Computers & Operations Research 38 (2011) 1521–1533

Contents lists available at ScienceDirect

Computers & Operations Research

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

Restarted Iterated Pareto Greedy algorithm for multi-objective flowshop scheduling problems

Gerardo Minella *, Rubén Ruiz, Michele Ciavotta

Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Valencia, Spain

ARTICLE INFO

Available online 31 January 2011

Keywords:
Scheduling
Flowshop
Multi-objective
Iterated Greedy

ABSTRACT

Multi-objective optimisation problems have seen a large impulse in the last decades. Many new techniques for solving distinct variants of multi-objective problems have been proposed. Production scheduling, as with other operations management fields, is no different. The flowshop problem is among the most widely studied scheduling settings. Recently, the Iterated Greedy methodology for solving the single-objective version of the flowshop problem has produced state-of-the-art results. This paper proposes a new algorithm based on Iterated Greedy technique for solving the multi-objective permutation flowshop problem. This algorithm is characterised by an effective initialisation of the population, management of the Pareto front, and a specially tailored local search, among other things. The proposed multi-objective Iterated Greedy method is shown to outperform other recent approaches in comprehensive computational and statistical tests that comprise a large number of instances with objectives involving makespan, tardiness and flowtime. Lastly, we use a novel graphical tool to compare the performances of stochastic Pareto fronts based on Empirical Attainment Functions.

1. Introduction and problem description

In many industrial environments the term *job* usually refers to a set of tasks to be carried out by machines over semi-finished goods and/or raw materials in order to obtain a final product. Flowshop is a production layout in which every job consists of the same set of ordered tasks. The goal for this problem is to find a solution (i.e., a sequence of jobs for each machine) that optimises a given objective function. In this work we deal with the permutation flowshop scheduling problem (PFSP), a special case in which, due to shortage of inter-machine buffering or lack of automated handling systems, among other reasons, all machines must process the same sequence of jobs.

Each job is defined by a set of data mainly depending on the criterion to optimise. In this paper we use the processing time of each job j, $j \in N$ on each machine i, $i \in M$, p_{ij} and the due date d_j that represents the delivery date for each job, i.e., job must preferably be finished before d_j.

The completion time of job j on machine i is denoted as C_{ij}. The completion time of a job j in the last machine m, commonly indicated as C_j, is the time needed to complete that job in the shop.

Most literature referring to PFSP focuses on the optimisation of one single objective. The most studied objective in the scheduling theory is, beyond any doubt, the maximum completion time minimisation or makespan, $C_{\max} = \max\{C_j\} = \max\{C_{m,j}\}, j \in N$, i.e., the time needed to complete the processing of all the jobs. Minimising makespan is closely related to the increase of throughput and machine utilisation. A second common criterion is the total flowtime, $TFT = \sum_{j=1}^{n} F_j$. The flowtime for a job is the time elapsed between its release time and its completion time: $F_j = C_j - r_j$. The release time in the earliest possible starting time for a job, in this paper we assume $r_j = 0, \forall j \in N$. Another widely studied objective is the total tardiness, $TT = \sum_{j=1}^{n} T_j$, where T_j is the tardiness of job j, defined as $T_j = \max\{C_j - d_j, 0\}$ and represents the delay in its completion with respect to its due date. Reviews and comparative evaluations for flowshop and makespan objective are presented in Framinan et al. [1], Ruiz and Maroto [2] and Hejazi and Saghafian [3] and Gupta and Stafford [4]. Total tardiness criterion is considered in Vallada et al. [5]. Finally, El-Bouri et al. [6] and Rajendran and Ziegler [7], among many others, consider the total flowtime criterion in their studies.

It is not enough to study each objective separately, as real life problems are often intrinsically multi-objective. A processing sequence with a very good makespan might result in a poor tardiness and vice versa. To take effective decisions, multiple objectives must be considered simultaneously. Multi-objective optimisation has received a big impulse in the last decade and many theoretical and practical results have been obtained.

* Corresponding author. Tel.: +34 96 387 99 52; fax: +34 96 387 72 39.
E-mail addresses: mgerar@iti.upv.es (G. Minella), rruiz@eio.upv.es (R. Ruiz), mciavotta@iti.upv.es (M. Ciavotta).

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.01.010

Furthermore, many new effective techniques have been presented for single-objective optimisation in the PFSP, as for example the Iterated Greedy (IG) of Ruiz and Stützle [8]. These techniques have been seldom explored in multi-objective settings.

In this paper we propose a novel multi-objective algorithm based on the successful ideas of the Iterated Greedy methodology. An efficient management of the Pareto front, a modified crowding selection operator, an effective local search and other techniques are employed in order to obtain high quality and well spread Pareto fronts.

This paper is organised as follows: In Section 2 we provide a short review of multi-objective optimisation results for the PFSP. Section 3 describes the general Iterated Greedy (IG) algorithm and the multi-objective extension. Section 4 presents a description of performance measures, the benchmark used in the computational campaign, the algorithm calibration and the analysis of the results. Finally, conclusions are given in Section 5.

2. Literature review

There are several different approaches to multi-objective optimisation. The most immediate and commonly employed methodology is the so-called "a priori" scheme. As the name implies, this methodology requires some desirability or a priori-istic information given by the decision maker. The simplest method in this class is the weighted combination of all objectives into one single function, which effectively transforms the problem into a single-objective one. The main drawback of this approach is the need of weights for each objective. Setting these weights is not intuitive. Furthermore, different objectives are usually measured in different scales, making the choice of the weighs even more complicated. Broadly speaking, the other class of techniques are referred to as "a posteriori" methods. In this case no former information is provided and a whole set of compromise solutions is returned.

The final goal of an "a posteriori" approach is to provide a set of non-dominated solutions that cover the trade-off between the studied objectives. This set of non-dominated solution is referred to as the Pareto frontier. The decision maker, after the optimisation has been carried out, selects the desired solution from the Pareto frontier. It is out of the scope of this paper to provide a comprehensive and thorough treatment of multi-objective optimisation. Zitzler et al. [9] presented an extensive notation for multi-objective problems, later extended by Paquete [10] and by Knowles et al. [11].

Recently, Minella et al. [12] presented a comprehensive literature review and computational analysis of multi-objective approaches for the PFSP. The literature review that we present now is therefore brief, since most of the existing research is already studied in Minella et al. [12]. Three methods were identified as the best performers in the review of Minella et al. [12]. The first one is an improved version of the simulated annealing of Varadharajan and Rajendran [13] MOSAII$_M$ that provided state-of-the-art over other 22 compared methods. The second one was a hybrid genetic algorithm with local search, referred to as MOGALS from Arroyo and Armentano [14]. The third best performer turned out to be an advanced tabu search method, also hybridised with local search, referred to as MOTS and proposed by Armentano and Arroyo [15].

Recently, other methods have been proposed and they were not included in the review of Minella et al. [12]. One example is the MOIGS algorithms of Framinan and Leisten [16], a multi-objective Iterated Greedy algorithm, based on the Iterated Greedy algorithm of Ruiz and Stützle [8]. Notice that this is the first work that implements the Iterated Greedy ideas into a multi-objective setting. Another recent method is that of Yandra and Tamura [17], referred to as hMGA, which is a multi-objective genetic algorithm with heterogeneous populations. As we will later show, we include these two last methods in our computational experiments.

3. The Restarted Iterated Pareto Greedy algorithm

The Iterated Greedy (IG) algorithm was first proposed in Ruiz and Stützle [8] and it basically consists of iteratively destructing some elements of a solution, reconstructing a new one using a constructive greedy technique and, finally improving it by means of an optional local search procedure. Hence, it is possible to identify inside the algorithm two main phases: the destruction/reconstruction and the local search. During the first phase, some elements of the current solution are randomly removed, and then reinserted in such a way that a new complete, and hopefully better solution is obtained. The reinsertion procedure for the PFSP is based on the well known NEH heuristic of Nawaz et al. [18]. Basically, the NEH is a greedy constructive method that tests every removed element into all possible positions of the current partial solution. The element is placed in the position resulting in the best objective function value. IG is currently being studied in many other research works. For example, Ruiz and Stützle [19] extended the IG method to other objectives and to sequence-dependent setup times. Ying [20] applied IG to multistage hybrid flowshop scheduling problems with multiprocessor tasks. Toyama et al. [21] used Iterated Greedy algorithms for node placement in street networks. Zhi et al. [22] applied IG algorithms to train scheduling problems and finally Tasgetiren et al. [23] used IG for single machine scheduling problems with sequence-dependent setup times.

In the next sections we describe how we have instantiated the IG methodology for the multi-objective PFSP. It has to be noted that IG was originally devised as a non-population-based method and therefore, several extensions and improvements are needed in order to accommodate multiple objectives.

3.1. Basic Iterated Pareto Greedy algorithm

The main contribution of this new algorithm is the handling of a population of non-dominated solutions as a working set instead of just a single solution. In order to make this possible, at each iteration, one solution from the working set has to be selected for further processing. We developed a selection operator which picks up a solution from the working set in such a way so to accelerate the search and to maximise the spread of the final Pareto front. The selected solution is then processed by the greedy phase, in which some elements are removed, like in the original IG. Next, the reconstruction procedure generates a whole set of non-dominated solutions by inserting each removed element into a population of partial solutions. This step is clearly different from the original IG which works with one solution at a time. The working set is updated with the recently created set of non-dominated solutions from the reconstruction procedure. The local search phase is also very different to that of the original IG. Recall that after the greedy phase, the working set is updated, possibly with new solutions. Therefore, the solution selected previously for the greedy phase might not be present in the working set anymore because of being dominated by other new solutions. As a result, the selection operator is applied again to select one solution that will undergo local search. These two phases, namely, greedy and local search, are repeated until a termination criterion is met.

This is the basic version of what we call Iterated Pareto Greedy (IPG). The basic IPG, as we will later elaborate, showed a weak

spot. For large problems, the basic IPG resulted in top performance. However, for small instances, premature convergence appeared. As a result of this, the basic IPG was extended with a simple Restart operator. Experimental results reported in Section 4 demonstrate that the addition of the restart phase increases the quality of the Pareto front for small and mid sized instances, while the performance for large instances remains unchanged. Another interesting point is the relevance of the initial solutions. One has to be careful, since good initial solutions are preferred but at the same time, a sufficiently diverse initial working set is needed. After adding the restart phase, the proposed method has been referred to as Restarted Iterated Pareto Greedy (RIPG). In what follows, we detail all aforementioned aspects of the proposed RIPG.

3.2. Initialisation and selection operator

Initial experiments clearly showed that a good initial working set greatly improves the quality of RIPG. This is certainly expected as it is also the case with the single-objective PFSP. In the single-objective version, most proposed algorithms from the literature use the well known NEH heuristic for the makespan criterion. In the multi-objective version, at least two objectives are considered. Ideally, we should look for a good solution for each objective. To this end, we select two well known heuristics, the aforementioned NEH heuristic of Nawaz et al. [18] which gives good results for makespan and total flowtime and the heuristic proposed in Rajendran and Ziegler [24], suited for total tardiness. We apply each heuristic changing the objectives, obtaining four initial solutions for two-objective problems. Further treatment is needed. First, there is no guarantee that these initial four solutions will be well spread in the Pareto front. Second, the greedy phase is capable of greatly improving initial solutions. However, in some situations, this is not always a positive thing. Selecting only one of the initial solutions for the greedy phase could have a negative result: all other initial solutions could be dominated after this phase. As a result, we loose diversity and coverage in the Pareto front. If all initial solutions are removed in the first step, the search will discard promising search directions. With this in mind, in the first step of the RIPG, all initial solutions are processed by the greedy phase, without applying the selection operator, and for each one, a non-dominated set is obtained. After this, all sets of solutions are joined together and dominated solutions are discarded. After this initialisation, the working set of non-dominated solutions is ready for the main algorithm phases.

In subsequent iterations the Modified Crowding Distance Assignment (MCDA) procedure assigns a fitness value to each element in working set and, based on this value, a solution is selected for the greedy phase. This method is based on the well known Crowding Distance operator initially proposed by Deb [25]. The original method divides the working set into dominance levels, i.e., the set of non-dominated solutions form the first-level Pareto front. Once we remove these elements, we have another non-dominated set of solutions, which correspond with the second-level Pareto front. This procedure is repeated until all solutions are assigned to a Pareto front. Afterwards, the Crowding Distance operator assigns a value to each solution of the working set based on the distance between it and its nearest neighbours belonging to the same Pareto front level. Such technique favours the selection of the most isolated solutions of the first frontier. The idea is that solitary solutions need to be further explored in order to close gaps in the objective solution space. Recall that in multi-objective optimisation, a good coverage of the ideal Pareto front is sought. The main drawback of this technique is that it does not keep track of how many times a solution has been previously selected, because of that, it keeps choosing it again and again. After selection, RIPG applies the greedy and local search phases. Therefore, applying the standard Crowding Distance procedure results in an algorithm that gets easily stuck, as if no improvements are found after the greedy and local search phases, the Pareto fronts do not change and the same solution is selected repeatedly. To avoid this, we add a selection counter to each solution which counts the number of times each solution has been selected. In this way, the probability of selecting a solution from the working set diminishes as the selection counter increases. The proposed Modified Crowding Distance Assignment is given in pseudo-code form in Fig. 1.

procedure Modified Crowding Distance Assignment
$WorkingSet :=$ Current set of solutions
for each $m \in Objectives$
  Sort $WorkingSet$ using the current objective m
  Set the distance of the extreme elements on the $WorkingSet$ to ∞
  $f_m^{max} :=$ maximum value for objective m in $WorkingSet$
  $f_m^{min} :=$ minimum value for objective m in $WorkingSet$
  //Calculate the distance of all other elements in the $WorkingSet$
  for $i := 2, \ldots, WorkingSet_{LastElement-1}$
    $WorkingSet_i.distance := WorkingSet_i.distance + \frac{(WorkingSet_{i+1}.Objetive_m - WorkingSet_{i-1}.Objetive_m)}{(f_m^{max} - f_m^{min})}$
  end for
end for
//Adjust the fitness value with the selection counter in each individual
//First obtain the extreme distance values (for normalising)
$MaxDist := \max(WorkingSet.distance)$
$MinDist := \min(WorkingSet.distance)$
for each $i \in WorkingSet$
  //The Infinite values of the extreme solutions should be replaced
  //by the maximum acceptable value
  if $WorkingSet_i.distance = \infty$ **then**
    $WorkingSet_i.fitness := \frac{1}{(WorkingSet_i.SelectionCounter+1)}$
  else
    $WorkingSet_i.fitness := \frac{\frac{WorkingSet_i.distance + MinDist}{MaxDist + MinDist}}{(WorkingSet_i.SelectionCounter+1)}$
  end if
end for

Fig. 1. Modified Crowding Distance Assignment (MCDA) procedure.

3.3. Greedy phase

The greedy phase works in two steps: First, a block of d consecutive elements is randomly removed from the MCDA-selected solution. Note that in the original IG, the removal is not carried out by groups of elements. The second step iteratively reconstructs the solution by reinserting, one by one, all the d removed elements into all possible positions of a group of partial solutions. In the first iteration, the first of the d removed elements is inserted in all positions of the partial solution (selected solution without the d removed elements). This generates $n-d+1$ new partial solutions. The next removed element, will be inserted in all positions of all previous $n-d+1$ partial solutions, generating a new set of partial solutions of size $(n-d+1)\times(n-d+2)$. This process is repeated until the last removed element is inserted and a set of complete solutions is generated. At the end of this process the total number of generated complete solutions would be equal to

$$\prod_{i=1}^{d}(n-d+i) \geq \prod_{i=1}^{d}(n-d) + \prod_{i=1}^{d} i = (n-d)^d + d!$$

Note how this procedure is radically different to the destruction and reconstruction phase of the regular IG method of Ruiz and Stützle [8] where only one complete solution is kept at all times

and the regular constructive NEH heuristic is used. Our proposed method actually keeps a population of partial solutions to which each removed element is reinserted.

The proposed procedure has one main drawback. Note how for large d values, the size of the partial solutions grows exponentially. For example, for $n=25$ and $d=5$ the number of complete final solutions would be more than 3×10^6. To overcome this problem, each time a set of partial solutions is generated, only the non-dominated partial solutions are kept and the dominated ones are discarded. In the next step, the next removed element is only reinserted in the non-dominated partial solutions. This greedy phase is precisely described in the pseudo-code of Fig. 2.

The solution set obtained from the greedy phase is added to the working population and the dominated elements are removed. Finally, the MCDA is applied on this working set and a new solution is selected for the local search phase.

3.4. Local search phase

A simple local search phase has been designed to refine the search in the space close to the selected solution. As our goal is to have a light, reliable and fast algorithm, we tried not to turn to more complex and time consuming local search techniques.

The local search phase consists in randomly selecting each time, one element from the selected solution, and reinserting it into the n_{neigh} adjacent positions to the right and to the left of the original position, where n_{neigh} is a user-specified parameter. The above procedure is repeated *SelectionCounter* times. This is because if a solution has been selected previously, its closest neighbourhood has been already explored. In the hope of improving the selected solution further, a deeper local search has to be carried out. An upper bound is imposed to the number of removed elements.

Similar to the greedy phase, instead of keeping one full solution in this local search phase, we keep a local working set of solutions. At each step, a removed element is reinserted, and we add the new solutions to the local working set. For each of the removed elements, $n_{neigh}\times2$ new solutions are added to the local working set. After the local search phase, dominated elements are removed from the local working set and the remaining solutions are finally added to the algorithm's working set.

3.5. Restart phase

The last phase is the restart procedure and consists in archiving the current working set, and then creating a new one with randomly generated solutions. This is the simplest possible restart scheme that still allows the algorithm to escape from a situation in which the current working set is stalled.

One of the difficulties of this phase is to know when to perform the restart. A very simple approach is to restart when the working population has not changed for a given number of iterations. The issue of determining when a working set has not changed is not trivial, as this can be measured in a number of ways. Following our ideal that the proposed RIPG has to be simple, we choose to trigger the restart when the size of the working set has not changed for a number of user-specified iterations. Furthermore, initial tests showed that this number of iterations can be set according to the size of the input instance to $n\times2$. This value was obtained as a result of a calibration experiment that will be presented later in Section 4.

```
procedure Greedy Phase
SelectedSolution := MCDA-selected solution
PartialSolution := destruct SelectedSolution by removing a block of d consecutive elements
DestructedElements := Set of d elements removed from SelectedSolution
PartialSolutionsSet := PartialSoluction //Set of partial solutions used in this phase
 for each i ∈ DestructedElements
    for each n ∈ PartialSolutionsSet
      Counter := 0
       for each position j of PartialSolutionsSet_n
          NewPartialSolutionsSet_Counter := Insert Element i in position j of PartialSolutionsSet_n
          Counter := Counter + 1
       end for
    end for
   PartialSolutionsSet := remove dominated partial solutions of NewPartialSolutionsSet
 end for
return PartialSolutionsSet
```

Fig. 2. Greedy phase pseudo-code.

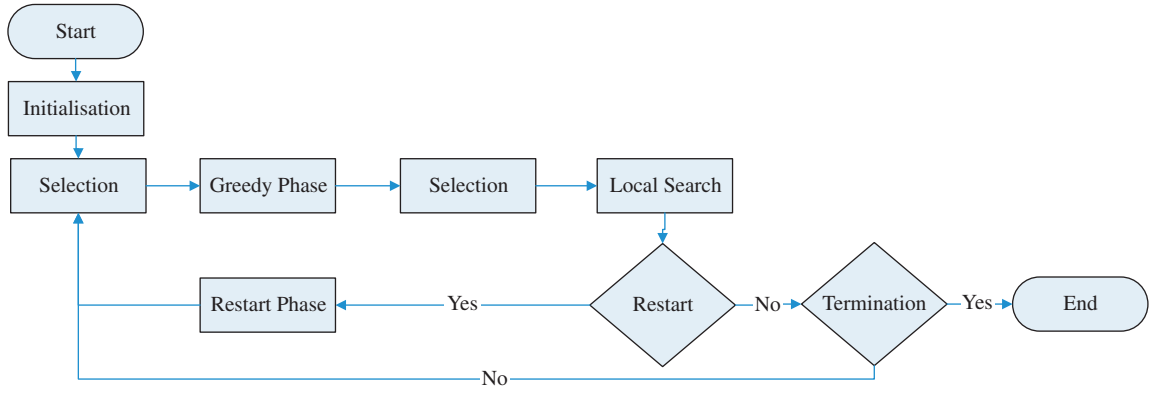

Fig. 3. Final RIPG algorithm flowchart.

Fig. 3 shows the complete flowchart of the final version of the RIPG algorithm with all of its phases.

4. Experimental analysis

In single-objective optimisation a single value is obtained as the result of the application of a given algorithm. Therefore, comparing results of different algorithms is relatively easy. Multi-objective optimisation, however, is much more complicated. Recall that the outcome of a multi-objective method is actually an approximation of the optimal Pareto front with potentially hundreds of solutions. Furthermore, two solutions can be even incomparable. As a matter of fact, the only case in which the comparison of the outcomes of two multi-objective algorithms is clear is when the approximation of the Pareto front of the first algorithm completely dominates that of the second. This only happens when all solutions of the second Pareto front are dominated by at least one solution of the first Pareto front. Nowadays, the issue of comparing multi-objective algorithm in a sound way is still a hot topic of research.

Commonly, Pareto fronts are condensed in single performance measures for comparison purposes. These are commonly referred to as performance indicators. However, these are not the only possible performance measures. Recently, Zitzler et al. [26] carried out a comprehensive study of such existing performance measures. The results show that some metrics frequently used in multi-objective research are not Pareto-compliant, i.e., in some cases, a non-Pareto-compliant performance measure can assign a better value to a Pareto frontier B respect to frontier A even if A dominates B. Again, this result highlights the fact that comparing Pareto fronts is not an easy task at all.

In this paper we employ two performance indicators that were shown to be Pareto-compliant in Zitzler et al. [26]. The first is the Hypervolume Indicator I_H presented in Zitzler and Thiele [27] and more precisely, its unary version from Zitzler et al. [9]. This indicator measures the hypervolume of the objective space dominated by a given Pareto set of points. Notice that in the comparison of two Pareto fronts, a higher value of I_H indicates a better frontier. The hypervolume needs a reference point for closing the volume. In our case, this reference point is obtained by multiplying the worst objective values by 20%. As the objective values are normalised, the maximum I_H value can be obtained by the product of the reference point values: $1.2 \times 1.2 = 1.44$. More formally, the I_H indicator can be defined as follows: Given a set S of solutions and being $s \in S$ one solution. Obj is the number of objectives and h is the number of solutions in set S, then the hypervolume for that solution s is calculated as $I_H(s) = \sum_{1 \le o \le Obj} \sum_{1 \le i \le h} (s_{i,o} - \min S_o)/(\max S_o - \min S_o)$, where min S_o and max S_o are the best and worst values, respectively, for objective o in all solutions in set S.

The second performance indicator is the so-called Unary Epsilon Indicator I_ε^1, proposed by Knowles et al. [11]. It measures the minimum distance between a given Pareto front and the optimal or reference Pareto front. The objective values are normalised before calculating the I_ε^1, this way the indicator varies between 1 and 2. A value close to 1 means that the given Pareto front is close to the reference set. It is formally calculated as follows. P is the Pareto front or a reference set and S is an approximation to the Pareto front. Actually, $I_\varepsilon^1 = I_\varepsilon(S,P)$ where $I_\varepsilon(S,P) = \inf_{\varepsilon \in \mathbb{R}} \{\forall x^2 \in P \exists x^1 \in S : x^1 \preceq x^2\}$ and $x^1 \preceq x^2 \Longleftrightarrow \forall i \in 1, \ldots, n : f_i(x^1) \le \varepsilon + f_i(x^2)$.

This approach of using two performance indicators allows the detection of incomparability situations when the two indicators give contradictory results. This approach was successfully employed for comparing more than 20 multi-objective heuristics for the PFSP in Minella et al. [12].

Another interesting procedure is a graphical tool called the Empirical Attainment Function or EAF. It was first proposed by Grunert da Fonseca et al. [28] and later analysed in more detail by Zitzler et al. [26]. EAF basically depicts the probability for an algorithm to dominate a given point of the objective space in a single run. Since algorithms can be stochastic, different Pareto fronts might be obtained in different runs even for the same instance. EAFs use colour gradients to show the relative number of times that each region of the objective space is dominated. Contrary to performance indicators, EAF do not condense information and the behaviour over the whole Pareto front can be observed. In this work we use the Differential Empirical Attainment Function or Diff-EAF recently proposed by López-Ibáñez et al. [29]. This graphical methodology consists in showing the differences between two EAFs in a single chart. By analysing Diff-EAF images one can easily see in which part of the solution space a method is better than the other. Diff-EAFs show the results for two algorithms and for one single instance in one plot. The main drawback of this analysis method is that one plot has to be generated for each instance and pair of compared methods.

We use the same benchmark set of instances used in Minella et al. [12] which in turn is based on the first 110 instances of Taillard [30]. The benchmark is organised in 11 groups with 10 instances each. Each group contains different combinations of number of jobs n and number of machines m. The $n \times m$ combinations are: $\{20,50,100\} \times \{5,10,20\}$ and $200 \times \{10,20\}$. The processing times (p_{ij}) are generated from a uniform distribution in the range [1,99]. As regards the due dates for the tardiness criterion we use the same approach of Hasija and Rajendran [31]. A tight due date d_j is assigned to each job $j \in N$ following the expression: $d_j = P_j \times (1 + random \cdot 3)$ where $P_j = \sum_{i=1}^{m} p_{ij}$ is the sum of the processing times over all machines for job j and $random$ is a random number uniformly distributed in [0,1]. The whole set of instances is available at http://soa.iti.es.

There is still the issue of the reference Pareto front. In the same web page we have published the best known Pareto fronts for each objective combination and instance. Basically, each reference Pareto front has been constructed from all non-dominated solutions found from all tested methods and experiments. Notice that these reference Pareto fronts are only needed for the calculation of the Unary Epsilon Indicator I_ε^1.

4.1. Algorithm calibration

Before comparing RIPG with the best performing algorithms from the literature, we carry out a calibration experiment. The objective of the calibration is to determine the best value for each parameter of the RIPG algorithm. We also use this calibration experiment in order to gauge the importance and relevance of each algorithm feature.

We employ the Design of Experiments (DOE) technique for the experiment, where the factors affecting the performance of the RIPG are tested in a full factorial experiment which is later analysed by means of the multifactor Analysis of Variance (ANOVA). There are five factors to be tested: (1) The initialisation at two levels: Random or our proposed method. (2) The size of the destruction block d, which has been tested with two levels: 5 and 10. (3) The size of the local search neighbourhood or n_{neigh}, which has been tested at two levels: 3 and 5. (4) Number of local search iterations, which has been tested at two levels: 5 and *SelectionCounter* (recall that *SelectionCounter* is the number of times that a selected solution was previously selected). (5) Restart operator, which has been tested at three levels: no restart, restart after 10 non-improving iterations and restart after $n \times 2$ non-improving iterations, where n is the number of jobs in the tested instance.

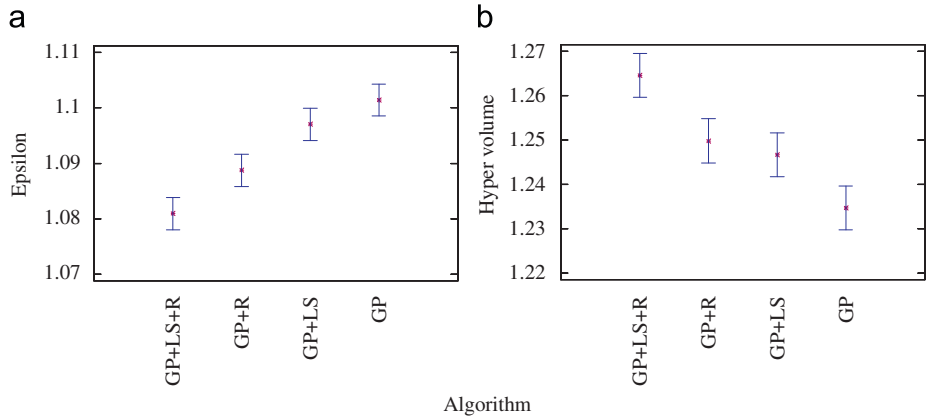

Fig. 4. Means plot and 99% Tukey intervals for the Epsilon indicator (a) and hypervolume indicator (b) in the ANOVA experiment for several configurations of RIPG algorithm.

More specifically, the factors (3) and (4) are actually combined into a single factor with five levels where the fifth level is actually not using local search at all.

As a result, there are four factors at 2, 2, 5 and 3 levels, respectively. This gives a total of 60 algorithm configurations. Each configuration is tested with the 110 instances and 10 replicates are run for each instance. A total number of 66,000 results is therefore obtained. RIPG was coded in Delphi 2007 and run on a cluster of 12 computers with Intel Core 2 Duo E6600 processors running at 2.4 GHz and 1 GByte of RAM memory. Only one core is used at each computer. Each configuration is run during the exact same CPU time stopping criterion that depends on the size of the input instance: $T_{CPU} = n \cdot m/2 \cdot t$ ms. where $t=100$. The total calibration time has been therefore 41.57 CPU days. The response variables of the ANOVA experiment are the Hypervolume and Epsilon indicators.

Since the ANOVA is a parametric statistical inference tool, it is necessary to check the three main hypotheses which are normality, homoskedasticity and independence of the residuals. From the analysis of the residuals resulting from the experimental data all three hypotheses are easily satisfied. After the calibration we observed that a random initialisation had a very poor performance. Therefore, our proposed initialisation proves to be much better. The best size for the destruction block resulted to be $d=5$. The other factors were fixed as follows: $n_{neigh} = 5$ and number of local search iterations fixed to *SelectionCounter*. Lastly, the number of non-improving iterations after which Restart is applied is fixed to $n \times 2$.

As previously mentioned, during this experiment we also tested the relevance of each phase of our proposed RIPG. Some of these results are presented in Fig. 4 for epsilon (a) and hypervolume (b) indicators, respectively. The configurations reported in the picture are: *GP*, which indicates that only the greedy phase without local search and without restart is tested, *GP+LS* that is the basic algorithm *IPG* made of only greedy and local search phases (no restart), *GP+R* that consists only of the greedy and restart phases, and finally *GP+LS+R* that consists of all the phases working together and conform the RIPG algorithm.

As we can see, each algorithm phase improves results as *GP* is the worst performer. Surprisingly, the impact of the restart phase on the results is very high, as *GP+R* produces better results than the local search *GP+LS*. This result not only justifies the inclusion of this restart phase inside our algorithm but also encourages the development of such methods of solution improvements.

4.2. Computational evaluation

We compare our algorithm with the three best performing algorithms according to the review of Minella et al. [12], along with two recent methods proposed later and that were not evaluated by Minella et al. [12]. Table 1 shows the list of the re-implemented methods in more detail. All algorithms are re-implemented in Delphi 2007 following the original papers. The only modification carried out is the accelerations included in the MOIGS of Framinan and Leisten [16]. We test two different bi-objective combinations, makespan-flowtime and makespan-total tardiness. All methods are run on the same cluster of computers mentioned in the calibration section. The stopping criterion also changes with the instance size as: $n \cdot m/2 \cdot t$ ms, where t is an input value. By using this stopping criteria we give more computation time for bigger instances which have larger solution spaces. Two different values of t are used in the experiments: 100, 200 in order to check how different times affect the tested methods. Each algorithm is run 10 times (replicates) for each combination of stopping criterion and pair of objectives. For each run and each algorithm, a computer from the cluster is selected randomly, and the results are collected at the end of the experiment. A total of 7 [algorithms] $\times$ 10 [replicates] $\times$ 110 [instances] $\times$ 2 [objectives] $\times$ 2 [CPU time termination criteria] = 30,800 samples have been collected and analysed by means of a multi-factor parametric Analysis of Variance (ANOVA) test along with a non-parametric Friedman rank-based test on both performance indicators. We carry out two statistical tests for each combination of objectives, performance indicators and stopping times for a total of 16 statistical tests. Normality, homogeneity of variance and independence of residuals hypothesis were checked on data before applying ANOVA tests. We are performing four different statistical tests on each set of results and therefore, a correction on the confidence levels must be performed since the same data set is being used to make more than one inference. In order to counter this potential problem we employ the Bonferroni adjustment, and we set the adjusted significance level α_s to $\alpha/4 = 0.05/4 \simeq 0.01$. This means that all the tests are carried out at a 0.01 adjusted confidence level for a real confidence level of 0.05. The outcomes

Table 1
List of re-implemented or adapted methods compared in this work.

Algorithm	Type	Author/s	Year
MOTS	Taboo search	Armentano and Arroyo	2004
MOGALS	Genetic algorithm	Arroyo and Armentano	2005
MOSAII	Simulated annealing	Varadharajan and Rajendran	2005
hMGA	Genetic algorithm	Yandra and Tamura	2007
MOIGS	Iterated Greedy	Framinan and Leisten	2008
$MOSAII_M$	Simulated annealing	MOSAII adaptation	2008
RIPG	Iterated Greedy	Presented in this work	

of those tests are reported here in the form of ANOVA charts with 99% Tukey confidence intervals (95% adjusted confidence level). Notice that overlapping intervals of two algorithms indicates that there is no statistical difference between their performance. For reasons of space only eight ANOVA graphs and two tables have been included into this work. All other tests as well as the non-parametric plots are available as online material and from http://soa.iti.es.

Table 2
Results for the makespan and total flowtime criteria for $t=100$ and 200 stopping times. The methods are sorted according to I_H.

#	Time	100			200	
	Method	I_H	I_ε^1	Method	I_H	I_ε^1
1	RIPG	**1.270630**	**1.078920**	RIPG	**1.295480**	**1.066000**
2	$MOSAII_M$	1.227270	1.114630	$MOSAII_M$	1.248160	1.105500
3	MOIGS	1.170090	1.135420	MOIGS	1.212110	1.111510
4	MOSAII	1.154200	1.150990	MOSAII	1.146800	1.153970
5	MOGALS	1.130070	1.169700	MOGALS	1.151050	1.156690
6	MOTS	1.041530	1.230810	MOTS	1.066500	1.212490
7	hMGA	0.569154	1.547800	hMGA	0.592918	1.528700

Table 3
Results for the makespan and total tardiness for $t=100$ and 200 stopping times. The methods are sorted according to I_H.

#	Time	100			200	
	Method	I_H	I_ε^1	Method	I_H	I_ε^1
1	RIPG	**1.256220**	**1.08086**	RIPG	**1.280370**	**1.066090**
2	$MOSAII_M$	1.214630	1.11139	$MOSAII_M$	1.234290	1.102120
3	MOIGS	1.153630	1.13636	MOIGS	1.194390	1.113260
4	MOSAII	1.147280	1.14568	MOSAII	1.136630	1.149310
5	MOGALS	1.026660	1.24139	MOGALS	1.046410	1.227070
6	MOTS	0.890548	1.33638	MOTS	0.917784	1.318060
7	hMGA	0.611876	1.51659	hMGA	0.636000	1.497120

4.2.1. Makespan and total flowtime results

Table 2 shows the average values of hypervolume and epsilon indicators for the results with the makespan and total flowtime criteria. Table 3 gives the same results but for makespan and total tardiness. Both tables show the results for $t=100$ and 200 CPU time termination criteria. Each value is averaged across 110 instances and 10 replicates per instance (1100 values). The values in the tables are given in descending order with respect to the hypervolume indicator, this means that the first algorithm appearing is the best performer for the hypervolume indicator. The best values for both indicators are highlighted using bold characters. In a second column we show the epsilon indicator values. Observing Table 2 it can be noticed that RIPG outperforms all other methods for both indicators and stopping times. The second position is occupied by the $MOSAII_M$ algorithm which is the modified version of MOSAII. The previous table only shows the total averages for all instances and replicates. A more detailed analysis can be carried out. For example, Framinan and Leisten [16] showed that their proposed MOIGS was better than MOSAII. However, the authors only considered small and medium instances of up to 100 jobs. In our experiment, for 200 jobs, MOSAII turns out to be better as shown in Fig. 5 which depicts the hypervolume and epsilon indicator results for the $t=100$ CPU stopping time and makespan and total tardiness criteria pair. Fig. 6 shows the ANOVA test for $t=100$ and I_ε^1, while Fig. 7

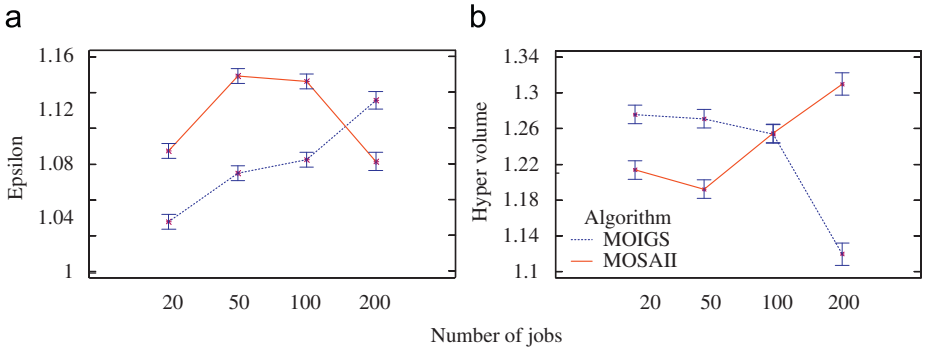

Fig. 5. Means plot and 99% Tukey intervals for the epsilon indicator (a) and hypervolume indicator (b) factors in the ANOVA experiment for MOIGS and MOSAII algorithms.

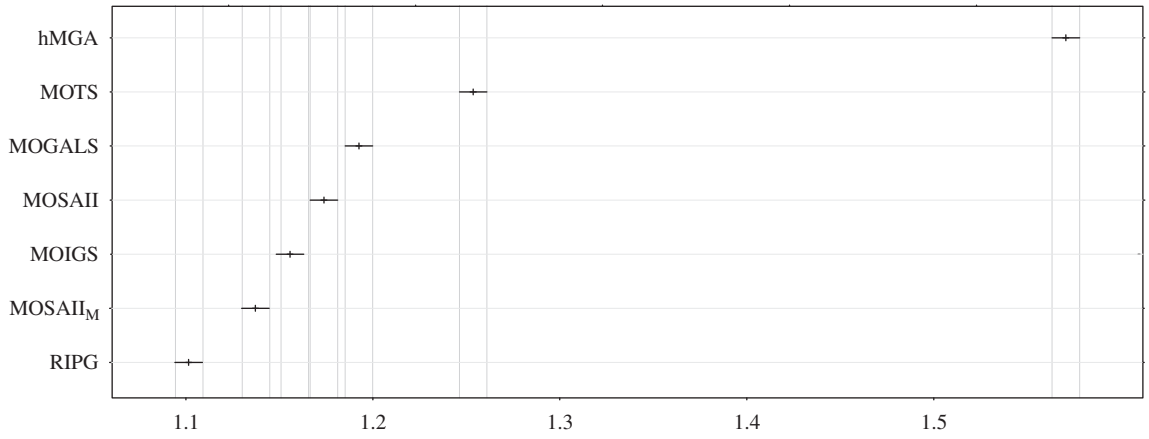

Fig. 6. Results of ANOVA test for epsilon indicator and $t=100$ for makespan and total flowtime objectives with 99% Tukey intervals (95% adjusted confidence level).

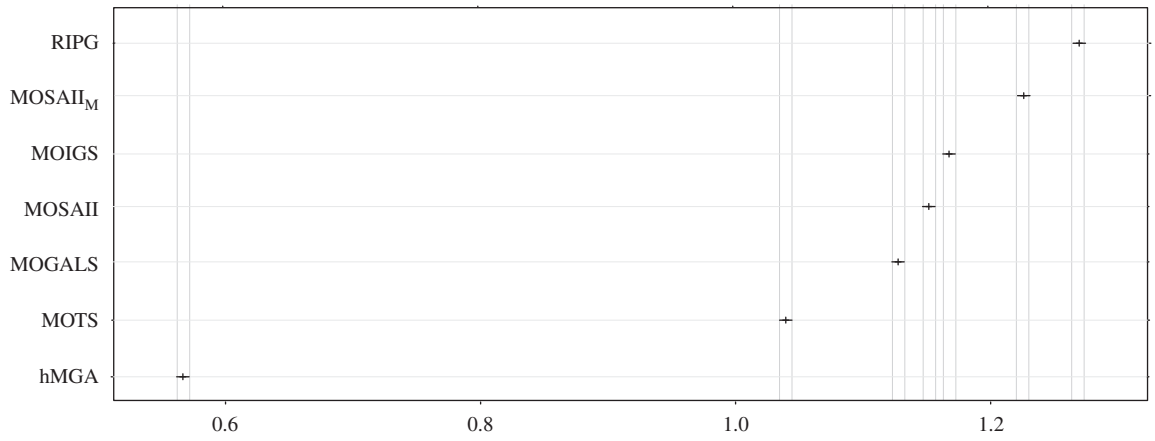

Fig. 7. Results of ANOVA test for hypervolume indicator and $t=100$ for makespan and total flowtime objectives with 99% Tukey intervals (95% adjusted confidence level).

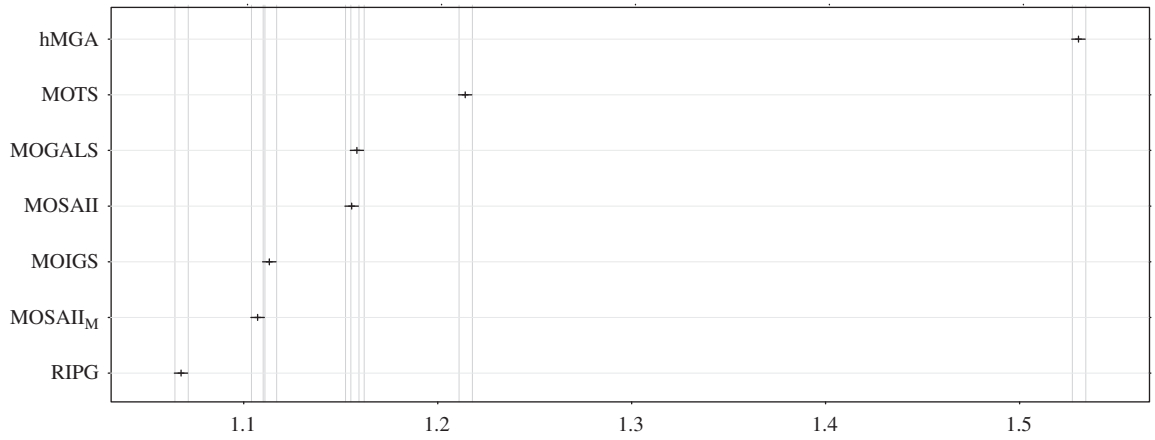

Fig. 8. Results of ANOVA test for epsilon indicator and $t=200$ for makespan and total flowtime objectives with 99% Tukey intervals (95% adjusted confidence level).

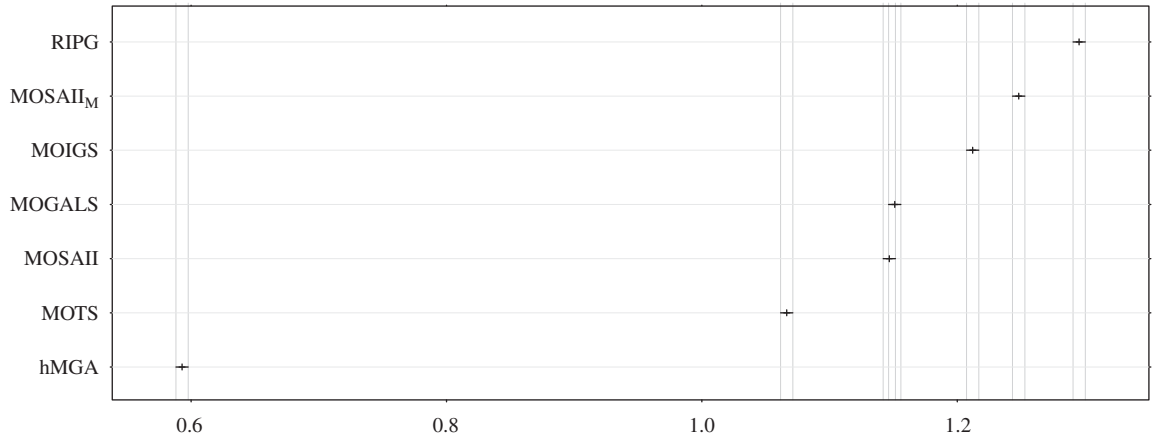

Fig. 9. Results of ANOVA test for hypervolume indicator and $t=200$ for makespan and total flowtime objectives with 99% Tukey intervals (95% adjusted confidence level).

gives results for $t=100$ and I_H. Both figures refer to makespan and total flowtime criteria combination. Similarly, Figs. 8 and 9 display the results for the same experiment configuration for the stopping time of $t=200$. As we can see from the results, apart from some minor exceptions, all tested algorithms are shown to be statistically different in all tests, with RIPG showing clearly superior results in all cases.

The previous results are total averages, it is also interesting to study the behaviour of the different methods according to instance size. Fig. 10 shows the different performance results obtained at each instance group. In small instances, all the algorithms give similar results. However, for medium and large instances, we can see how the RIPG performs much better. Also MOIGS deteriorates in the largest instances.

4.2.2. Makespan and total tardiness results

Table 3 presents results for the makespan and total tardiness criteria. The RIPG algorithm turns out to be again the best performer in terms of I_H and I_ε^1, the second one in the ranking is MOSAII$_M$ while the worst is hMGA. The same conclusions can be drawn from the statistical tests whose results are displayed in Figs. 11–14. As we can see, most plots indicate that there are strong and statistically

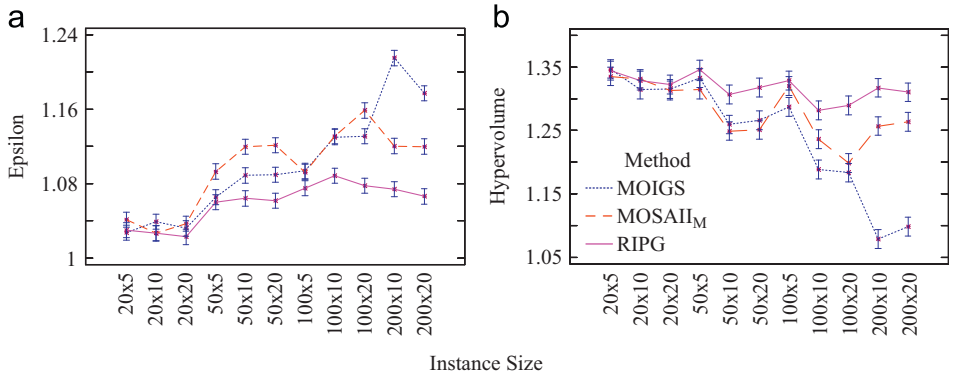

Fig. 10. Means plot and 99% Tukey intervals for the epsilon (a) and hypervolume (b) indicators in the ANOVA experiment for all algorithms against instance size. Makespan and total flowtime objectives.

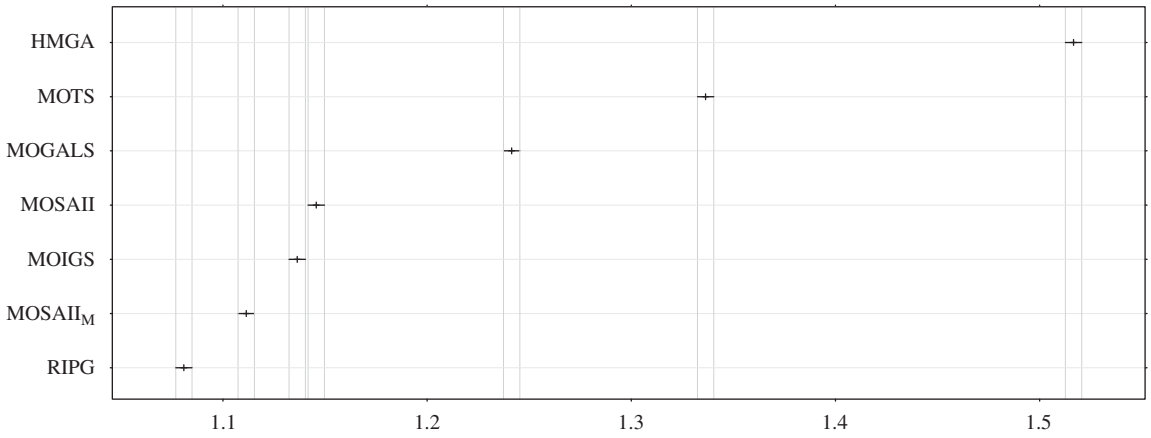

Fig. 11. Results of ANOVA test for epsilon indicator and $t=100$ for makespan and total tardiness objectives with 99% Tukey intervals (95% adjusted confidence level).

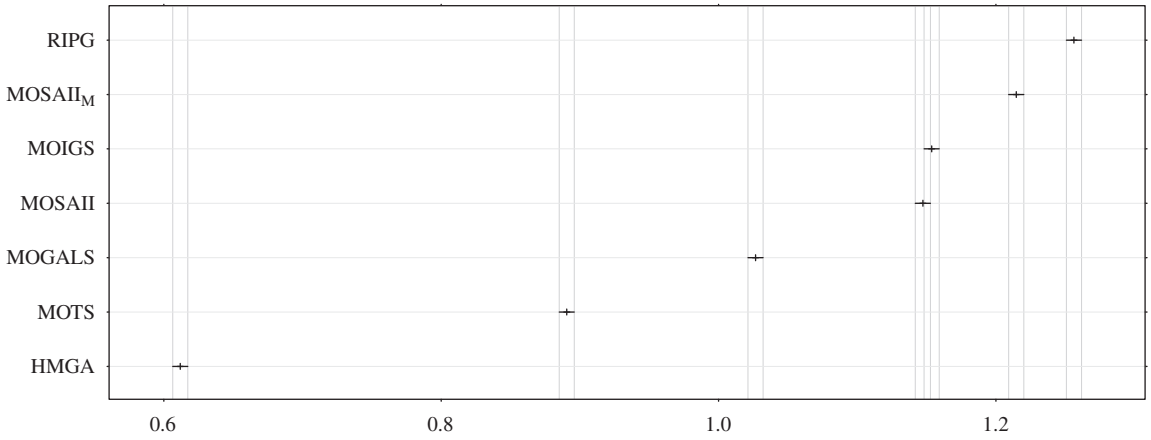

Fig. 12. Results of ANOVA test for hypervolume indicator and $t=100$ for makespan and total tardiness objectives with 99% Tukey intervals (95% adjusted confidence level).

significant differences between the algorithms. It has to be noted though that for this combination of objectives, the differences are more acute. Similar to the previous experiment, we also show here the performance of the algorithms against instance size. Fig. 15 shows the corresponding plot. It is interesting to note that all three methods behave very similarly than with the previous experiment with makespan and total flowtime.

4.3. *Differential Empirical Attainment Functions*

The comparison of algorithms by means of unary indicators is objectively limited because a large piece of information about the behaviour of the methods in the objectives space is lost. Hence, although Pareto-compliant unary indicators tell us whether a frontier (weakly) dominates or is dominated by another one, there is no way to infer in which part of the objective space each algorithm performs better. The Attainment function is a possible answer for this issue. It describes the probability for an algorithm to generate, in a single run, a Pareto approximation set that dominates an arbitrary point in the space of the objectives. The following formal description of Attainment functions is obtained from Grunert da Fonseca et al. [28]. Let $x \in \mathbb{R}^d$ be an arbitrary vector and $SS = \{s_j \in \mathbb{R}^d, j = 1, \ldots, M\}$ a solution set made of non-dominated elements. The Attainment function is characterised by

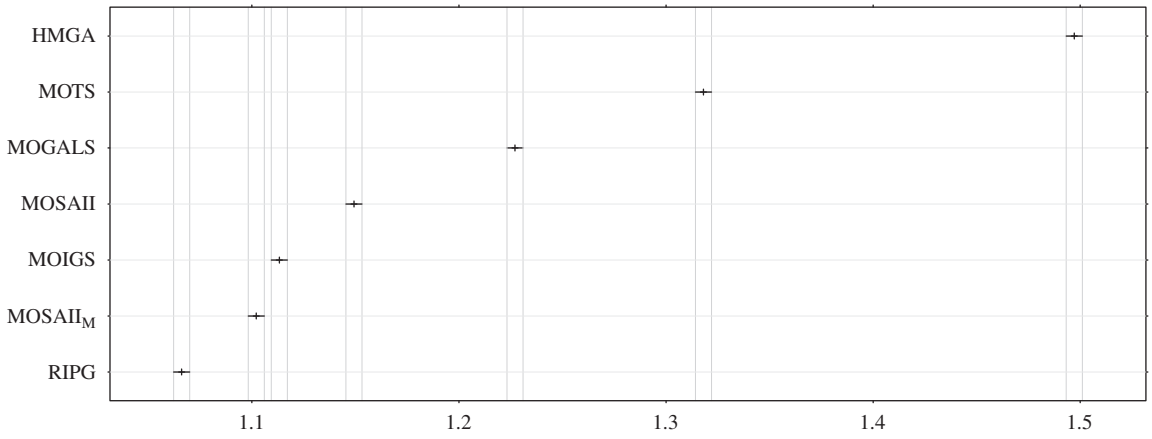

Fig. 13. Results of ANOVA test for epsilon indicator and $t=200$ for makespan and total tardiness objectives with 99% Tukey intervals (95% adjusted confidence level).

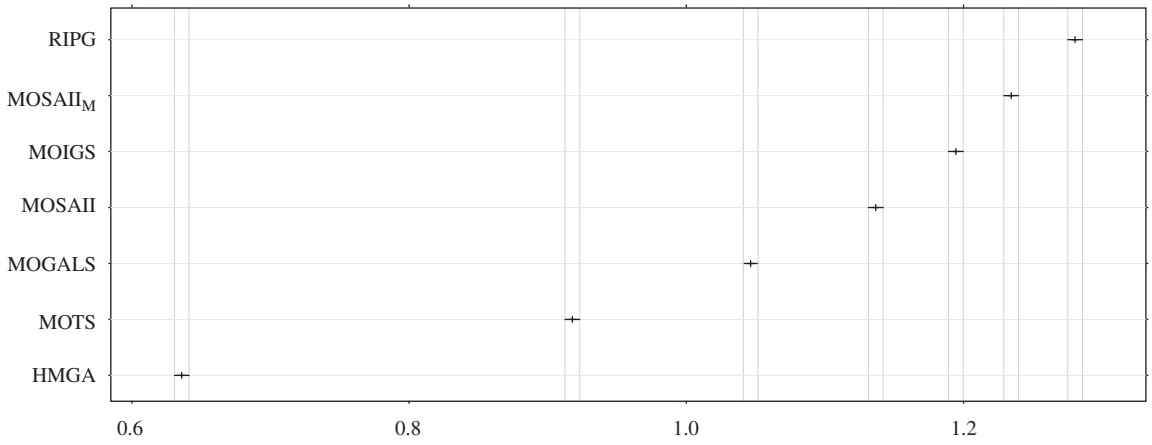

Fig. 14. Results of ANOVA test for hypervolume indicator and $t=200$ for makespan and total tardiness objectives with 99% Tukey intervals (95% adjusted confidence level).

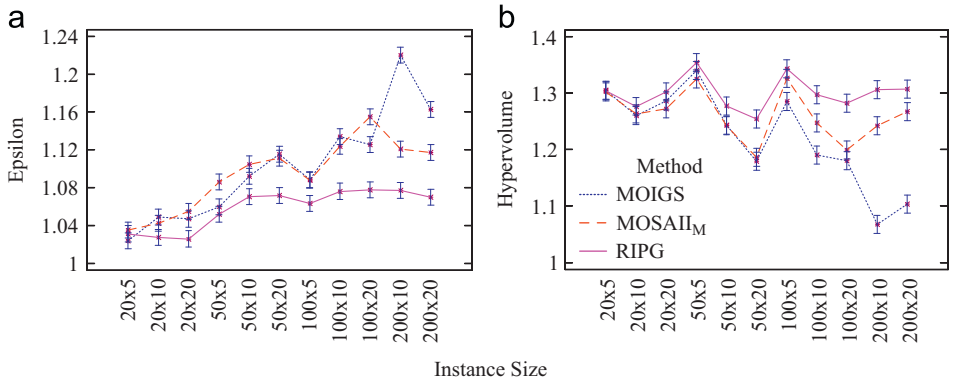

Fig. 15. Means plot and 99% Tukey intervals for the epsilon (a) and hypervolume (b) indicators in the ANOVA experiment for all algorithms against instance size. Makespan and total tardiness objectives.

$AF(x) = P(\exists s_j \in SS: s_j \trianglelefteq x)$ which describes the probability of the algorithm producing, in a single run, at least one solution that weakly dominates x. In the case of stochastic algorithms it is not possible to express this function in a closed form but one can approximate it empirically using the outcomes of several runs. This approximation is called Empirical Attainment Function or EAF and is defined in [28] as

$$EAF(x) = \frac{1}{k}\sum_{i=1}^{k} I(SS_i \trianglelefteq x) \quad (1)$$

where SS_1, SS_2, …, SS_i,…,SS_k represent k Pareto set approximations obtained in k-independent algorithm's runs. In order to complete the analysis of the results presented previously, we compare RIPG against MOSAII$_M$, the best second performer according to our experiment using EAF graphical tests. These tests graphically show, for a certain instance, which area is more likely to be dominated by each algorithm. We use a gradient of colours (blue and red) to indicate zones with high probability to be dominated (intense colours) from zones hardly dominated (light colours). We report here only the EAFs of these two algorithms for two instances. The remaining plots are available

as online material and from http://soa.iti.es. Although EAF is a powerful tool which gives us a spacial description of the statistical behaviour of one algorithm, its major drawback is that it does not allow for a direct comparison between methods. In order to solve such problem, we use the Differential Empirical Attainment Function or Diff-EAF proposed by López-Ibáñez et al. [29]. Diff-EAF is a function that expresses for each point of the solution space the probability to be dominated by only one of the two compared algorithms. This function is obtained by calculating the difference between the EAFs of two different methods. We assign two gradients of colour (blue and red) to positive and negative values of Diff-EAF, respectively. This way, we are able to display

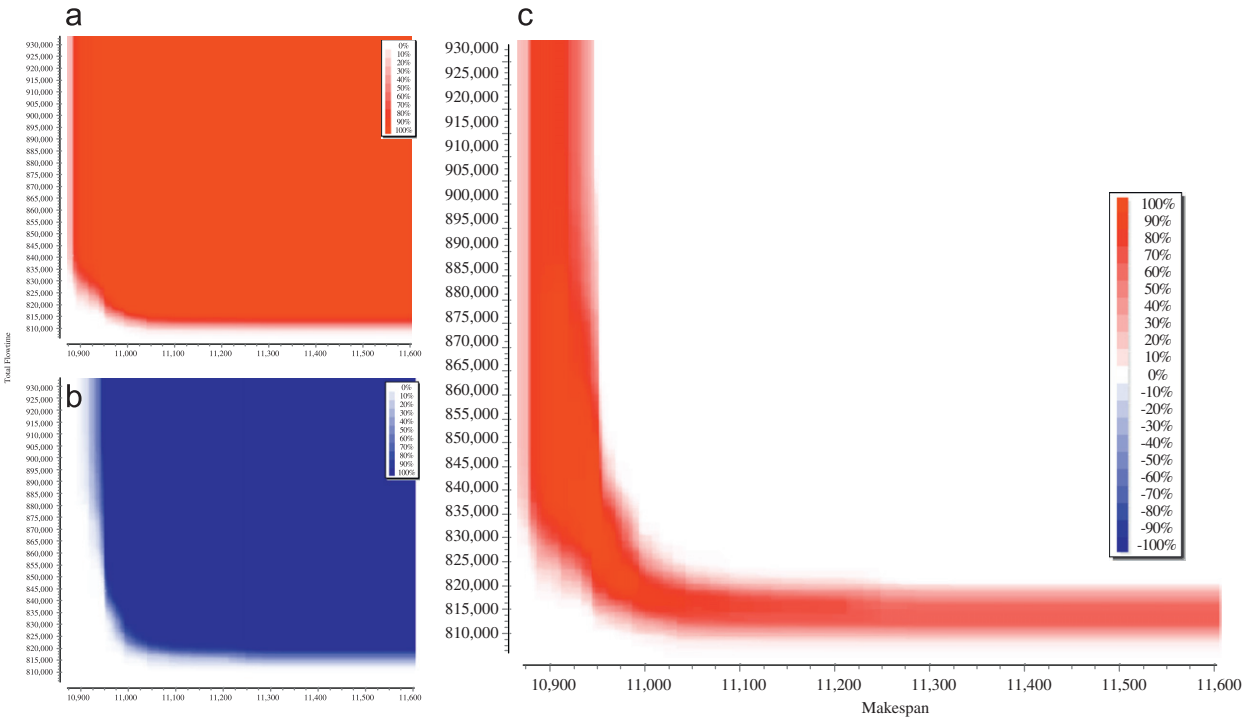

Fig. 16. Attainment function plot. $t=100$ makespan and total flowtime for the MOSAII$_M$ and RIPG algorithms. (a) represents the Attainment function for the RIPG, (b) for the MOSAII$_M$. (c) represents the difference between Attainment functions (a), (b) or Diff-EAF. Instance Ta091 with 200 jobs and 10 machines.

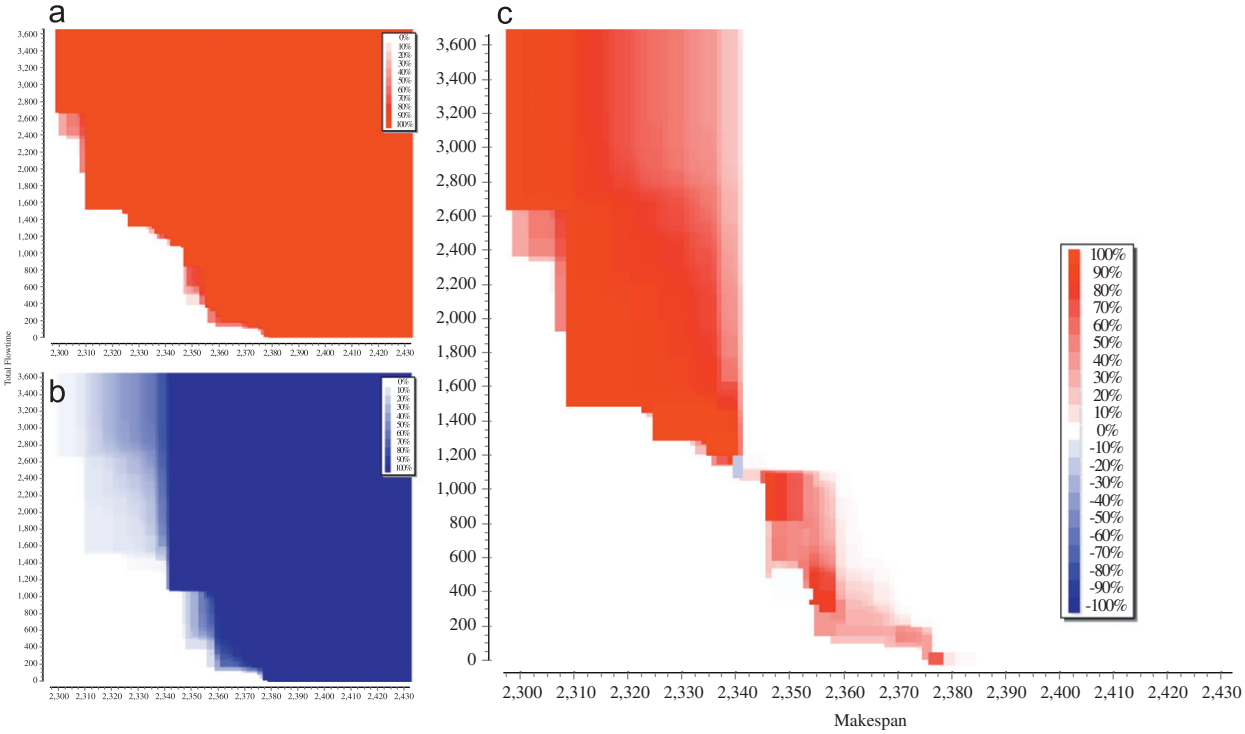

Fig. 17. Attainment function plot. $t=100$ makespan and total flowtime for the MOSAII$_M$ and RIPG algorithms. (a) represents the Attainment function for the RIPG, (b) for the MOSAII$_M$. (c) represents the difference between Attainment functions (a), (b) or Diff-EAF. Instance Ta021 with 20 jobs and 20 machines.

the two EAFs as an image which allows identifying which algorithm prevails in each zone of the objective space. The intensity of the colour assigned for each algorithm shows the probability of dominating a point in the space, of that algorithm over the other algorithm. Points of low colour intensity show lower differences between the algorithms. Notice that white or no colour indicates that algorithms cannot generate solutions dominating this region, or both algorithms have the same probability of generating dominating points and hence the difference is zero. Fig. 16 depicts, as subplots, the two EAFs and the Diff-EAF of RIPG and MOSAII$_M$ calculated after 100 runs of each method over a big size instance with 200 jobs and 10 machines (Ta091), and considering the makespan and total tardiness. The Diff-EAF clearly shows that the RIPG turns out to dominate MOSAII$_M$ all over objective space for this instance. Similarly, Fig. 17 shows a similar plot but in this case for instance Ta021 with 20 jobs and 20 machines.

5. Conclusions and future research

In this work we have presented the Restarted Iterated Pareto Greedy algorithm, devoted to the solution of multi-objective permutation flowshop problems. A comprehensive computational and statistical analysis of each one of its phases was carried out. With this we want to highlight the relevance of a scientific and algorithm engineering approach in designing and developing algorithms. RIPG has been compared against three state-of-the-art performing methods presented in a recent review (see Minella et al. [12]) and against two newly proposed algorithms.

For the computational experiments we have employed two Pareto-compliant performance indicators, two combinations of three commonly used objectives in flowshop problems and two stopping criteria. We have also used a new type of graphical analysis tool strictly related to EAF, the Differential Empirical Attainment Function, that allows the direct comparison of two algorithms. In all computational tests, all performance measures, all objective combinations and stopping times, the proposed RIPG algorithm clearly yields better results and many times in a significant way. The closest competing algorithm, the MOSAII$_M$ has been shown in the Diff-EAF plots to be dominated by RIPG in all regions of the objective space, specially in the largest instances.

Future work stems from the consideration of other problem characteristics like the presence of parallel machines inside one of the more stages of the flow-line (hybrid flowshops), the use of setup times or precedence constraints are other clear examples. Rich single machine problems have not been that much studied from a Pareto perspective, as in Soroush [32] or Valente and Schaller [33]. Another possible research direction consists in improving and generalising the Diff-EAF idea in such a way it could take into account a whole set of instances instead of just one.

Acknowledgements

The authors are indebted to the anonymous referees for their helpful comments which have helped in improving an earlier version of this manuscript. This work is partially 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. 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.

References

[1] 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.
[2] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 2005;165(2):479–94.
[3] Hejazi SR, Saghafian S. Flowshop-scheduling problems with makespan criterion: a review. International Journal of Production Research 2005;43(14):2895–929.
[4] Gupta JND, Stafford Jr EF. Flowshop scheduling research after five decades. European Journal of Operational Research 2006;169(3):699–711.
[5] 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.
[6] El-Bouri A, Balakrishnan S, Popplewell N. A neural network to enhance local search in the permutation flowshop. Computers & Industrial Engineering 2005;49(1):182–96.
[7] Rajendran C, Ziegler H. Two ant-colony algorithms for minimizing total flowtime in permutation flowshops. Computers & Industrial Engineering 2005;48(4):789–97.
[8] 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.
[9] Zitzler E, Thiele L, Laumanns M, Fonseca CM, Grunert da Fonseca V. Performance assessment of multiobjective optimizers: an analysis and review. IEEE Transactions on Evolutionary Computation 2003;7(2):117–32.
[10] Paquete LF. Stochastic local search algorithms for multiobjective combinatorial optimization: method and analysis. PhD thesis, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany; 2005.
[11] Knowles J, Thiele L, Zitzler E. A tutorial on the performance assessment of stochastic multiobjective optimizers. Technical Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland, revised version; 2006.
[12] Minella G, Ruiz R, Ciavotta M. A review and evaluation of multi-objective algorithms for the flowshop scheduling problem. Informs Journal on Computing 2008;20(3):451–71.
[13] Varadharajan T, Rajendran C. A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. European Journal of Operational Research 2005;167(3):772–95.
[14] Arroyo JEC, Armentano VA. Genetic local search for multi-objective flowshop scheduling problems. European Journal of Operational Research 2005;167(3):717–38.
[15] Armentano VA, Arroyo JEC. An application of a multi-objective tabu search algorithm to a bicriteria flowshop problem. Journal of Heuristics 2004;10(5):463–81.
[16] Framinan JM, Leisten R. A multi-objective iterated greedy search for flowshop scheduling with makespan and flowtime criteria. OR Spectrum 2008;30:787–804.
[17] Yandra, Tamura H. A new multiobjective genetic algorithm with heterogeneous population for solving flowshop scheduling problems. International Journal of Computer Integrated Manufacturing 2007;20(5):465–77.
[18] Nawaz M, Enscore Jr EE, Ham I. A heuristic algorithm for the *m* machine, *n* job flowshop sequencing problem. Omega-International Journal of Management Science 1983;11(1):91–5.
[19] Ruiz R, Stützle T. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European Journal of Operational Research 2008;187(3):1143–59.
[20] Ying K-C. An iterated greedy heuristic for multistage hybrid flowshop scheduling problems with multiprocessor tasks. IEEE Transactions on Evolutionary Computation 2008;60(6):810–7.
[21] Toyama F, Shoji K, Miyamichi J. An iterated greedy algorithm for the node placement problem in bidirectional manhattan street networks. In: GECCO '08: Proceedings of the 10th annual conference on genetic and evolutionary computation. New York, NY, USA: ACM; 2008. p. 579–84.
[22] Zhi Y, Armin F, Henning H, Prasanna B, Thomas S, Michael S. Iterated greedy algorithms for a real-world cyclic train scheduling problem. In: Hybrid metaheuristics. Berlin: Springer; 2008. p. 102–16.
[23] Tasgetiren MF, Pan Q-K, Liang Y-C. A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. Computers & Operations Research 2009;36(6):1900–15.
[24] Rajendran C, Ziegler H. A heuristic for scheduling to minimize the sum of weighted flowtime of jobs in a flowshop with sequence-dependent setup times of jobs. Computers & Industrial Engineering 1997;33(1–2):281–4.

[25] Deb K. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 2002;6(2):182–97.
[26] Zitzler E, Knowles J, Thiele L. Quality assessment of Pareto set approximations. In: Multiobjective optimization: interactive and evolutionary approaches. Berlin, Heidelberg: Springer-Verlag; 2008. p. 373–404.
[27] Zitzler E, Thiele L. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Transactions on Evolutionary Computation 1999;3(4):257–71.
[28] Grunert da Fonseca V, Fonseca CM, Hall AO. Inferential performance assessment of stochastic optimisers and the attainment function. In: Evolutionary multi-criterion optimization, first international conference. Lecture notes in computer science, vol. 1993. Spring-Verlag; 2001. p. 213–25.
[29] López-Ibáñez M, Paquete L, Stützle T. Hybrid population-based algorithms for the bi-objective quadratic assignment problem. Journal of Mathematical Modelling and Algorithms 2006;5(1):111–37.
[30] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64(2):278–85.
[31] Hasija S, Rajendran C. Scheduling in flowshops to minimize total tardiness of jobs. International Journal of Production Research 2004;42(11):2289–301.
[32] Soroush HM. Single-machine scheduling with inserted idle time to minimise a weighted quadratic function of job lateness. European Journal of Industrial Engineering 2010;4(2):131–66.
[33] Valente JMS, Schaller JE. Improved heuristics for the single machine scheduling problem with linear early and quadratic tardy penalties. European Journal of Industrial Engineering 2010;4(1):99–129.