–55.
[8] Hejazi SR, Saghafian S. Flowshop-scheduling problems with makespan criterion: a review. International Journal of Production Research 2005;43(14):2895–929.
[9] Du J, Leung J. Minimizing total tardiness on one machine is $\mathcal{NP}$-hard. Operations Research 1990;38(1):22–36.
[10] Vallada E, Ruiz R, Minella G. Minimising total tardiness in the m-machine flowshop problem: a review and evaluation of heuristics and metaheuristics. Computers & Operations Research 2008;35(4):1350–73.
[11] Campbell H, Dudek R, Smith M. A heuristic algorithm for the n job m machine sequencing problem. Management Science 1970;16(10):B-630–7.
[12] Gelders L, Sambandam N. Four simple heuristics for scheduling a flow-shop. International Journal of Production Research 1978;16(3):221–31.
[13] Nawaz M, Enscore Jr. E, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA, The International Journal of Management Science 1983;11(1):91–5.
[14] Sen T, Dileepan P, Gupta J. The two-machine flowshop scheduling problem with total tardiness. Computers & Operations Research 1989;16(4):333–40.
[15] Kim Y. A new branch and bound algorithm for minimizing mean tardiness in 2-machine flowshops. Computers & Operations Research 1993;20:391–401.
[16] Pan J, Fan E. Two-machine flowshop scheduling to minimize total tardiness. International Journal of Systems Science 1997;28:405–14.
[17] Pan J, Chen J, Chao C. Minimizing tardiness in a two-machine flow-shop. Computers & Operations Research 2002;29:869–85.
[18] Schaller J. Note on minimizing total tardiness in a two-machine flowshop. Computers & Operations Research 2005;32(5):3273–81.
[19] Kim Y. Minimizing total tardiness in permutation flowshops. European Journal of Operational Research 1995;85:541–55.
[20] Chung C, 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):1–10.
[21] Ow P. Focused scheduling in proportionate flowshops. Management Science 1985;31(7):852–69.
[22] Kim Y. Heuristics for flowshop scheduling problems minimizing mean tardiness. Journal of Operational Research Society 1993;44(1):19–28.
[23] Kalczynski P, Kamburowski J. On the NEH heuristic for minimizing the makespan in permutation flow shops. OMEGA, The International Journal of Management Science 2007;35(1):53–60.
[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] Kim Y, Lim H, Park M. Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process. European Journal of Operational Research 1996;91:124–43.
[26] Adenso-Díaz B. Restricted neighbourhood in the tabu search for the flowshop problem. European Journal of Operational Research 1992;62:27–37.
[27] Widmer M, Hertz A. A new heuristic method for the flow shop scheduling problem. European Journal of Operations Research 1989;41:186–93.
[28] Adenso-Díaz B. An SA/TS mixture algorithm for the scheduling tardiness problem. European Journal of Operational Research 1996;88:516–24.
[29] 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.
[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 & Control 1997;8(5): 475–83.
[31] Parthasarathy S, Rajendran C. Scheduling to minimize mean tardiness and weighted mean tardiness in flowshop and flowline-based manufacturing cell. Computers & Industrial Engineering 1998;34(2):531–46.
[32] Onwubolu G, Mutingi M. Genetic algorithm for minimizing tardiness in flowshop scheduling. Production Planning & Control 1999;10(5):462–71.
[33] Armentano V, Ronconi D. Tabu search for total tardiness minimization in flowshop scheduling problems. Computers & Operations Research 1999;26:219–35.
[34] 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(3):513–22.
[35] Hasija S, Rajendran C. Scheduling in flowshops to minimize total tardiness of jobs. International Journal of Production Research 2004;42(11):2289–301.
[36] Onwubolu G, Davendra D. Scheduling flow shops using differential evolution algorithm. European Journal of Operational Research 2006;171:674–92.
[37] Ronconi D, Henriques L. Some heuristic algorithms for total tardiness minimization in a flowshop with blocking. OMEGA, The International Journal of Management Science 2009;37:272–81.
[38] Chen C-L, Vempati VS, Aljaber N. An application of genetic algorithms for flow shop problems. European Journal of Operational Research 1995;80:389–96.
[39] Reeves CR. A genetic algorithm for flowshop sequencing. Computers & Operations Research 1995;22(1):5–13.

[40] Murata T, Ishibuchi H, Tanaka H. Genetic algorithms for flowshop scheduling problems. Computers & Industrial Engineering 1996;30(4):1061–71.
[41] Etiler O, Toklu B, Atak M, Wilson J. A genetic algorithm for flow shop scheduling problems. Journal of the Operational Research Society 2004;55(8):830–5.
[42] Zhang G, Lai K. Combining path relinking and genetic algorithms for the multiple-level warehouse layout problem. European Journal of Operational Research 2006;169:413–25.
[43] Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 2007;177(3):2033–49.
[44] Wineberg M, Oppacher F. The underlying similarity of diversity measures used in evolutionary computation. In: Genetic and evolutionary computation—GECCO 2003, vol. 2774; 2003. p. 1493–504.
[45] Glover F, Laguna M. Tabu search. Boston: Kluwer Academic Publishers; 1997.
[46] Laguna M, Martí R. GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS, Journal on Computing 1999;11(1):44–52.
[47] Laguna M, Martí R. Scatter search. Boston: Kluwer Academic Publishers; 2004.
[48] Álvarez-Valdés R, Crespo E, Tamarit J, Villa F. GRASP and path relinking for project scheduling under partially renewable resources. European Journal of Operational Research 2008;189:1153–70.
[49] Nowicki E, Smutnicki C. Some aspects of scatter search in the flow-shop problem. European Journal of Operational Research 2006;169:654–66.
[50] Basseur M, Seynhaeve F, Talbi E. Path relinking in Pareto multi-objective genetic algorithms. Evolutionary multi-criterion optimization. Lecture Notes in Computer Science 2005;3410:120–34.
[51] Reeves C, Yamada T. Genetic algorithms, path relinking and the flowshop sequencing problem. Evolutionary Computation 1998;12(4):335–44.
[52] Potts C, Van Wassenhove L. A decomposition algorithm for the single machine total tardiness problem. Operations Research Letters 1982;1(5):177–81.
[53] Zemel E. Measuring the quality of approximate solutions to zero-one programming problems. Mathematics of Operations Research 1981;6(3): 319–32.
[54] Montgomery D. Design and analysis of experiments. 4th ed, New York: Wiley; 2000.
[55] Ruiz R, Allahverdi A. No-wait flowshop with separate setup times to minimize maximum lateness. International Journal of Advanced Manufacturing Technology 2007;35:551–65.
[56] 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.
[57] Conover W. Practical nonparametric statistics. 3rd ed, New York: Wiley; 1999.
[58] Chiarandini M. Stochastic local search methods for highly constrained combinatorial optimisation problems. PhD thesis, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany; 2005.
[59] Allahverdi A, Ng C, Cheng T, Kovalyov M. A survey of scheduling problems with setup times or costs. European Journal of Operational Research 2008;187: 985–1032.
[60] Chen J, Wu T. Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints. OMEGA, The International Journal of Management Science 2006;34:81–9.
[61] Pan Q, Wang L. A novel differential evolution algorithm for no-idle permutation flow-shop scheduling problems. European Journal of Industrial Engineering 2008;2(3):279–97.