Vallada Regalado, E.; Ruiz García, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problemwith sequence dependent setup times. European Journal of Operational Research. 211:612-622. doi:10.1016/j.ejor.2011.01.011.

A Genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times

Eva Vallada, Rubén Ruiz

Grupo de Sistemas de Optimización Aplicada. Instituto Tecnológico de Informática.

Universidad Politécnica de Valencia, Edificio 7A. Camino de Vera S/N, 46021, Valencia, SPAIN.

email: {evallada,rruiz}@eio.upv.es

December 27, 2010

Abstract

In this work a genetic algorithm is presented for the unrelated parallel machine scheduling problem in which machine and job sequence dependent setup times are considered. The proposed genetic algorithm includes a fast local search and a local search enhanced crossover operator. Two versions of the algorithm are obtained after extensive calibrations using the Design of Experiments (DOE) approach. We review, evaluate and compare the proposed algorithm against the best methods known from the literature. We also develop a benchmark of small and large instances to carry out the computational experiments. After an exhaustive computational and statistical analysis we can conclude that the proposed method shows an excellent performance overcoming the rest of the evaluated methods in a comprehensive benchmark set of instances.

Keywords: Parallel machine, scheduling, makespan, setup times.

1 Introduction

In the unrelated parallel machine scheduling problem, there is a set $N = \{1, \ldots, n\}$ of n jobs that have to be processed on exactly one machine out of a set $M = \{1, \ldots, m\}$ of m parallel machines. Therefore, each job is made up of one single task that requires a

[1]Corresponding author. Tel: +34 96 387 70 07, ext: 74911. Fax: +34 96 387 74 99

given processing time. Machines are considered unrelated when the processing times of the jobs depend on the machine to which they are assigned to. This is the most realistic case which is also a generalisation of the uniform and identical machines cases. Moreover, the consideration of setup times between jobs is very common in the industry. The setup times considered in this paper are both sequence and machine dependent, that is, setup time on machine i between jobs j and k is different than setup time on the same machine between jobs k and j. In addition, setup time between jobs j and k on machine i is different than setup time between jobs j and k on machine i'.

The most studied optimisation criterion is the minimisation of the maximum completion time of the schedule, a criteria that is known as makespan or $C_{\max}$. Summing up, in this paper we deal with the unrelated parallel machine scheduling problem with sequence dependent setup times denoted as $R/S_{ijk}/C_{\max}$ (Pinedo, 2008). We propose and evaluate a genetic algorithm that includes a fast local search and a local search enhanced crossover operator among other innovative features that, as we will see, result in a state-of-the-art performance for this problem.

The remainder of this paper is organised as follows: in Section 2 we review the literature on this problem. In Section 3 a Mixed Integer Programing (MIP) model formulation is presented. In Section 4 we describe in detail the proposed genetic algorithm and preliminary computational results. In Section 5, a design of experiments approach is applied in order to calibrate the genetic algorithm. Results of a comprehensive computational and statistical evaluation are reported in Section 6. Finally, conclusions are given in Section 7.

2 Literature review

Parallel machine scheduling problems have been widely studied in the past decades. However, the case when the parallel machines are unrelated has been much less studied. Additionally, the consideration of sequence dependent setup times between jobs has not been considered until recently. In Allahverdi et al. (2008) a recent review of scheduling problems with setup times is presented, including the parallel machine case. In this section we focus our attention on the available algorithms for the parallel machine scheduling problems considering setup times.

In the literature, we can find several heuristic and metaheuristic algorithms for the mentioned problem. However, most of them are focused on the identical parallel machine case. In Guinet (1993), a heuristic is proposed for the identical parallel machines case with sequence dependent setup times and the objective to minimise the makespan. A tabu search algorithm is given in Franca et al. (1996) with the objective to minimise the total

completion time. A three phase heuristic is proposed by Lee and Pinedo (1997) for the same problem with sequence dependent setup times (independent of the machine) and the objective to minimise the sum of weighted tardiness of the jobs. In Kurz and Askin (2001), the authors proposed several heuristics and a genetic algorithm to minimise makespan. Other heuristics for the same problem are those proposed by Gendreau et al. (2001) and Hurink and Knust (2001). In both cases the objective is to minimise the makespan and in the latter case precedence constraints are also considered. In Eom et al. (2002) and Dunstall and Wirth (2005) heuristics are proposed for the family setup times case. In Tahar et al. (2006) a linear programming approach is proposed where job splitting is also considered. Anghinolfi and Paolucci (2007) and Pfund et al. (2008) present heuristic and metaheuristic methods for the same problem, respectively.

The unrelated parallel machines case with sequence dependent setup times has been less studied and only a few papers can be found in the literature. A tabu search algorithm is given in Logendran et al. (2007) for the weighted tardiness objective. Another heuristic for the unrelated parallel machine case with the objective to minimise weighted mean completion time is that proposed by Weng et al. (2001). Kim et al. (2002) proposed a simulated annealing method with the objective to minimise the total tardiness. In Kim et al. (2003) and Kim and Shin (2003) a heuristic and tabu search algorithm were proposed with the objective to minimise the total weighted tardiness and the maximum lateness, respectively. The same problem is also studied in Chen (2005), Chen (2006) and Chen and Wu (2006) where resource constraints are also considered, with the objective to minimise makespan, maximum tardiness and total tardiness, respectively. In Rabadi et al. (2006) a heuristic for the unrelated machine case with the objective to minimise makespan is also presented. Rocha de Paula et al. (2007) proposed a method based on the VNS strategy for both cases, identical and unrelated parallel machines for the makespan objective. In Low (2005) and Armentano and Felizardo (2007) the authors proposed a simulated annealing method and a GRASP algorithm, with the objective to minimise the total flowtime and the total tardiness, respectively.

Regarding the exact methods, there are some papers available in the literature for the parallel machine problem. However, most of them are able to solve instances with a few number of jobs and machines (more details in Allahverdi et al., 2008).

In this paper, we deal with the unrelated parallel machine scheduling problem in which machine and job sequence dependent setup times are considered, i.e., the setup times depend on both, the job sequence and the assigned machine. We evaluate and compare some of the above methods available in the literature. We also propose a genetic algorithm that shows excellent performance for a large benchmark of instances.

3 MIP mathematical model

In this section, we provide a Mixed Integer Programming (MIP) mathematical model for the unrelated parallel machine scheduling problem with sequence dependent setup times. Note that this model is an adapted version of that proposed by Guinet (1993). We first need some additional notation in order to simplify the exposition of the model.

- p_{ij}: processing time of job j, $j \in N$ at machine i, $i \in M$.
- S_{ijk}: machine based sequence dependent setup time on machine i, $i \in M$, when processing job k, $k \in N$, after having processed job j, $j \in N$.

The model involves the following decision variables:

$$\begin{aligned} X_{ijk} &= \begin{cases} 1, \text{ if job } j \text{ precedes job } k \text{ on machine } i \\ 0, \text{ otherwise} \end{cases} \\ C_{ij} &= \text{Completion time of job } j \text{ at machine } i \\ C_{\max} &= \text{Maximum completion time} \end{aligned}$$

The objective function is:

$$\min C_{\max} \tag{1}$$

And the constraints are:

$$\sum_{i \in M} \sum_{\substack{j \in \{0\} \cup \{N\} \\ j \neq k}} X_{ijk} = 1, \qquad \forall k \in N \tag{2}$$

$$\sum_{i \in M} \sum_{\substack{k \in N \\ j \neq k}} X_{ijk} \leq 1, \qquad \forall j \in N \tag{3}$$

$$\sum_{k \in N} X_{i0k} \leq 1, \qquad \forall i \in M \tag{4}$$

$$\sum_{\substack{h \in \{0\} \cup \{N\} \\ h \neq k, h \neq j}} X_{ihj} \geq X_{ijk}, \qquad \forall j, k \in N, j \neq k, \forall i \in M \tag{5}$$

$$C_{ik} + V(1 - X_{ijk}) \geq C_{ij} + S_{ijk} + p_{ik}, \qquad \forall j \in \{0\} \cup \{N\}, \forall k \in N, j \neq k, \forall i \in M \tag{6}$$

$$C_{i0} = 0, \qquad \forall i \in M \tag{7}$$

$$C_{ij} \geq 0, \qquad \forall j \in N, \ \forall i \in M \tag{8}$$

$$C_{\max} \geq C_{ij}, \qquad \forall j \in N, \ \forall i \in M \tag{9}$$

$$X_{ijk} \in \{0,1\}, \qquad \forall j \in \{0\} \cup \{N\}, \forall k \in N, j \neq k, \forall i \in M \tag{10}$$

The objective is to minimise the maximum completion time or makespan. Constraint set (2) ensures that every job is assigned to exactly one machine and has exactly one predecessor. Notice the usage of dummy jobs 0 as X_{i0k}, $i \in M, k \in N$. With constraint set (3) we set the maximum number of successors of every job to one. Set (4) limits the number of successors of the dummy jobs to a maximum of one on each machine. With Set (5) we ensure that jobs are properly linked in machine: if a given job j is processed on a given machine i, a predecessor h must exist on the same machine. Constraint set (6) is to control the completion times of the jobs at the machines. Basically, if a job k is assigned to machine i after job j (i.e., $X_{ijk} = 1$), its completion time C_{ik} must be greater than the completion time of j, C_{ij}, plus the setup time between j and k and the processing time of k. If $X_{ijk} = 0$, then the big constant V renders the constraint redundant. Sets (7) and (8) define completion times as 0 for dummy jobs and non-negative for regular jobs, respectively. Set (9) defines the maximum completion time. Finally, set (10) defines the binary variables. Therefore, in total, the model contains n^2m binary variables, $(n+1)m+1$ continuous variables and $2n^2m + nm + 2n + 2m$ constraints. This MIP model will be used later in the computational experiments.

4 Proposed genetic algorithm

Genetic algorithms (GAs) are bio-inspired optimisation methods (Holland, 1975) that are widely used to solve scheduling problems (Goldberg, 1989). Generally, the input of the GA is a set of solutions called population of individuals that will be evaluated. Once the evaluation of individuals is carried out, parents are selected and a crossover mechanism is applied to obtain a new generation of individuals (offspring). Moreover, a mutation scheme is also applied in order to introduce diversification into the population. The main features of the proposed genetic algorithm are the application of a fast local search procedure and a local search enhanced crossover operator. In the following subsections we report a detailed description about the developed GA.

4.1 Representation of solutions, initialisation of the population and selection operator

The most commonly used solution representation for the parallel machine scheduling problem is an array of jobs for each machine that represents the processing order of the jobs assigned to that machine. This representation is complete in the sense that all feasible

solutions of the MIP model presented in Section 3 can be represented (recall that $C_{\max}$ is a regular criterion and therefore no machine should be left idle when capable of being occupied by a job). The GA is formed by a population of P_{size} individuals, where each individual consists of m arrays of jobs (one per machine).

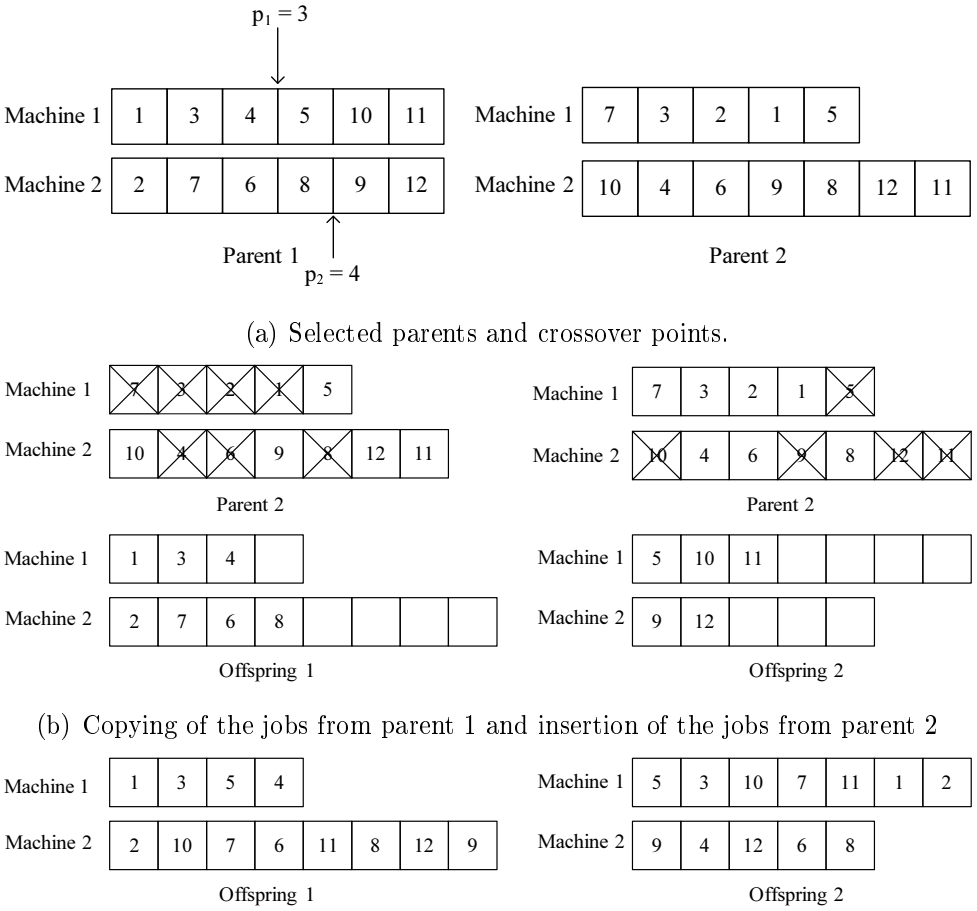
It is also common to randomly generate the initial population in a genetic algorithm. However, a recent trend consists in including in the population some good individuals provided by some effective heuristics. One of the best heuristics for the parallel machine scheduling problem is the Multiple Insertion (MI) heuristic proposed by Kurz and Askin (2001). The MI heuristic starts by ordering the jobs according to a matrix with the processing and setup times. After the initial solution, each job is inserted in every position of every machine and places the job in the position that results in the lowest makespan. One individual of the population is obtained by means of the MI heuristic and the remaining ones are randomly generated. However, in order to obtain a good initial population, the MI heuristic is applied to the random individuals, that is, for each random individual each job is inserted in every position of every machine and finally the job is placed in the position that results in the best makespan.

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 (Ruiz and Allahverdi, 2007). In this case, according to a parameter called "*pressure*", a given percentage of the population is randomly selected. The individual with the lowest makespan value among the randomly selected percentage of individuals wins the tournament and is finally selected. This results in a very fast selection operator, since for each individual the $C_{\max}$ value can be directly used as a fitness value.

4.2 Local search enhanced crossover and mutation operators

Once the selection is carried out and the parents have been selected, the crossover operator is applied according to a probability P_c. 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. One of the most used crossover operators is the One Point Order Crossover adapted to the parallel machine case. Therefore, for each machine one point p is randomly selected from parent 1 and jobs from the first position to the p position are copied to the first offspring. Jobs from the point $p+1$ position to the end are copied to the second offspring. In Figure 1, an example for 12 jobs and two machines is given (setup times between jobs are

not shown for clarity). Two parents are shown and for each machine a point p is selected (Figure 1(a)). Specifically, point p_1 (machine 1) is set to 3 and point p_2 (machine 2) is set to 4. Therefore, the first offspring is formed with the jobs of parent 1 from position 1 to position 3 in machine 1 and jobs from position 1 to 4 in machine 2. The second offspring will contain jobs of parent 1 from position 4 to the last one in machine 1 and from position 5 to the last one in machine 2 (Figure 1(b)). Then, jobs from the parent 2 which are not already in the offspring are inserted in the same order. At this point, it is easy to introduce a limited local search procedure in the crossover operator. When missing jobs are inserted from the parent 2, they are inserted in every position of the offspring at the same machine and finally the job is placed in the position that results in the minimum completion time for that machine. In this way we obtain a crossover operator with a limited local search. In Figure 1(b), jobs from parent 2 which are not already in the offspring will be inserted, that is, jobs 7, 3, 2, 1 of machine 1 from parent 1 are already inserted in offspring 1, so they are not considered. Job 5 will be inserted in every position of machine 1 and finally will be placed in the best position (lowest completion time of the machine). In machine 2, jobs 10, 9, 12 and 11 will be inserted in every position of the machine 2 and will be placed in the best position. In a similar way, in offspring 2 jobs 7, 3, 2 and 1 will be inserted in every position of machine 1 and jobs 4, 6 and 8 will be inserted in every position of machine 2. In Figure 1(c) we can see the offspring obtained after the application of the local search enhanced crossover operator (remember that setup times are not represented). In Section 6, results after testing this local search enhanced crossover operator (LSEC) will be given.



(a) Selected parents and crossover points.

(b) Copying of the jobs from parent 1 and insertion of the jobs from parent 2

(c) Resulting offspring

Figure 1: Example of the local search enhanced crossover operator (LSEC).

Once we obtain the offspring, a mutation operator can be applied according to a probability (P_m). After short experiments testing different mutation operators, the best performance was obtained by the most usual in scheduling problems, that is, the shift mutation. A machine is randomly selected and then a job is also randomly selected and re-inserted to a different position randomly chosen in the same machine.

4.3 Local search and generational scheme

Local search procedures are widely used in genetic algorithms as well as in other meta-heuristic methods to improve solutions. For the parallel machine scheduling problem with the objective to minimise the makespan, it is possible to apply a really fast local search procedure based on the insertion neighborhood, the most used in scheduling problems. We test the inter-machine insertion neighborhood (IMI), which consists of, for all machines, insertion of all the jobs in every position of all the machines. In order to reduce the

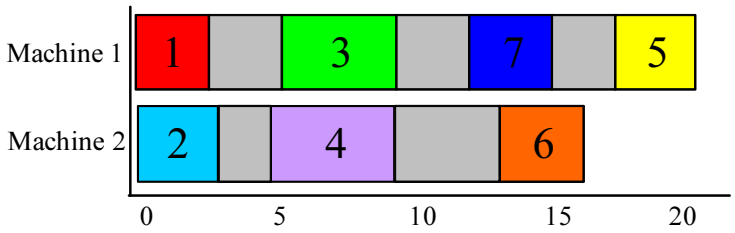
computational effort needed by the local search, it is possible to introduce a simple and very efficient speed up procedure: when a job is inserted, it is not necessary to evaluate the complete sequence for obtaining the new completion time value of the machine. Let us give an example, we have 7 jobs and 2 machines (Figure 2), grey blocks represent the setup times between the jobs. Remember that these setup times are both machine and job sequence dependent. Moreover, these setup times differ between machines 2 and 1. The makespan value for the current shown solution is 21 and a step of IMI local search is applied. For example, we have that job 1 is going to be inserted on the second position of machine 2. To obtain the new completion time of the machines, we only have to remove the processing time of job 1 and the setup time between job 1 and 3 on machine 1. Then, if we want to insert job 1 on machine 2, we have to remove the setup time between jobs 2 and 4. Finally, we have to add the setup time between job 2 and job 1 on machine 2, the processing time of job 1 on machine 2, that could be different than the processing time of job 1 on machine 1, and the setup time between jobs 1 and 4 on machine 2. So we can obtain the new completion time of the machines (Figure 2(b)), with only 3 subtractions and 3 additions and therefore, we can obtain the new makespan value (20) which is better than the previous one. In general, for each insertion, the number of subtractions and additions will be four and four, respectively, if the job is not inserted in the first or last position. For insertions in the first or last position, the number of subtractions and additions will be three and three, respectively (there is not setup time at the beginning or the end of the machine sequence).

In total, the proposed local search will contain the following number of steps (inter-machine insertions):

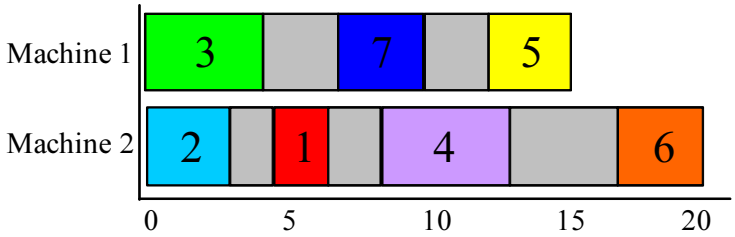
$$\sum_{i \in M} \sum_{\substack{l \in M \\ l \neq i}} n_i (n_l + 1)$$

where n_i and n_l are the number of jobs assigned to machine i and l, respectively, that is, each job on machine i can be inserted in $n_l + 1$ positions on machine l.

During the local search procedure, we have to decide which movements are accepted applying an acceptance criterion. In our algorithm, local search is based on the previous IMI neighborhood, so two machines are involved in the local search procedure. That is, the neighborhood is examined in the following way: for all the machines, all the jobs assigned to this machine are inserted in all positions of all other machines. However, the examination of the neighborhood is carried out between pairs of machines, that is, jobs assigned to machine 1 are inserted in all positions of machine 2 and the acceptance criterion is applied. Then, jobs from machine 1 are inserted in all positions of machine



(a) Example with 7 jobs and 2 machines.



(b) Resulting sequence after the insertion of a job from machine 1 to machine 2.

Figure 2: Example with 7 jobs and 2 machines for the IMI local search.

3 and so on. For every pair of machines, the following acceptance criterion based on the new completion times of the machines after an insertion is analyzed:

- Completion time of both machines is reduced: this is the ideal situation since if the movement is applied both machines reduce the completion time. Obviously, the movement is accepted in this case.

- Completion time of one machine is reduced and completion time of the other machine is increased: in this case we have to decide the acceptance of the movement. If the amount of time reduced on one machine is greater than the amount of time increased on the other machine and the makespan value is not increased, then the movement is accepted. Specifically :

  - C_i, C'_i: completion time of machine i before and after the insertion movement, respectively, $i \in M$.
  - C_l, C'_l: completion time of machine l before and after the insertion movement, respectively, $l \in M, l \neq i$.
  - $C_{\max}, C'_{\max}$: makespan of the sequence before and after the insertion movement, respectively.

  Therefore, in this case the movement will be accepted when:

$$C'_i < C_i, \qquad i \in M \tag{11}$$

and

$$C'_l > C_l, \qquad l \in M \tag{12}$$

and

$$C_i - C'_i > C'_l - C_l, \qquad i, l \in M, i \neq l \tag{13}$$

and

$$C'_{\max} \leq C_{\max} \tag{14}$$

With expression (11), the new completion time of machine i is reduced. Expression (12) shows that new completion time of machine l is increased. Expression (13) compares the amount reduced vs. the amount increased and finally, expression (14) tests the new makespan value.

- Completion time of both machines is increased: in this case the movement is not accepted.

Moreover, the IMI local search is applied until local optima, that is, if after a step of the local search one or more movements are accepted, which means an improvement, the IMI local search is applied again from the beginning. As we can see, applying IMI until local optima results in a very intensive and fine tuned local search. This is only possible with the speed-ups and accelerations proposed. The local search procedure is applied according to a probability (P_{ls}) to the best individual of the initial population and to the offspring. The way to accept movements in the local search is an important feature of the algorithm. We will see in Section 6 the effect of the local search, that improves the results significantly. Moreover, different ways to accept movements were tested obtaining much worse results, so the idea to accept movements analyzing the machines by pairs and when one machine reduces the completion time and the other one increases the completion time, is an innovative feature of the algorithm.

Another aspect to consider is the way the generated offspring after selection, crossover and mutation are inserted into 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 makespan 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. This kind of generational scheme was firstly applied in Ruiz et al. (2006) providing very good results.

4.4 Variants of the proposed genetic algorithm

First, we test a standard genetic algorithm that will be improved by adding new features. The different versions of the proposed algorithm share the features explained in the above sections. Differences among them are mainly related to the crossover operator and the fast local search procedure. Specifically, we start with a genetic algorithm where the initial population is obtained as it was explained in Section 4.1 and new features are added in the following way:

- GAStd1: standard algorithm where the crossover operator is applied without the limited local search explained in Section 4.2 and there is no local search.
- GAStd2: in this case the crossover operator with the limited local search is applied and there is no local search procedure.
- GAStd3: in the third version, crossover operator without the limited local search and the fast local search procedure explained in Section 4.3 are applied.
- GAStd4: the last version includes the crossover operator with the limited local search as well as the fast local search procedure.

We can see in Table 1 a summary with the features of all the proposed versions. Notice that the objective is to start from a very simple genetic algorithm and to add new features in order to improve the effectiveness.

Version	Local search enhanced crossover	Local Search
GAStd1	No	No
GAStd2	Yes	No
GAStd3	No	Yes
GAStd4	Yes	Yes

Table 1: Variants of the genetic algorithm proposed.

Regarding the parameters of the algorithms, all four versions have the same parameter values (standard) that are summarized in Table 2. These parameters will be later calibrated.

Parameter	Value
Population size (P_{size})	50
Selection pressure ($Pressure$)	30
Probability of crossover(P_c)	0.5
Probability of mutation (P_m)	0.2
Probability of local search (P_{ls})	0.4

Table 2: Standard parameter values for the four variants of the genetic algorithm.

4.5 Benchmark of instances

We tested all the variants of the proposed genetic algorithm under a proposed benchmark of instances. The processing times are uniformly distributed between 1 and 99 as it is common in the literature. Regarding the setup times, we generate 4 subsets where the setup times are uniformly distributed between 1 to 9, 1 to 49, 1 to 99 and 1 to 124, respectively. In order to test the behaviour and the sensitivity to the size of the instance of the different algorithms, two sets of instances are generated. The first one, small instances, with the following combinations of number of jobs and number of machines: $n = \{6, 8, 10, 12\}$, $m = \{2, 3, 4, 5\}$. The second one, large instances, where the following values are tested: $n = \{50, 100, 150, 200, 250\}$ and $m = \{10, 15, 20, 25, 30\}$. We generate 10 replicates for each possible combination and in total we have 640 small instances and 1000 large instances to test the algorithms, all of them are available from `http://soa.iti.es`.

Moreover, a different set of instances is generated for the calibration experiments. In this case, 200 instances are randomly generated in the same way explained before. Only large instances are generated for calibration experiments, two replicates for each

combination set of $n \times m$ and setup times. These instances are also available from the same website. This is an important aspect since calibrating over final test instances could bias the results. By using a separated calibration set of instances we ensure that the final results are not skewed by overfitting issues.

4.6 Computational results for the variants of the proposed genetic algorithm

A first experiment was carried out in order to check the features of the proposed genetic algorithm. In Section 4.4 the fourth versions of the algorithm were explained from the most basic to the most advanced. We denote the different algorithms as GAStd1, GAStd2, GAStd3, GAStd4 as in Table 1.

The GAs are coded in Delphi 2007 and run on computers with an Intel Core 2 Duo processor running at 2.4 Ghz with 2 GB of main memory. In order to evaluate all the variants, the benchmark of instances explained in subsection 4.5 is used. The stopping criterion is set to a maximum elapsed CPU time of $n \cdot (m/2) \cdot 30$ milliseconds. Therefore, the computational effort increases as the number of jobs and/or machines increases.

Regarding the response variable for the experiments, the Average Relative Percentage Deviation (RPD) is computed for each instance according to the following expression:

$$\text{Relative Percentage Deviation}(RPD) = \frac{Method_{sol} - Best_{sol}}{Best_{sol}} \cdot 100, \tag{15}$$

where $Best_{sol}$ is the best known solution, obtained after all the experiments carried out throughout the paper, and $Method_{sol}$ is the solution obtained with a given version of the algorithm. We run five replicates of each algorithm. The results for the large instances are shown in Table 3 where the 40 instances of each $n \times m$ group have been averaged (recall that the instance set contains also different values for the setup times). Worst and best results for each $n \times m$ set are in italics and bold face, respectively. The first interesting outcome is the great improvement of the algorithm as the new features are added, especially when the fast local search procedure is included. The second version of the algorithm (GAStd2) which incorporates the crossover operator with the limited local search procedure improves the first version (GAStd1) by almost a 5%. Regarding the fast local search, if the procedure is included in GAstd1 obtaining the GAStd3 version, the results are much better, specifically by a quite large 97%. Finally, the best results are obtained by the GAStd4 version which includes the crossover operator with limited local search as well as the local search procedure. This method improves the previous version

by a 43%. Let us stress that each algorithm is run during the same CPU time, therefore, the results are fully comparable.

Instance	GAStd1	GAStd2	GAStd3	GAStd4
50×10	*25.98*	25.39	16.84	**13.45**
50×15	*22.69*	22.14	15.51	**15.15**
50×20	20.98	*21.16*	15.42	**14.55**
50×25	*21.05*	20.22	**13.74**	14.18
50×30	*24.71*	24.23	17.01	**14.79**
100×10	*31.67*	29.86	18.02	**10.81**
100×15	*27.64*	26.60	19.16	**16.15**
100×20	*24.46*	23.93	16.92	**16.09**
100×25	19.49	*19.55*	**13.19**	13.85
100×30	*18.53*	18.19	**12.28**	12.56
150×10	*31.32*	29.67	15.52	**8.15**
150×15	*28.92*	28.06	18.10	**13.35**
150×20	*25.12*	24.37	16.40	**14.87**
150×25	*19.10*	18.78	13.20	**12.45**
150×30	*16.88*	16.53	11.83	**11.04**
200×10	*29.54*	28.28	15.03	**7.17**
200×15	*27.97*	27.21	16.45	**11.36**
200×20	*24.05*	23.17	14.61	**12.65**
200×25	*19.43*	18.86	12.81	**11.16**
200×30	*16.99*	16.24	11.68	**10.80**
250×10	*29.19*	27.70	14.49	**7.03**
250×15	*27.01*	25.56	14.57	**9.28**
250×20	*23.90*	23.30	14.47	**11.18**
250×25	*19.16*	18.18	12.47	**10.17**
250×30	*16.51*	15.94	10.53	**9.61**
Average	*23.69*	22.93	14.81	**12.07**

Table 3: Average Relative Percentage Deviation (RPD) for variants of the proposed algorithm (large instances).

In order to validate the results, it is interesting to check wether the previous differences in the RPD values are statistically significant. We apply an analysis of variance (ANOVA), (Montgomery, 2007), once the three hypotheses for this statistical test are checked: normality, homocedasticity and independence of the residuals. We can see in Figure 3 the means plot with HSD Tukey intervals (α=0.05). We can clearly see that there are statistically significant differences between the average RPD values among the variants. We can observe that the fourth version (GAStd4) shows the best performance, that is, the genetic algorithm proposed will include the crossover operator with limited local search (LSEC) as well as the local search procedure.

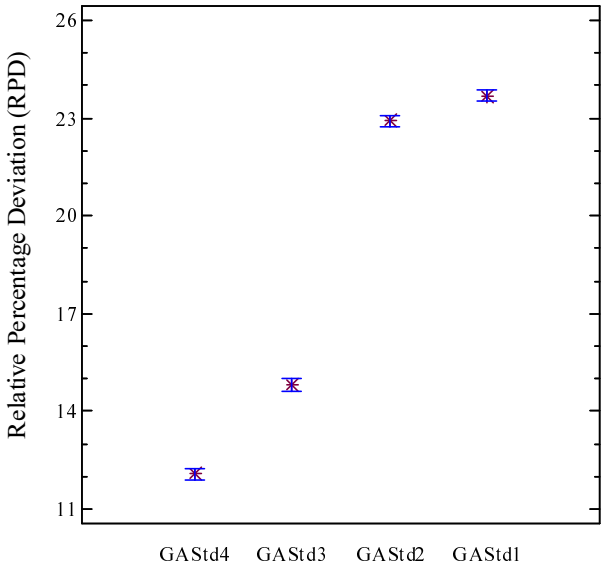


Figure 3: Means Plot and Tukey HSD intervals at the 95% confidence level for the versions of the standard genetic algorithm proposed (large instances).

5 Calibration of the GASTd4 algorithm

Calibration of algorithms is one of the most important steps in order to obtain good results. In the previous Section, we proposed an standard genetic algorithm that was improved by adding new features. After obtaining a good genetic algorithm (GAStd4), in this Section, a first preliminary calibration experiment is carried out where several values are tested for the different parameters. Finally, a second more finely-tuned calibration is applied from the results obtained by the first one. Both calibration experiments are carried out by means of a Design of Experiments (Montgomery, 2007)

We use the benchmark of instances explained in subsection 4.5 for both calibration experiments. That is, 200 test instances for the calibration experiments. Regarding the performance measure, we use again the Relative Percentage Deviation (RPD) following expression (15) and the stopping criterion is set to a maximum elapsed CPU time of $n \cdot (m/2) \cdot 30$ milliseconds as in Section 4.6. We run five replicates of each treatment and the results are analysed by means of an analysis of variance (ANOVA).

The parameters considered in both calibration experiments are: population size (P_{size}), crossover probability (P_c), mutation probability (P_m), local search probability (P_{ls}) and pressure of the selection operator ($Pressure$). We can see in Table 4, the values tested for both calibration experiments (details about the statistical analysis are not shown due to space restrictions). The best combination of values is in bold face. We can see that the second calibration experiment is a refinement of the first one where more values are

added. In this way, we can check the effect of the calibration step. It is also interesting to notice that in the second calibration experiment the value of the local search probability is 1, that is, the local search procedure is always applied, which is mainly due to the fact of being an extremely fast local search.

Parameter	First Calibration	Second Calibration
Population size (P_{size})	20;40;**60**	60;**80**
Probability of crossover(P_c)	0;0.25;**0.5**	**0.5**;1
Probability of mutation (P_m)	0;0.25;**0.5**	**0.5**;1
Probability of local search (P_{ls})	0;0.25;**0.5**	0.5;**1**
Pressure ($Pressure$)	**20**;30;40	**10**;20

Table 4: Values tested for the parameters in both calibration experiments (best combination in bold face).

6 Computational results

After the calibration experiments, two genetic algorithms are obtained with the best combination parameter values for each one, that we denote as GA1 (GAStd4 after the first calibration experiment) and GA2 (GAStd4 after the second calibration experiment), respectively. Along this section, the benchmark of instances used for all the experiments is that proposed in subsection 4.5 (set of 640 small instances and 1000 large instances). The performance measure used is again the Relative Percentage Deviation (RPD) following expression (15) and the stopping criterion is set to a maximum elapsed CPU time of $n \cdot (m/2) \cdot t$ milliseconds as in previous sections. In this case we will test different stopping times (values of t), in order to study the effect of CPU time over the results.

A first experiment is carried out just after the calibration in order to check its effect. Specifically, the two calibrated algorithms (GA1 and GA2) are compared against the fourth initial proposed variants (GAStd1, GAStd2, GAStd3 and GAStd4) which were not calibrated. The parameter t of the stopping criterion is set to 30 as in previous experiments. In Table 5 we can see the results. Worst and best results for each $n \times m$ set are in italics and bold face, respectively. We can clearly observe that the calibration experiments improve the initial algorithms. The results are up to 13% better if we compare the best initial algorithm (GAStd4) with the algorithm obtained after the first calibration (GA1). This difference increases up to 72% after the second calibration. If we focus our attention in the two calibrated algorithms, GA2 is 57% better than GA1. We have to

remark that GAStd4, GA1 and GA2 are exactly the same algorithm, only the parameter values are different. Therefore, the calibration of the algorithms is a really important step since we can improve the results significantly.

Instance	GAStd1	GAStd2	GAStd3	GAStd4	GA1	GA2
50×10	*25.98*	25.39	16.84	13.45	12.29	**6.91**
50×15	*22.69*	22.14	15.51	15.15	13.95	**8.92**
50×20	20.98	*21.16*	15.42	14.55	12.58	**8.04**
50×25	*21.05*	20.22	13.74	14.18	12.39	**8.49**
50×30	*24.71*	24.23	17.01	14.79	14.42	**10.19**
100×10	*31.67*	29.86	18.02	10.81	10.46	**6.76**
100×15	*27.64*	26.60	19.16	16.15	13.95	**8.36**
100×20	*24.46*	23.93	16.92	16.09	13.65	**9.79**
100×25	*19.49*	19.55	13.19	13.85	11.30	**7.86**
100×30	*18.53*	18.19	12.28	12.56	11.31	**8.69**
150×10	*31.32*	29.67	15.52	8.15	8.19	**5.75**
150×15	*28.92*	28.06	18.10	13.35	11.93	**8.09**
150×20	*25.12*	24.37	16.40	14.87	12.66	**9.53**
150×25	*19.10*	18.78	13.20	12.45	10.98	**7.89**
150×30	*16.88*	16.53	11.83	11.04	10.06	**8.03**
200×10	*29.54*	28.28	15.03	7.17	7.56	**6.01**
200×15	*27.97*	27.21	16.45	11.36	10.66	**7.20**
200×20	*24.05*	23.17	14.61	12.65	10.77	**8.36**
200×25	*19.43*	18.86	12.81	11.16	9.86	**7.47**
200×30	*16.99*	16.24	11.68	10.80	9.49	**7.09**
250×10	*29.19*	27.70	14.49	7.03	7.13	**5.99**
250×15	*27.01*	25.56	14.57	9.28	8.97	**6.70**
250×20	*23.90*	23.30	14.47	11.18	10.04	**7.72**
250×25	*19.16*	18.18	12.47	10.17	9.05	**7.49**
250×30	*16.51*	15.94	10.53	9.61	8.10	**6.80**
Average	*23.69*	22.93	14.81	12.07	10.87	**7.77**

Table 5: Average Relative Percentage Deviation (RPD) for the proposed algorithms (large instances).

In order to obtain a more exhaustive evaluation of the proposed method, we proceed now to compare the two calibrated genetic algorithms (GA1 and GA2) against other existing methods for the same problem and the same optimisation objective extracted from the literature. Specifically, we compare the results against the multiple insertion heuristic and the genetic algorithm proposed by Kurz and Askin (2001), denoted as MI and GAK, respectively. The heuristic presented by Rabadi et al. (2006), denoted as Meta, is also considered.

All the methods have been implemented following the guidelines and the explanations of the original papers. Nevertheless, in order to obtain a better picture of the comparison, the heuristic method presented by Rabadi et al. (2006) is also calibrated. The calibration of this method was carried out in the same way that the calibration of the proposed

algorithm, explained in Section 5 and using the same benchmark of test instances. After 14 CPU days running the Meta heuristic for calibration experiments, its three parameters were found to be statistically significant. The calibrated heuristic was also included in the computational evaluation and is denoted by MetaC. More details of the calibration experiments are not shown due to space restrictions but are available upon request from the authors.

The comparative evaluation was carried out under the same benchmark of instances explained in Section 4.5 and used in the previous experiments, that is, the set of 640 small instances and 1000 large instances. Regarding the stopping criterion for the metaheuristic methods, is set to a maximum elapsed CPU time of $n \cdot (m/2) \cdot t$ milliseconds. In this case, all the metaheuristic methods are tested setting different values to t, specifically 10, 30 and 50. In this way, we can study the behaviour of the methods when the amount of time is decreased or increased. For large instances, the maximum CPU time varies from 2.5 seconds for 50 jobs and 10 machines to 37.5 seconds for 250 jobs and 30 machines, when the value of t is set to 10. When t value is set to 50, maximum CPU time varies from 12.5 to 187.5 seconds. Moreover, all metaheuristic methods are run five independent times over each instance to get an accurate picture of the results.

6.1 Comparative evaluation for small instances

First, we carry out a comparative evaluation of all the mentioned methods using the set of 640 small instances. The MIP model explained in Section 3 is also evaluated (denoted as MIP). We construct a model in LP format for each instance in the small set. The resulting 640 models are solved with ILOG-IBM CPLEX 10 on an Intel Core 2 Duo running at 2.4 Ghz each one with 2 GB of main memory. The maximum CPU time for the MIP model is set to one hour, that is, if after one hour no optimal solution is obtained, the best current integer solution is returned.

In Table 6, results for small instances are shown for all the evaluated methods including the MIP model explained in Section 3. Results for the three t values are separated by a slash (t=10/t=30/t=50), except for the MI heuristic method and the MIP model where there is only one value, since heuristic methods do not depend on the CPU time. Obviously, if CPLEX obtained the optimal solution, the RPD value is computed over the optimal makespan value. Specifically, the MIP model is able to obtain the optimal solution for all the instances with six and eight jobs. Regarding the 10 jobs instances, is able to optimally solve all the instances with four and five machines and 11 over 40 and 33 over 40 instances with two and three machines, respectively. For 12 jobs case, the MIP model only obtains the optimal solution for 14 instances with four machines and 34 instances for

five machines. Therefore, the MIP model obtains the optimal solution for 492 over the 640 small instances.

Regarding the rest of the methods, we can see that the best results are provided by the proposed genetic algorithms, specially the GA2 version. In order to validate the results, as in previous experiments, an analysis of variance (ANOVA), is applied in order to check if the observed differences are statistically significant. We can see in Figure 4 the means plot for all t values, on average, with HSD Tukey intervals (α=0.05). We can see that despite of the differences observed from the results in Table 6, GA1, GA2 and the MIP model are not statistically different (confidence intervals are overlapped), that is, on average, the behaviour of three methods is the same. This result is really important since it supports the idea that conclusions can not be obtained from a table of averaged results. From the table, we can state that GA2 is over 38% better than GA1, but when the statistical analysis is applied, the conclusion is much different. Regarding the rest of the methods, we can observe that they show an average and very similar behaviour, except for the MI heuristic, which performance is clearly worse for small instances. Note that MetaC deteriorated, for small instances, when compared to Meta. As we will see later on, the situation reverses for large instances. It seems that the set of calibration parameters for MetaC should be changed for small instances. However, we avoided over-calibration by using the same set of test instances.

Instance	MI	MIP	Meta	MetaC	GAK	GA1	GA2
6×2	13.62	0.00	5.07/5.19/5.29	5.42/5.40/5.39	1.28/1.28/1.28	0.04/0.00/0.00	0.00/0.00/0.00
6×3	19.24	0.00	5.91/6.29/5.92	6.50/6.69/6.33	0.19/0.19/0.19	0.26/0.06/0.15	0.07/0.08/0.08
6×4	18.84	0.00	10.52/9.74/10.33	12.75/12.55/11.24	0.02/0.00/0.00	0.30/0.57/0.40	0.16/0.37/0.27
6×5	15.49	0.00	10.22/10.95/16.08	11.27/10.78/15.96	2.71/0.48/0.36	0.11/0.21/0.23	0.10/0.12/0.21
8×2	14.15	0.00	4.09/3.77/3.98	5.24/4.77/4.80	1.58/1.58/1.58	0.00/0.02/0.07	0.03/0.00/0.03
8×3	18.94	0.00	4.50/4.56/4.55	6.11/5.63/5.60	1.76/1.23/1.23	0.41/0.34/0.20	0.32/0.31/0.24
8×4	19.10	0.00	9.22/9.00/8.67	9.81/9.90/9.55	11.05/4.99/2.65	0.75/0.77/0.66	0.50/0.41/0.39
8×5	22.58	0.00	12.14/12.60/11.28	13.31/13.25/13.38	20.01/11.79/8.78	0.58/0.65/0.49	0.23/0.11/0.20
10×2	16.56	0.40	2.78/2.76/2.72	3.64/3.60/3.34	2.61/2.61/2.61	0.24/0.13/0.19	0.19/0.07/0.17
10×3	19.89	0.09	4.07/3.96/3.86	5.32/5.42/4.82	7.15/3.21/2.71	0.30/0.40/0.26	0.15/0.18/0.20
10×4	21.17	0.00	5.93/5.85/5.59	7.09/7.05/6.97	16.79/10.57/8.77	0.46/0.53/0.45	0.26/0.30/0.32
10×5	24.38	0.00	9.84/10.15/9.63	1.53/11.69/11.19	30.31/23.00/19.45	1.38/1.51/1.16	1.15/1.03/1.15
12×2	17.26	1.63	2.16/2.01/1.97	3.22/3.35/3.25	3.56/2.94/2.94	0.18/0.16/0.21	0.15/0.10/0.09
12×3	23.35	3.19	2.64/2.43/2.33	4.25/3.72/4.01	14.28/9.56/7.77	0.54/0.20/0.22	0.15/0.12/0.08
12×4	25.99	2.57	5.87/5.82/5.82	7.51/7.16/7.36	27.73/22.26/19.61	1.44/1.73/1.29	1.00/0.89/0.75
12×5	21.35	0.24	6.71/6.17/6.21	8.93/8.79/8.44	47.88/39.22/35.85	2.32/1.95/1.98	1.54/1.49/1.50
Average	19.49	0.51	6.35/6.33/6.51	7.62/7.49/7.60	11.81/8.43/7.24	0.58/0.58/0.50	0.37/0.35/0.36

Table 6: Average Relative Percentage Deviation (RPD) for the proposed algorithms: setting of t to 10/30/50 in the stopping criterion only for metaheuristic methods (small instances).

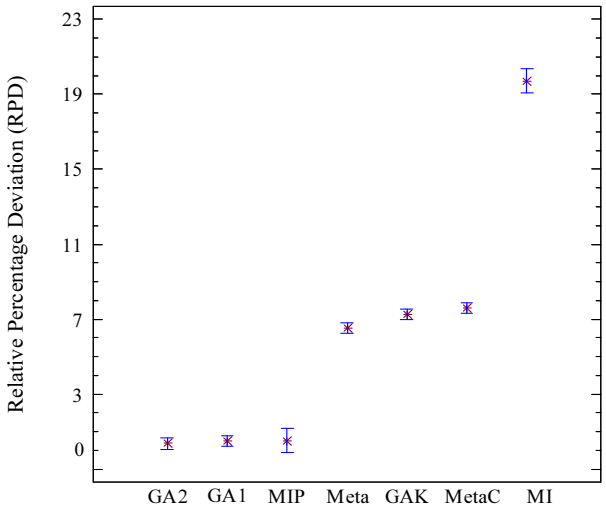


Figure 4: Means Plot and Tukey HSD intervals at the 95% confidence level for the evaluated methods (small instances).

6.2 Comparative evaluation for large instances

In order to analyze the algorithms when the size of the instance is increased, experiments are also carried out using the large set of instances. In this case MIP model is not tested. In Tables 7 we can see the results for large instances obtained by all the methods where the 40 instances of each $n \times m$ group have been averaged. Results for the three t values are separated by a slash (t=10/t=30/t=50), as in the previous case. The first interesting outcome is the great difference in the behaviour of most of the methods. If we focus our attention on the genetic algorithm GAK proposed by Kurz and Askin (2001), we can see that for small instances showed a good performance and for large instances is really far from the best methods, more than 400%. This characteristic is also observed in other methods of the evaluation. Therefore, all these methods are really sensitive to the size of the instance. Regarding the proposed genetic algorithms, both versions, GA1 and GA2 show a very good performance and provide the best results, that is, the proposed algorithms are robust as regards the size of the instances. In order to validate the results, we apply again an statistical analysis (ANOVA) as in previous experiments (GAK algorithm is removed from the statistical analysis since is clearly worse than the remaining ones). In Figure 5 we can see the means plot for all t values, on average, with HSD Tukey intervals (α=0.05). In this case, we can observe that differences between GA1 and GA2 are statistically significant (confidence intervals are not overlapped). So we can state that GA2 is better than GA1, on average. Specifically, after an statistical analysis focused only on both methods and not shown in the paper due to space restrictions, we can conclude that GA2 is the best method when t=30 and t=50, by a 39.8% and a 48.3%,

respectively. However, when t=10, the best method is GA1, specifically almost a 10% better than GA2. This outcome could be expected since when t=10 the total amount of CPU time is really small and we have to remark that local search is always applied in the GA2 algorithm. Therefore, when the amount of time is really small, GA1 is the best method.

Regarding the remaining methods, if we focus our attention on the heuristic proposed by Rabadi et al. (2006), we can see that the calibrated version (MetaC) improves the original one (Meta). More specifically, MetaC version is up to 16% better than Meta algorithm run with the original values for the different parameters. Moreover, we want to remark that all the versions of the proposed algorithm, even those not calibrated, obtain results much better than the remaining ones. In this point, one can think that the good performance of the proposed algorithm is exclusively due to the fast local search procedure. In order to test it, a new experiment was carried out where the fast local search was applied to the MI heuristic by Kurz and Askin (2001) and after this, iterations of the local search starting from a random solution were run until the stopping criterion was met. Results showed that the local search starting from random solutions was not able to improve the solution obtained by the MI with the local search. Moreover, results reported by the MI with the local search were really close to those obtained just by the MI heuristic. Therefore, the good performance of the proposed genetic algorithm is not only due to the calibration phase or the fast local search, but due to the innovative features included, specially the crossover operator, and of course the fast local search and the acceptance criterion of the fast local search.

As regards the rest of the methods, we want to remark the good results obtained by the MI heuristic by Kurz and Askin (2001), which overcomes the rest of the algorithms of the comparison, including the GAK algorithm proposed by the authors in the same paper. This outcome fits with the original paper where the performance of the GAK was shown to be worse than the performance of the MI heuristic when the size of the instance was slightly increased.

Instance	MI	Meta	MetaC	GAK	GA1	GA2
50 × 10	38.99	26.55/22.87/21.51	22.97/20.05/18.88	234.78/228.64/223,77	13.56/12.31/11.66	7.79/6.92/6.49
50 × 15	35.69	31.60/28.17/25.87	28.59/24.23/23.42	333.02/320.22/312.79	13.87/13.95/12.74	12.25/8.92/9.20
50 × 20	39.87	41.41/37.06/34.94	38.65/34.41/32.55	432.11/415.55/410.68	12.92/12.58/13.44	11.08/8.04/9.57
50 × 25	41.63	48.53/41.92/40.25	44.96/39.46/36.90	517.66/499.12/490.61	13.18/12.39/11.87	10.48/8.49/8.10
50 × 30	50.02	65.75/59.83/57.89	63.51/56.18/52.73	638.63/615.16/603.93	15.16/14.42/14.42	10.98/10.19/9.40
100 × 10	42.81	32.98/30.60/29.91	29.05/26.61/25.53	265.36/257.41/255.16	13.11/10.46/9.68	15.72/6.76/5.54
100 × 15	38.43	41.79/38.68/37.31	35.83/33.33/32.20	348.25/340.18/337.81	15.41/13.95/12.94	22.15/8.36/7.32
100 × 20	34.31	49.94/45.97/44.84	44.10/40.50/38.43	434.09/423.80/418.56	15.34/13.65/13.60	22.02/9.79/8.59
100 × 25	29.09	55.41/51.35/49.67	49.55/45.55/43.42	506.90/493.82/485.80	12.47/11.30/11.29	16.71/7.86/8.07
100 × 30	30.04	65.20/60.55/57.68	59.18/54.68/51.84	586.67/570.81/565.94	11.11/11.31/10.98	15.69/8.69/7.90
150 × 10	39.90	34.05/32.41/31.70	29.48/27.27/26.37	273.38/269.12/266.95	10.95/8.19/7.69	18.40/5.75/5.28
150 × 15	38.82	45.67/43.16/42.15	39.36/36.57/35.07	364.05/356.16/353.70	14.51/11.93/11.78	24.89/8.09/6.80
150 × 20	33.75	52.51/49.55/47.63	45.09/41.93/40.68	441.28/430.94/427.80	13.82/12.66/12.49	22.63/9.53/7.40
150 × 25	27.77	56.19/52.76/51.66	49.45/45.30/43.59	500.40/491.64/485.53	11.74/10.98/10.12	17.16/7.89/7.05
150 × 30	26.49	64.35/60.50/58.11	56.96/51.85/50.12	559.26/546.02/539.84	10.74/10.06/9.72	14.85/8.03/7.17
200 × 10	37.73	33.50/31.62/30.72	28.52/26.39/25.68	276.95/274.29/270.74	9.75/7.56/6.17	8.40/6.01/4.24
200 × 15	35.94	45.99/43.03/42.45	38.67/36.22/35.01	372.88/367.34/362.06	12.68/10.66/9.82	10.95/7.20/6.21
200 × 20	30.99	52.22/49.41/48.09	43.67/41.48/39.91	444.28/433.86/431.00	12.59/10.77/10.37	11.07/8.36/6.71
200 × 25	24.70	58.31/54.55/52.72	49.22/45.52/44.27	504.69/494.26/488.29	11.05/9.86/9.51	9.83/7.47/6.82
200 × 30	25.17	63.76/60.23/59.68	55.35/51.54/50.02	560.96/549.97/545.78	10.17/9.49/8.65	9.49/7.09/7.09
250 × 10	36.37	34.28/32.69/31.97	28.81/27.31/26.46	286.79/281.90/280.87	9.64/7.13/5.88	8.20/5.99/4.38
250 × 15	33.34	45.06/42.86/41.71	37.13/35.40/34.17	374.79/369.68/368.84	11.64/8.97/8.05	9.85/6.70/5.12
250 × 20	29.88	53.59/51.49/50.27	44.83/42.03/41.13	453.35/444.10/442.00	11.28/10.04/9.17	10.73/7.72/6.92
250 × 25	25.63	58.60/55.16/54.08	48.87/45.50/44.42	504.55/495.54/489.12	10.37/9.05/7.88	9.55/7.49/6.02
250 × 30	22.34	64.16/60.81/58.70	54.08/50.73/49.29	558.09/548.40/544.23	9.08/8.10/7.18	8.95/6.80/5.91
Average	33.99	48.86/45.49/44.06	42.64/39.20/37.68	430.93/420.72/416.07	12.25/10.87/10.28	13.59/7.77/6.93

Table 7: Average Relative Percentage Deviation (RPD) for the proposed algorithms: setting of t to 10/30/50 in the stopping criterion only for metaheuristic methods (large instances).

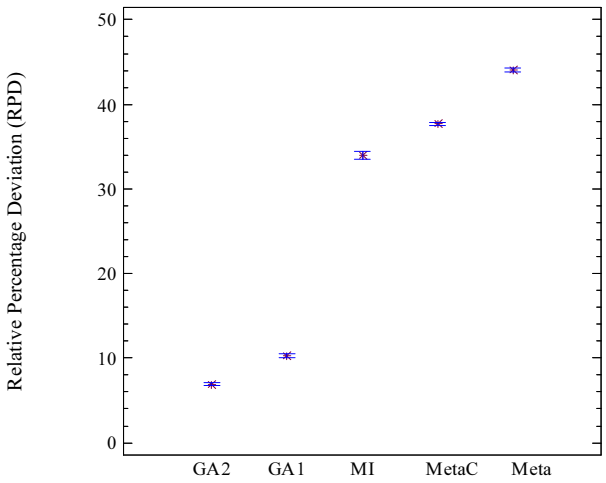


Figure 5: Means Plot and Tukey HSD intervals at the 95% confidence level for the evaluated methods (large instances).

Another interesting factor to study in the experiments is the parameter t for the metaheuristic methods. Remember that all metaheuristic methods are run setting the t value parameter to 10, 30 and 50. In this way, the CPU time for the stopping criterion is increased and the results can be analyzed from the t parameter point of view. We apply an analysis of variance (ANOVA) as in the previous experiments but in this case we focus our attention on the interaction between t and algorithm factors. We can see in Figure 6 the means plot with HSD Tukey intervals (α=0.05) for the interaction between t and algorithm factors for large instances (a very similar picture is obtained for small ones). In this case, we observe that the RPD value is much better as the t parameter increases. However, these differences are much smaller between t=30 and t=50 than between t=10 and t=30, which indicates that some algorithms are converging and/or stalling. If we focus our attention on the proposed algorithms, we can see in Figure 7 the corresponding interaction plot for GA1 and GA2.

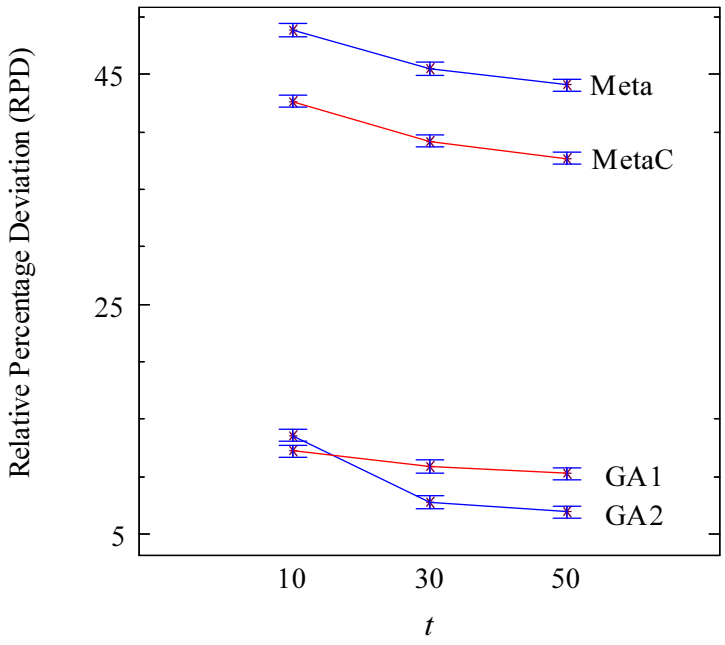


Figure 6: Means Plot and Tukey HSD intervals at the 95% confidence level for the interaction between time (t) and algorithm factors (large instances).

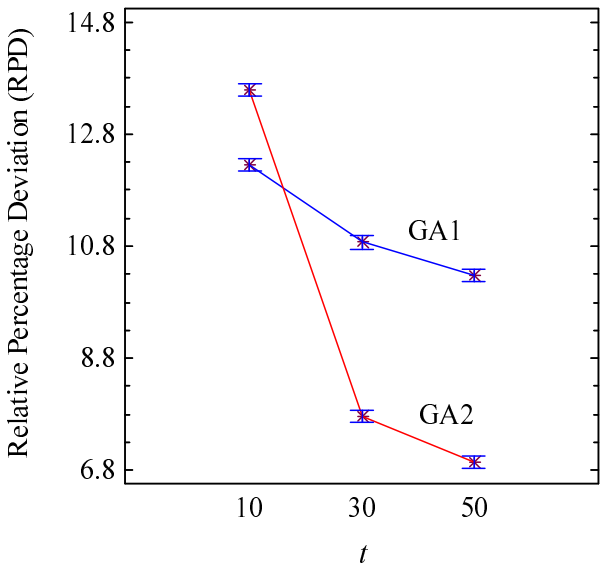


Figure 7: Means Plot and Tukey HSD intervals at the 95% confidence level for the interaction between time (t) factor and algorithms GA1 and GA2 (large instances).

7 Conclusions and future research

In this work we have proposed a genetic algorithm for the parallel machine scheduling problem with sequence dependent setup times with the objective to minimise the makespan. The algorithms include a new crossover operator which includes a limited local search procedure. The application of a very fast local search based on the insertion neighborhood and the acceptance of movements during the local search are also innovative characteristics of the proposed method. A MIP model is also formulated for the same problem.

We carried out two calibration experiments by means of a design of experiments (DOE) approach that involves the evaluation of many different alternatives in two phases. After this calibration we have obtained the best combination of parameters for each proposed method, which resulted in two versions of the proposed algorithm.

We have carried out an extensive comparison of the two versions of the proposed algorithm against some of the best existing methods for the same problem under a comprehensive benchmark of instances. For all the evaluated metaheuristic methods the stopping criterion is set to a maximum elapsed CPU time, testing different values for the amount of time. From the results, we can conclude that both versions of the proposed genetic algorithm obtain the best results, specially the second calibrated version (a refinement of the first one). After several statistical analysis, we can conclude that the proposed methods provide the best results for small instances and especially for large instances. From

the rest of the evaluated methods, we can state that they are very sensitive to the size of the instances since their behaviour changes significantly from results for small instances to results for large instances. Therefore, according to the extensive computational and statistical analysis, the proposed genetic algorithms clearly outperform, by a considerable margin, the remaining methods. We can also conclude that the calibration step is really important, since the second calibration step outperforms the first one, specially for large instances, by a 48%. This is an important result since we have to remark that is the same algorithm but with different parameter values. However, it is important to remark that the good performance of an algorithm is not a result of a calibration. Another algorithm from the literature was also calibrated and its performance was only slightly improved.

Future research directions involve the consideration of more complex neighbourhoods based on a variable neighbourhood search. Application to more sophisticated objectives like completion time variance or CTV as done in Chen et al. (2009) is also a worthy venue of research. Another interesting topic regarding parallel machine scheduling problems is to consider multiobjective optimisation. Other parallel machines problems like for example with batching machines (Damodaran et al., 2009) seems also interesting.

Acknowledgments

The authors are indebted to the referees and editor for the many constructive comments that have significantly improved the paper. This work is partially funded by the Spanish Ministry of Science and Innovation, under the project "SMPA - Advanced Parallel Multiobjective Sequencing: Practical and Theoretical Advances" with reference DPI2008-03511/DPI. The authors should also thank the IMPIVA - Institute for the Small and Medium Valencian Enterprise, for the project OSC with references IMIDIC/2008/137, IMIDIC/2009/198 and IMIDIC/2010/175 and the Polytechnic University of Valencia, for the project PPAR with reference 3147. Eva Vallada is also partly funded by the Government of Comunitat Valenciana under a grant (BEST 2009). The authors are also indebted with Dario Diotallevi for his help in coding some of the re-implemented methods from the literature used in the tests.

References

Allahverdi, A., Ng, C., Cheng, T., and Kovalyov, M. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187:985–1032.

Anghinolfi, D. and Paolucci, M. (2007). Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach. *Computers & Operations Research*, 34:3471–3490.

Armentano, V. and Felizardo, M. (2007). Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based GRASP approach. *European Journal of Operational Research*, 183:100–114.

Chen, J. (2005). Unrelated parallel machine scheduling with secondary resource constraints. *International Journal of Advanced Manufacturing Technology*, 26:285–292.

Chen, J. (2006). Minimization of maximum tardiness on unrelated parallel machines with process restrictions and setups. *International Journal of Advanced Manufacturing Technology*, 29:557–563.

Chen, J. and Wu, T. (2006). Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints. *OMEGA, The International Journal of Management Science*, 34:81–89.

Chen, Y. R., Li, X. P., and Sawhney, R. (2009). Restricted job completion time variance minimisation on identical parallel machines. *European Journal of Industrial Engineering*, 3(3):261–276.

Damodaran, P., Hirani, N. S., and Velez-Gallego, M. C. (2009). Scheduling identical parallel batch processing machines to minimise makespan using genetic algorithms. *European Journal of Industrial Engineering*, 3(2):187–206.

Dunstall, S. and Wirth, A. (2005). Heuristic methods for the identical parallel machine flowtime problem with set-up times. *Computers & Operations Research*, 32:2479–2491.

Eom, D., Shin, H., Kwun, I., Shim, J., and Kim, S. (2002). Scheduling jobs on parallel machines with sequence dependent family set-up times. *International Journal of Advanced Manufacturing Technology*, 19:926–932.

Franca, P., Gendreau, M., Laporte, G., and Müller, F. (1996). A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, 43:79–89.

Gendreau, M., Laporte, G., and Morais-Guimaraes, E. (2001). A divide and merge heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 133:183–189.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading.

Guinet, A. (1993). Scheduling sequence-dependent jobs on identical parallel machines to minimize completion time criteria. *International Journal of Production Research*, 31:1579–1594.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor.

Hurink, J. and Knust, S. (2001). List scheduling in a parallel machine environment with precedence constraints and setup times. *Operations Research Letters*, 29:231–239.

Kim, C. and Shin, H. (2003). Scheduling jobs on parallel machines: a restricted tabu search approach. *International Journal of Advanced Manufacturing Technology*, 22:278–287.

Kim, D., Kim, K., Jang, W., and Chen, F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer Integrated Manufacturing*, 18:223–231.

Kim, D., Na, D., and Chen, F. (2003). Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. *Robotics and Computer Integrated Manufacturing*, 19:173–181.

Kurz, M. and Askin, R. (2001). Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, 39:3747–3769.

Lee, Y. and Pinedo, M. (1997). Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, 100:464–474.

Logendran, R., McDonell, B., and Smucker, B. (2007). Scheduling unrelated parallel machines with sequence-dependent setups. *Computers & Operations Research*, 34:3420–3438.

Low, C. (2005). Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. *Computers & Operations Research*, 32:2013–2025.

Montgomery, D. (2007). *Design and Analysis of Experiments*. John Wiley & Sons, New York, fifth edition.

Pfund, M., Fowler, J., Gadkari, A., and Chen, Y. (2008). Scheduling jobs on parallel machines with setup times and ready times. *Computers & Industrial Engineering*, 54:764–782.

Pinedo, M. (2008). *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, New Jersey, third edition.

Rabadi, G., Moraga, R., and Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, 17:85–97.

Rocha de Paula, M., Gómez-Ravetti, M., Robson-Mateus, G., and Pardalos, P. (2007). Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. *IMA, Journal of Management Mathematics*, 18:101–115.

Ruiz, R. and Allahverdi, A. (2007). No-wait flowshop with separate setup times to minimize maximum lateness. *International Journal of Advanced Manufacturing Technology*, 35(5-6):551–565.

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

Tahar, D., Yalaoui, F., Chu, C., and Amodeo, L. (2006). A linear programming approach for identical parallel machine scheduling with job splitting and sequence-dependent setup times. *International Journal of Production Economics*, 99:63–73.

Weng, M., Lu, J., and Ren, H. (2001). Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. *International Journal of Production Economics*, 70:215–226.