Efficient non-population-based algorithms for the permutation flowshop scheduling problem with makespan minimisation subject to a maximum tardiness

Victor Fernandez-Viagas[1]*, Jose M. Framinan[1]
†

[1] Industrial Management, School of Engineering, University of Seville,

Ave. Descubrimientos s/n, E41092 Seville, Spain, {vfernandezviagas,framinan}@us.es

July 19, 2015

Abstract

This paper focuses on the problem of scheduling jobs in a permutation flowshop with the objective of makespan minimisation subject to a maximum allowed tardiness for the jobs, a problem that combines two desirable manufacturing objectives related to machine utilisation and to customer satisfaction. Although several approximate algorithms have been proposed for this NP-hard problem, none of them can use the excellent speed-up method by Taillard (1990) for makespan minimisation due to the special structure of the problem under consideration. In this paper, several properties of the problem are defined in order to be able to partly apply Taillard's acceleration. This mechanism, together with a novel feasible tabu local search method, allows us to further exploit the structure of solutions of the problem, and are incorporated in two proposed algorithms: a bounded-insertion-based constructive heuristic and an advanced non-population-based algorithm. These algorithms are compared with state-of-the-art algorithms under the same computer conditions. The results show that both algorithms improve existing ones and therefore, constitute the new state-of-art approximate solution procedures for the problem.

Keywords: Scheduling, Flowshop, Heuristics, NEH, PFSP, maximum tardiness, makespan, bounded insertion, non-population algorithm

*Corresponding author. Tel.: +34-954487220.

†This is an Accepted Manuscript of an article published by Elsevier in Computers & Operations Research, Volume 64, Dec 2015, Pages 86-96, available online: http://dx.doi.org/10.1016/j.cor.2015.05.006

1 Introduction

A permutation flow shop is a manufacturing layout in which a set of machines are arranged to be visited by a number of jobs in the same order, assuming that the sequence of the jobs remains the same for all machines. Usual additional hypotheses include the simultaneous availability of all jobs and all machines and deterministic processing times, among others (see e.g. Framinan et al., 2014 for a complete list of assumptions). Several criteria can be established to measure the performance of the different schedules (see e.g. Sun et al., 2011). Among them, the maximum completion time of a sequence or makespan is related to resource usage (see e.g. Ruiz and Maroto, 2005 and Fernandez-Viagas and Framinan, 2014b), while tardiness refers to the delay of the completion time of a job with respect to its committed due date (see e.g. Fernandez-Viagas and Framinan, 2015a and Framinan and Leisten, 2008). Since these are key aspects in manufacturing companies' competitiveness, it seems appropriate to consider both objectives together. Regarding tardiness minimisation, customer due dates may be regarded as 'hard' constraints (i.e. deadlines) in some manufacturing scenarios, while in others some flexibility is allowed by the customer as long as the deviation from the completion times of the jobs is limited. In contrast, makespan is an intra-company criteria that is related to maximising machine utilisation, which in turns minimises fixed unit costs. Therefore, one option to balance both objectives is to seek the minimisation of the makespan while allowing only a given deviation from the committed due dates, expressed as the maximum tardiness allowed. Note that this problem includes the special case where no deviation from the jobs' due dates is allowed, thus forcing the fulfilment of the committed due dates.

According to the notation by T'Kindt and Billaut (2006), the problem described in the previous paragraph can be denoted as $Fm|prmu|\epsilon(C_{max}/T_{max})$. This problem belongs to the class of ϵ-constrained multi-criteria scheduling problems, and it has been the subject of several research contributions in the last decades. Since the minimisation of any of the individual criteria (either makespan or maximum tardiness) in a flow shop is NP-hard, the research effort has focused on approximate procedures providing good –but not necessarily optimal– solutions in a relative short period of time. In this regard, the works by Daniels and Chambers (1990), Chakravarthy and Rajendran (1999), Framinan and Leisten (2006), and Ruiz and Allahverdi (2009)

develop different heuristics either for the problem, or for general cases of the problem. In this paper, we propose a constructive heuristic and a metaheuristic that exploits the specific structure of solutions of the problem to reduce the search space and to accelerate the evaluation of solutions. Both algorithms improve existing ones by a larger degree and constitute therefore the new state-of-art approximate solution procedures for the problem.

The remainder of the paper is structured as follows: Section 2 describes the problem under consideration and its state-of-the-art. In Section 3, some definitions and properties of the problem are defined. Section 4 is devoted to propose two algorithms (a constructive heuristic and a metaheuristic) which use the properties discussed previously. The algorithms are compared with the (up to now) state-of-the-art algorithms in Section 4 and, finally, conclusions are discussed in Section 5.

2 Problem Statement and State of the Art

In the $Fm|prmu|\epsilon(C_{max}/T_{max})$ problem under study, n jobs have to be scheduled in a flowshop composed of m one-machine stage. The processing time of job l on machine i is defined as p_{il}. Following this notation, given a sequence of jobs $\Pi := (\pi_1, \ldots, \pi_n)$, the processing time of job in position j, π_j, is denoted as $p_{i\pi_j}$.

Analogously, $C_{i\pi_j}(\Pi)$ denotes the completion time of job π_j on machine i according to the schedule given by Π. The makespan of the sequence Π is given by the completion time of last job in last machine, $C_{m,\pi_n}(\Pi)$, which is denoted as $C_{max}(\Pi)$. Whenever it does not lead to confusion, the sequence Π is omitted in the notation of the completion times and makespan.

In a similar manner, if d_{π_j} is the due date of job π_j, its tardiness is defined as $T_{\pi_j}(\Pi) = \max\{C_{m\pi_j}(\Pi) - d_{\pi_j}, 0\}$ and the maximum tardiness of the sequence Π as $T_{max}(\Pi) = \max_{j=1,\ldots,n}\{T_{\pi_j}(\Pi)\}$. As with the makespan, Π is omitted when it is clear from the context. The goal of the problem is to find a schedule Π for which the makespan is minimum subject to $T_{max}(\Pi) \leq \epsilon$.

As mentioned in Section 1, the problem is NP-hard since the minimisation of each individual criterion is already an NP-hard problem for the permutation flowshop (see e.g. T'Kindt and Billaut, 2006 for a detailed proof). Consequently, the interest lies in finding efficient approximate methods

or *heuristics*. Given the clear connection between our problem and that of makespan minimisation, most of the algorithms to solve the problem are based on the best heuristic for makespan minimisation: i.e. the NEH heuristic by Nawaz et al. (1983). It is then useful to recall the main steps in the NEH heuristic, which can be described as follows:

1. Jobs are ordered according to non-increasing sum of processing times.
2. A partial sequence is constructed only with first job of the previous phase.
3. Each remaining job of initial phase is iteratively inserted in all positions of the partial sequence. The makespan of all these sequences is evaluated, and the partial schedule for which the lowest makespan is reached is selected for the next iteration.
4. The procedure is repeated until no more jobs are available.

The above steps make clear that the computational burden of the NEH lies on the evaluation of all possible insertions in Step 3. In Taillard (1990), a mechanism –named in the following *Taillard's acceleration*– is proposed so the computational complexity of evaluating all insertions is equivalent to that of evaluating one sequence. In order to explain Taillard's acceleration, let us first define three variables (for a more detailed description of the variables, see Taillard, 1990):

- e_{i,π_j}: Earliest completion time of job in position j in machine i.
- q_{i,π_j}. Once a sequence of jobs has been defined (and therefore the makespan of this sequence is obtained), q_{i,π_j} is the difference between the makespan and the latest starting time of job in position j in machine i.
- f_{i,π_j}. Earliest completion time of the new job σ when it is inserted before job in position j in machine i. These are computed using e_{i,π_j} and the processing times of σ.

By means of these variables, the partial makespan C^j_{max} when introducing job σ before job in position j can be determined using the following expression:

$$C^j_{max} = \max_i (f_{i,\pi_j} + q_{i,\pi_j}) \tag{1}$$

As the job σ is inserted in the position with minimum makespan, the makespan of the sequence is defined by:

$$C_{max} = \min_{j}(C^{j}_{max}) \tag{2}$$

As it can be seen, although the cost of evaluating the insertion slot with lowest makespan is greatly reduced by Taillard's acceleration, the completion time of each job cannot be obtained using this mechanism, and therefore its tardiness cannot be computed. As a consequence, none of the heuristics proposed up-to-now in the literature for the problem under consideration use this mechanism.

Among the contributions on the $Fm|prmu|\epsilon(C_{max}/T_{max})$ problem, Daniels and Chambers (1990) were the first in proposing a constructive heuristic. In their heuristic, assuming a partial sequence Π formed by already scheduled jobs, a (partial) sequence is constructed for each non-scheduled job u_k by placing it as the first job, and then scheduling the jobs in Π after u_k according to the NEH algorithm. Out of these so-obtained sequences, the one with the lowest makespan is chosen for the next iterations (consequently, u_k is removed from the non-scheduled jobs set for the next iteration).

Chakravarthy and Rajendran (1999) propose a simulated annealing algorithm to solve the $Fm|prmu|\epsilon(Z/T_{max})$ where $Z = \lambda \cdot C_{max} + (1-\lambda) \cdot T_{max}$, $\lambda \in [0,1]$. Clearly, our problem is a special case of their problem when $\lambda = 1$. Their algorithm begins with the best sequence among the solutions found by the NEH heuristic, the earliest due date rule and the least slack rule (jobs ordered according to ascending order of $d_j - \sum_{i=1}^{m} p_{ij}$). The procedure iteratively samples neighbour solutions (using an adjacent pairwise interchange neighbourhood) until the stopping criterion is fulfilled.

Framinan and Leisten (2006) propose a constructive heuristic, denoted in the following as FL, based on the NEH algorithm to solve the $Fm|prmu|\epsilon(C_{max}/T_{max})$ problem. The heuristic tries to improve the makespan without worsening the tardiness by using a property of the problem. The heuristic is compared with those of Daniels and Chambers (1990) and Chakravarthy and Rajendran (1999) for small and big instances. The results show that the FL outperforms the other ones in

terms of both the quality of the solutions and the number of the feasible solutions obtained.

Finally, Ruiz and Allahverdi (2009) propose an iterated optimization algorithm to solve the $Fm|prmu|\epsilon(Z/T_{max})$ problem. More specifically, they proposed a high-performance Genetic Algorithm (GA in the following) where the selection procedure is based on n-tournament (see Ruiz and Allahverdi, 2007). The fitness values of the individuals are calculated depending on whether all individuals are feasible; feasible and infeasible; or only infeasible. The algorithm outperforms the FL for the $Fm|prmu|\epsilon(Z/T_{max})$ problem in an extended benchmark. Nevertheless, GA and FL were not compared for the specific $Fm|prmu|\epsilon(C_{max}/T_{max})$ problem.

To summarise the state of the art regarding the problem under consideration, there are some efficient heuristics for the problem, but their performance is not completely clear, as the comparison between the most efficient contributions (i.e. GA and FL) has been only partially conducted. In addition, both mechanisms made extensive use of insertion neighbourhoods, so it could be extremely interesting to devise a mechanism similar to that by Taillard to reduce the computational burden. Finally, it is also to note that all existing procedures make little use (or no use at all) of the knowledge on the problem domain.

3 Problem Properties

As mentioned in Section 2, Taillard's acceleration does not compute the completion times of each job and therefore, it cannot be used to compute the maximum tardiness of the sequence. Indeed, our problem is complicated by the fact that, when inserting a new job σ in position r of an existing partial sequence, an infeasible solution can be obtained due to either the increase in the completion times of the jobs after σ, or due to the completion time of job σ itself. In order to further classify these two possibilities, let us introduce the following definitions:

Definition 3.1 (First Feasible Position)**.** *Given a feasible (partial) sequence* $\Pi := (\pi_1, \ldots, \pi_k)$*, and a non scheduled job* σ*, let* $\Pi'_r := (\pi_1, \ldots, \pi_{r-1}, \sigma, \pi_r, \ldots, \pi_k)$ *be the (partial) sequence obtained by the insertion of* σ *in position* r *of* Π*. Then, the First Feasible Position (FFP) is defined as follows:*

$$FFP(\Pi, \sigma) := \underset{1 \leq r \leq k+1}{\arg\min} \{T_{\pi_j}(\Pi'_r) \leq \epsilon \quad \forall j = r, \ldots, k\}$$

As it can be seen from the definition, FFP is the lowest position where a new job can be inserted in an existing sequence without causing infeasible due dates in any of the jobs resulting in positions later than the insertion point. It is clear that, in a given instance of the problem and a partial sequence Π, it is not possible to obtain feasible schedules by inserting a non scheduled job σ into a position $j < FFP(\Pi, \sigma)$.

Note also that obtaining FFP for a tuple Π and σ does not guarantee that $\Pi_j^{'}$ is feasible for $j \geq FFP$, since the computation of FFP does not take into account the potential infeasibility caused by job σ.

Definition 3.2 (Last Feasible Position). *Given a feasible (partial) sequence* $\Pi := (\pi_1, \ldots, \pi_k)$*, and a non scheduled job* σ*, let* $\Pi_r^{'} := (\pi_1, \ldots, \pi_{r-1}, \sigma, \pi_r, \ldots, \pi_k)$ *be the (partial) sequence obtained by the insertion of* σ *in position* r *of* Π*. Then, the Last Feasible Position (LFP) is defined as follows:*

$$LFP(\Pi, \sigma) := \underset{1 \leq r \leq k+1}{\arg\max} \{T_\sigma(\Pi_r^{'}) \leq \epsilon\}$$

In this manner LFP is the highest position r where job π_r can be inserted without making its completion time infeasible. Note that the feasibility of the jobs in positions $r+1, r+2, \ldots$ is not considered when computing LFP.

The calculation of both limits is of interest due to some straightforward observations which follow from both definitions:

1. If $FFP(\Pi, \sigma) > LFP(\Pi, \sigma)$ for a given tuple Π and σ, then no feasible sequence can be obtained by inserting σ into Π.

2. If $FFP(\Pi, \sigma) \leq LFP(\Pi, \sigma)$ for a given tuple Π and σ, then the sequence obtained by inserting σ in position FFP of Π is feasible.

3. If $FFP(\Pi, \sigma) \leq LFP(\Pi, \sigma)$ for a given tuple Π and σ, then at least one feasible sequence can be obtained by inserting σ in positions between FFP and LFP, inclusive.

4. For a given tuple Π and σ with $FFP(\Pi, \sigma) \leq LFP(\Pi, \sigma)$, the set of feasible sequences obtained by inserting σ in positions between FFP and LFP represent *all* feasible sequences

that can be obtained by inserting σ into Π.

Once FFP and LFP are obtained, the sequence with the lowest makespan can be computed by using Taillard's acceleration between both bounds, i.e.:

$$C_{max} = \min_j(C^j_{max}) \quad j = FFP, \ldots, LFP \tag{3}$$

where C_{max} is obtained using Expressions (1) and (2). Note that the so-found sequence is not necessarily feasible. The advantage of this mechanism lies in speeding up the computations.

In view of the above expressions, the challenge now is to compute both FFP and LFP in an efficient manner. To do so, we introduce the following properties:

Property 3.1. *Given an instance of the $Fm|prmu|\epsilon(C_{max}/T_{max})$ problem, and given a tuple Π and σ of a partial sequence and a non-scheduled job respectively, then*

$$\underline{FFP}(\Pi, \sigma) := 1 + \arg\max_j\{C_{m\pi_j} + \min_{1\leq i\leq m} p_{i\sigma} - d_{\pi_j} > \epsilon\}$$

is a lower bound for $FFP(\Pi, \sigma)$. Furthermore, $\underline{FFP}$ can be computed in $O(n \cdot m)$

Proof. Recall that the completion times of the jobs in Π placed after the insertion of a job σ must increase at least $\min_i(p_{i,\sigma}), i \in [1, \cdots, m]$ (see also Fernandez-Viagas and Framinan, 2014a). Therefore, $C_{m\pi_j} + \min_{1\leq i\leq m} p_{i\sigma}$ is a lower bound of completion time of job in position j after the insertion of σ in position $r < j$. As a consequence, if $C_{m\pi_j} + \min_{1\leq i\leq m} p_{i\sigma} - d_{\pi_j} > \epsilon$, then the due date of job in position j is always infeasible, so $\underline{FFP}$ is a lower bound of FFP.

$\underline{FFP}$ can be computed in two steps: First $M = \min_{1\leq i\leq m} p_{i\sigma}$ is computed in $O(m)$. Next, the expression $E_j = C_{m,\pi_j} + M - d_{\pi_j}$ is computed in $O(n \cdot m)$ for $j = 1, 2, \ldots$ until it verifies that $E_j > \epsilon$. It is thus clear that the computation of $\underline{FFP}$ is $O(n \cdot m)$. □

Property 3.2. *Given an instance of the $Fm|prmu|\epsilon(C_{max}/T_{max})$ problem, and given a tuple Π and σ of a partial sequence and a non-scheduled job respectively, then LFP can be computed in $O(n \cdot m)$.*

Proof. First e_{i,π_j} the earliest completion times of the jobs before σ are computed in $O(n \cdot m)$ (see Section 2). Then, for a position k where σ can be inserted, the completion time of σ is computed

in $O(m)$ by adding the processing times of σ to e_{i,π_j}, and the result is compared to ϵ. Since this comparison is performed for all positions prior to the candidate position where the new jobs is to be inserted, it is clear that LFP can be computed in $O(n \cdot m)$. The detailed pseudo code is presented in Figure 2. □

Equipped with these problem properties, in Section 4 we propose efficient approximate procedures based on the insertion of jobs into existing partial schedules. More specifically, in Section 4.1 we present a constructive heuristic for the problem whereas in Section 4.2 we present a non-population based metaheuristic.

4 Proposed Algorithms

In this section, we propose two new approximate algorithms to solve the $Fm|prmu|\epsilon(C_{max}/T_{max})$ problem. The first one is a constructive heuristic based on a so-called *bounded insertion* using some properties of the problem, while the second one is a generic non-population-based algorithm searching for feasible solutions in each iteration that uses the proposed constructive heuristic as starting solution. In this manner, we attempt to provide efficient approximate solutions for a wide range of decision time intervals. To speed up the computation times, both algorithms use the problem properties presented in Section 3.

4.1 Bounded-Insertion-Based Constructive Heuristic, $BICH$

In this Section, we present a constructive heuristic based on a bounded insertion (BICH) that also repairs infeasibility by means of a tabu local search in each iteration (see pseudo code in Figure 1).

More specifically, the algorithm obtains a sequence $\Pi := (\pi_1, \ldots, \pi_n)$ in the following manner: Initially, jobs are sorted in non ascending order of the sum of their processing times, so a sorted sequence $\alpha := (\alpha_1, \ldots, \alpha_n)$ is obtained. The first job in the sorted sequence is also the first job in Π, i.e. $\pi_1 = \alpha_1$. Then, the remaining jobs in α are inserted in Π one by one in the following manner: in iteration k ($k \in [2, n]$), job α_k is removed from α and the following steps are carried out:

- **Compute $e_{i\pi_j}, q_{i\pi_j}$ and $f_{i\pi_j}$.** In this step, the variables required to apply Taillard's acceleration are calculated here according to the expressions described in Section 2. These computations can be implemented in $O(n \cdot m)$.

- **Compute $\underline{FFP}$.** In this step, $\underline{FFP}(\Pi, \alpha_k)$ is obtained according to Property 3.1.

- **Compute LFP.** In this step, $LFP(\Pi, \alpha_k)$ is obtained according to Property 3.2.

- **Obtaining the best makespan between $\underline{FFP}$ and LFP inclusive.** If $\underline{FFP} \leq LFP$, a set of schedules can be obtained when inserting α_k between these two indices. To select the position of insertion in Π, Taillard's acceleration is employed as described in Section 3. If $\underline{FFP} > LFP$, α_k is inserted in position LFP. Note that this does not necessarily mean that the so-obtained partial sequence is infeasible, as $\underline{FFP}$ is a lower bound for FFP.

- **Repairing infeasible solutions**. If the resulting partial sequence Π is infeasible, a Feasible Tabu Search (FTS) procedure is performed to try to get to a feasible solution. The FTS is an iterative procedure which maintains the idea of insertion between $\underline{FFP}$ and LFP. First, for each iteration, infeasible jobs are removed from the partial sequence Π and are randomly ordered. Then, they are successively inserted one by one between the $\underline{FFP}$ and LFP indices (inclusive), but in this case the feasibility of each so-obtained sequence is checked, so Taillard's acceleration cannot be used. Furthermore, a simple tabu procedure is introduced to avoid cycles: Each job has a tabu list of positions previously chosen. Once an infeasible job is inserted in a position, this position is added to the tabu list of such job. Thereby, when a job is to be inserted in the actual sequence, only positions that are not in its tabu list can be chosen. The tabu lists of all jobs are set to zero if the infeasible jobs of the current iteration are different from the previous infeasible jobs. Only in this case jobs from the tabu list can be removed from the lists. The procedure finishes when either there are no more infeasible jobs, or when more than x iterations have been run and the number of infeasible jobs has not decreased during the last iteration. The length of the tabu list is sufficiently long to allow storing each possible position (here the maximum number of iterations of the search method i.e. x). Furthermore, when the output sequence of this procedure is infeasible, the FTS is not further implemented in the rest of iterations of the

Procedure *BICH(x)*

$\alpha :=$ Jobs ordered by descending sums of processing times where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$;
$\Pi := \{\alpha_1\}$;
$flag :=$ true;
for $k = 2$ **to** n **do**
  Calculate $e_{i\pi_{j_1}}, q_{i\pi_{j_1}}$ and $f_{i\pi_{j_2}}$ for $i = 1 \ldots m$, $j_1 = 1 \ldots k$ and $j_2 = 1 \ldots k+1$;
  for $j = 1$ **to** k **do**
    if $e_{m,\pi_j} + \min_{i=1\ldots m}(p_{i\alpha_k}) - d_{\pi_j} > \varepsilon$ **then**
      $\underline{FFP} = j + 1$;
    end
  end
  $LFP := CalculateExactlyLFP(\Pi, \alpha_k, k, \{e_{i\pi_{j_1}}\})$;
  Test job α_k between the positions $\underline{FFP}$ and LFP of Π;
  $\Pi :=$ permutation obtained by inserting α_k in the position $j \in [\underline{FFP}, LFP]$ of Π with lowest makespan using Taillard's Acceleration (note that infeasible permutations are allowed here as feasibility is not checked);
  if $flag =$ *true* **then**
    $\Pi := FeasibleTabuSearch(\Pi, x)$;
    if Π *is infeasible* **then**
      $flag :=$ false;
    end
  end
end
end

Figure 1: *BICH*

BICH heuristic.

The pseudo code of the *FTS* procedure is shown in Figure 3.

4.2 Advanced Non-Population-Based Algorithm, *ANPA*

In this Section, an Advanced Non-Population-based Algorithm (*ANPA*) is proposed as an extension of the ideas presented in the constructive heuristic *BICH*. The algorithm tries to improve the solution by iteratively performing greedy methods and local search methods. The global procedure of the algorithm is shown in Figure 4.

ANPA starts with the sequence obtained by the heuristic *BICH* and tries to improve it by means of a bounded relative local search (denoted as *BRLS*) explained below in more detail. Then, the following phases are repeated until the stopping criterion is reached:

```
Function CalculateExactlyLFP(π, NewJob, k, {e_{iπ_j}})
    flag := false;
    LFP := 0;
    C_0 := 0;
    for i = 1 to m do
        C_0 = C_0 + p_{i,NewJob};
    end
    if C_0 − d_{NewJob} > ε then
        flag := true;
    end
    j := 1
    while j < k and flag = false do
        C_j := 0;
        for i = 1 to m do
            C_j = max(C_j, e_{i,π_j}) + p_{i,NewJob}
        end
        if C_j − d_{NewJob} > ε then
            flag := true;
        else
            LFP++
        end
        j++;
    end
    return LFP;
end
```

Figure 2: Procedure CalculateExactlyLFP

Procedure *FeasibleTabuSearch*(π, x)
  $\gamma :=$ Infeasible jobs of Π;
  $n_\gamma :=$ Number of infeasible jobs in Π, $|\gamma|$;
  $n_\gamma^{old} := n_\gamma + 1$
  $\#Iterations = 1$;
  while n_γ *!=0* **and** *(*$n_\gamma < n_\gamma^{old}$ **or** $\#Iterations <= x$*)* **do**
    $\Pi :=$ Extract jobs γ of Π;
    $\gamma :=$ Randomly order infeasible jobs γ;
    if *Infeasible jobs are different from last iteration* **then**
      Empty tabu list;
    end
    for $k = n_\gamma$ **to** 1 **do**
      for $j = 1$ **to** k **do**
        if $e_{M,\pi_j} + \min_i(p_{i\gamma_k}) - d_{\pi_j} > \varepsilon$ **then**
          $\underline{FFP} = j + 1$;
        end
      end
      $LFP := CalculateExactlyLFP(\Pi, \gamma_k, k, e_{i\Pi_j})$;
      Test job γ_k in the feasible and non-tabu positions between $\underline{FFP}$ and LFP of Π and denote bj the position with the lowest makespan;
      $\Pi :=$ permutation obtained by inserting γ_k in bj;
      Add position bj to the tabu list of job γ_k;
    end
    $n_\gamma^{old} := n_\gamma$
    $\gamma :=$ Infeasible jobs of Π;
    $n_\gamma = |\gamma|$;
    $\#Iterations++$;
  end
end

Figure 3: Procedure FeasibleTabuSearch

Procedure *ANPA*(d, x, T)
  $(\Pi, C_{max}) = BICH(x)$;
  $\Pi = BRLS(\Pi, C_{max})$;
  while *stopping criterion is not reach* **do**
    $\Pi^1 = \Pi$;
    $\Pi^1 :=$ randomly remove d jobs from Π^1 and insert it in Π^D;
    $(\Pi^2, C_{max}) := ConstructionPhase(\Pi^D, \Pi^1)$;
    $\Pi^2 := BRLS(\Pi^2, C_{max})$;
    $\Pi^2 := FeasibleTabuSearch(\Pi^2, x)$;
    $\Pi := SimulatedAnnealingCriterion(\Pi^2, T)$;
  end
end

Figure 4: *ANPA*

Procedure *ConstructionPhase*(Π^D, Π)
  for $k = n - d + 1$ **to** n **do**
    Calculate $e_{i\pi_{j_1}}, q_{i\pi_{j_1}}$ and $f_{i\pi_{j_2}}$ for $i = 1 \dots m$, $j_1 = 1 \dots k$ and $j_2 = 1 \dots k+1$;
    for $j = 1$ **to** k **do**
      if $e_{m,\pi_j} + \min_i(p_{i\pi^D[k-(n-d)]}) - d_{\pi_j} > \varepsilon$ **then**
        $\underline{FFP} = j + 1$;
      end
    end
    $LFP := CalculateExactlyLFP(\pi, \pi^D_{k-(n-d)}, k, \{e_{i\pi_{j_1}}\})$;
    Test job $\pi^D_{k-(n-d)}$ between the positions $\underline{FFP}$ and LFP of Π;
    $\Pi :=$ permutation obtained by inserting $\pi^D_{k-(n-d)}$ in the position $j \in [\underline{FFP}, LFP]$ of Π with the lowest makespan using Taillard's Acceleration (note that infeasible permutations are allowed here as feasibility is not checked);
  end
end

Figure 5: ConstructionPhase

- **Jobs Determination Phase**. In this phase, d jobs are randomly chosen to be removed from the sequence. The set of removed jobs is denoted Π_D.

- **Construction Phase**. Jobs in Π_D are re-inserted one by one in the sequence following a similar procedure as in the $BICH$ heuristic, but without applying the FTS procedure after each insertion. This phase is explained in detail in Figure 5.

- **Bounded RLS (BRLS)**. The solution of the previous phase is improved by a relative local search method. One by one, jobs are removed from the sequence, tried to be inserted in each position $j \in [1, n]$ and finally, placed in the position with the lowest makespan using Taillard's acceleration between $\underline{FFP}$ and LFP inclusive. This procedure finishes when n jobs are tried without improving the current best makespan. The pseudo code of this local search method is shown in Figure 6, which is similar to those in Pan et al. (2008) and in Fernandez-Viagas and Framinan (2014a).

- **Feasible Tabu Search**. After the ConstructionPhase and the $BRLS$ procedures, FTS is implemented in order to try to reach feasibility when the solution is infeasible.

- **Simulated Annealing-like Acceptance Criterion**. To add diversification to the algorithm, solutions are kept according to a simple simulated annealing procedure. When a

Procedure $BRLS(\Pi, C_{max})$

$h = 1$;
$i = 1$;
$\Pi^b := \Pi$;
while $i <= n$ **do**
  $k := h \bmod n$;
  $\Pi^0 :=$ remove job π_k from Π;
  Calculate $e_{i\pi_{j_1}}, q_{i\pi_{j_1}}$ and $f_{i\pi_{j_2}}$ for $i = 1 \ldots m$, $j_1 = 1 \ldots k$ and $j_2 = 1 \ldots k+1$;
  for $j = 1$ **to** k **do**
    if $e_{m,\pi_j} + \min_i(p_{i\pi_k}) - d_{\pi_j} > \varepsilon$ **then**
      $\underline{FFP} = j + 1$;
    end
  end
  $LFP := CalculateExactlyLFP(\Pi^0, \pi_k, n-1, \{e_{i\pi_{j_1}}\})$;
  Test job π_k between the positions $\underline{FFP}$ and LFP of Π^0;
  $\Pi :=$ permutation obtained by inserting π_k in the position $j \in [\underline{FFP}, LFP]$ of Π with lowest makespan, C'_{max}, using Taillard's Acceleration (note that infeasible permutations are allowed here as feasibility is not checked);
  if $C'_{max} < C_{max}$ **then**
    $C_{max} = C'_{max}$;
    $i = 1$;
    $\Pi^b := \Pi$;
  else
    $i++$;
  end
  $h++$;
end
return Π^b;
end

Figure 6: Bounded Relative Local Search, $BRLS$

solution, Π_2, is worse than the local search optimum, Π, it is maintained only if:

$$random \leq exp\left\{\frac{-(C_{max}(\Pi^2) - C_{max}(\Pi))}{Temperature}\right\}$$

where *random* is a random number between 0 and 1 and the $Temperature$ is a function that depends on parameter T:

$$Temperature = T \cdot \frac{\sum_{\forall i}\sum_{\forall j} p_{i,j}}{n \cdot m \cdot 10}$$

The temperature parameter has been generated following the suggestions by Osman and Potts (1989) (see e.g. Fernandez-Viagas and Framinan, 2014a, Pan et al., 2008 and Ruiz and Stützle, 2007 for similar approaches).

4.3 Experimental Parameter Tuning

The proposed algorithms use three parameters: T, d, and x. Therefore, it is interesting to investigate the values of these parameters for which the algorithms reach the best performance. Parameter x is used in both $BICH$ and $ANPA$, while the other two parameters are included only in $ANPA$. In order to simplify the experimentation, the three parameters have been tested only for $ANPA$, and the value obtained for parameter x was also chosen for the constructive heuristic $BICH$. The level of parameters tested are:

- $T \in [0.1, 0.2, 0.3, 0.4]$
- $d \in [4, 5, 6, 7]$
- $x \in [10, 20, 30]$

$ANPA$ is tested following the same calibration test as in Vallada and Ruiz (2010), which in turn is based in Vallada et al. (2008). This calibration benchmark can be summarized as follows. The number of jobs and machines is set to $n = \{50, 150, 250, 350\}$, $m = \{10, 30, 50\}$ and due dates are generated according to the procedure employed by Gelders and Sambandam (1978) using an uniform distribution between $P \cdot (1 - T - R/2)$ and $P \cdot (1 - T + R/2)$. Parameters T and P take the

following values in the test: $T = \{0.2, 0.4, 0.6\}$ and $R = \{0.2, 0.6, 1.0\}$. Additionally, processing times are generated according to a uniform distribution between 1 and 99. For each combination of parameters n, m, T and R, two instances are generated summing 216 instances. The stopping criterion adopted is to halt the procedure when the CPU time in milliseconds reaches the value $n \cdot (m/2) \cdot 20$.

To establish statistically significant differences between parameters T, d and x, a non-parametric Kruskal-Wallis test is performed, since the normality and homoscedasticity assumptions required for an analysis of variance were not satisfied. As a result of the test, statistically significant differences between the levels of the parameters x and T were found, but not for d since the significance values were 0.011, 0.000 and 0.870 respectively. The best combination of parameters was found for $d = 5$, $T = 0.4$ and $x = 30$, so these were used in the computational experience carried out in the next section.

5 Computational Results

In this Section, the performance of the proposed algorithms $BICH$ and $ANPA$ is compared with the best algorithms so far for the problem, i.e. the GA by Ruiz and Allahverdi (2009) and the constructive heuristic FL by Framinan and Leisten (2006). Additionally, two efficient heuristics for makespan minimisation (i.e. the NEH heuristic, and the iterated greedy algorithm, IG_{RS_LS}) are included in the comparison as they are two of the most efficient constructive heuristic and iterative improvement algorithm, respectively. The adaptations of these heuristic to our problem are denoted A_NEH and A_IGA, respectively. When adapting both methods to the proposed problem, the following assumptions are adopted:

- The objective function remains the original of these algorithms, i.e. the minimisation of makespan.
- In the insertion phases of the algorithms, only feasible sequences are considered, i.e. the jobs to be inserted are placed in the feasible position with the lowest makespan.
- Taillard's acceleration is removed from both algorithms since the calculation of the tardiness

for each job does not allow its use.

To be able to determine the most efficient algorithms for the problem, all methods have been coded under the same programming language, C#, and under the same computer (an Intel Core i7-3770 with 3.4 GHz and 16GB RAM). The algorithms are tested using the set of instances of the benchmark of Vallada et al. (2008), which includes $n = \{50, 150, 250, 350\}$ jobs and $m = \{10, 30, 50\}$ machines. Processing times are generated following an uniform distribution $[1, 99]$ as well as the due dates are generated according to Gelders and Sambandam (1978) as explained in Section 4.3 with $T = \{0.2, 0.4, 0.6\}$ and $R = \{0.2, 0.6, 1.0\}$. 5 instances are obtained for each combination of n, m, T and R forming a total of 540 instances. Furthermore, in order to increase the accuracy of the iterated improvement algorithms, five runs have been performed per instance and the average values are recorded for the makespan and for the CPU times.

The same stopping criteria as in Ruiz and Allahverdi (2009) are applied for the iterative improvement algorithms. These stopping criteria depend on the size of the instance (i.e. the number of jobs and machines) following the expression $t \cdot n \cdot m \cdot /2$ milliseconds where the values 2, 5, 20 and 60 are tested for the parameter t. The FL and BICH constructive heuristics stop naturally when their final sequences are constructed.

Due to the fact that the problem under consideration is subject to maximum tardiness, the evaluation of the quality of both constructive and iterative algorithms is not trivial. Usually, the decision maker would first look for the feasibility of the solutions (i.e. tardiness of each job lower than the maximum tardiness) and, once it is achieved, he/she would look for a low value in the makespan. Finally, the quality of the sequences obtained by each algorithm has to be balanced against the time interval required to obtain the sequences, as the high CPU time requirements posed by some of the algorithms may not be acceptable for some scenarios. Therefore, there is a trade-off among these goals that increases the difficulty of a direct comparison of the algorithms. To make an exhaustive analysis of all these aspects, three indicators have been chosen to determine the quality of the solutions obtained by the algorithms, as well as the CPU time required to obtain the solutions. The indicators are:

- Number of feasible solutions obtained by each procedure.

- Makespan value of the solution (in terms of Average Relative Percentage Deviation) obtained by each procedure.
- Number of instances with the best solution obtained by each procedure.

These indicators are discussed in the next subsections.

5.1 Number of Feasible Solutions

The average number of feasible solutions obtained by the algorithms are shown in Table 1. For 494 instances out of the 540 instances in the benchmark it was possible to find a feasible solution by one/several algorithms. Among them, 492 feasible solutions were found by $ANPA$ for the stopping criterion $t = 60$, being the best algorithm in terms of the number of feasible solutions obtained. Next is the $ANPA$ heuristic with 491 feasible for the stopping criteria $t = 2$, $t = 5$ and $t = 20$, 490 for the $BICH$ heuristic. 457 feasible solutions were found by GA with $t = 60$ followed by the FL heuristic with 448 feasible solutions. The worst results were obtained for A_NEH and A_IGA algorithms.

It is worth to note that both $BICH$ and $ANPA$ found more feasible solutions within lesser CPU time than the rest of the procedures, a remarkable result specially as $BICH$ had very small CPU requirements. As it can be seen in the Table 1 and in Figure 7, the efficient algorithms following this criterion would be: $BICH$, $ANPA(t = 2)$ and $ANPA(t = 60)$.

5.2 Average Relative Percentage Deviation

The makespan of the solutions obtained by each algorithm can be evaluated by means of the Relative Percentage Deviation (RPD), which can be defined as follows:

$$RPD_i = \frac{C_{max,i} - Best}{Best} \cdot 100$$

where $C_{max,i}$ is the makespan of algorithm i while $Best$ is the best known value of the makespan (the lowest among the implemented heuristics). It has to be noted that RPD is

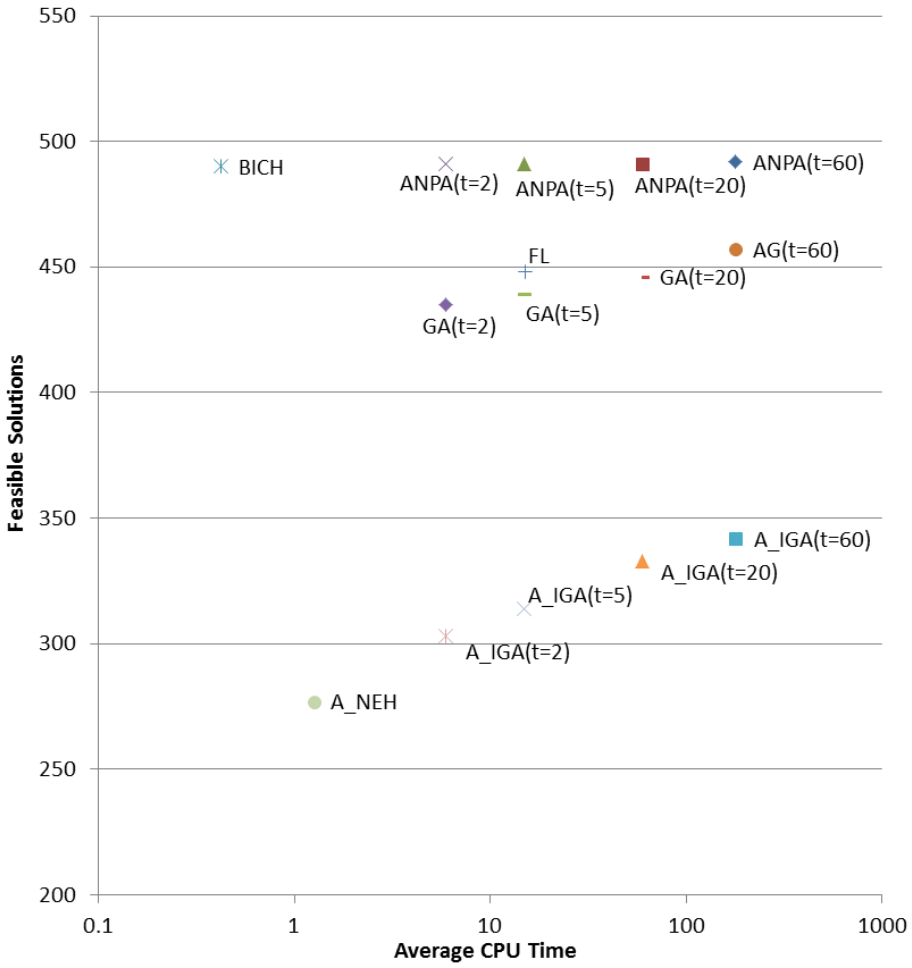

Figure 7: Number of feasible solutions vs Average CPU times for each algorithm. X-Axis is shown in logarithmic scale.

Table 1: Values of the three quality indicators and the average CPU time of each algorithm

Algorithm	#Feasible Solutions	# Best Instances	ARPD	Average CPU Time (s.)
A_NEH	277	9	2.83	1.27
$BICH$	490	14	3.24	0.43
FL	448	1	6.96	15.22
$ANPA(t=2)$	491	69	0.76	6
$A\_IGA(t=2)$	303	26	1.32	6
$GA(t=2)$	435	18	2.73	6
$ANPA(t=5)$	491	80	0.52	15
$A\_IGA(t=5)$	314	28	1.13	15
$GA(t=5)$	439	26	2.23	15
$ANPA(t=20)$	491	112	0.20	60
$A\_IGA(t=20)$	333	43	0.80	60
$GA(t=20)$	446	39	1.63	60
$ANPA(t=60)$	492	491	0.00	180
$A\_IGA(t=60)$	342	68	0.56	180
$GA(t=60)$	457	47	1.37	180

computed only if a feasible solution is found by the algorithm, otherwise the results would be greatly biased.

The average RPD (denoted as $ARPD$) is shown in Table 1. Since certain algorithms do not find feasible solutions for some instances for which others do, the $ARPD$ is calculated with different sample sizes depending on the algorithm, e.g. 333 instances for $A\_IGA(t=20)$ and instances 491 by $ANPA(t=20)$. This fact might cause that algorithms with lesser feasible solutions than other ones could have less $ARPD$, as it is the case with $A\_IGA(t=20)$ and $A\_IGA(t=60)$ as compared to $GA(t=60)$. Nevertheless, we also include it in the analysis since it is the usual way in which this analysis is carried out (see e.g. Framinan and Leisten, 2006; Ruiz and Allahverdi, 2009). As it can be seen in Figure 8, the efficient heuristics with $ARPD$ as indicator would be A_NEH, $BICH$, $ANPA(t=2)$, $ANPA(t=5)$, $ANPA(t=20)$ and $ANPA(t=60)$. In order to statistically justify this statement, we use Holm's procedure (Holm, 1979) with the following hypotheses:

- H_1: $ANPA(t=2) = GA(t=2)$
- H_2: $ANPA(t=2) = A\_IGA(t=2)$

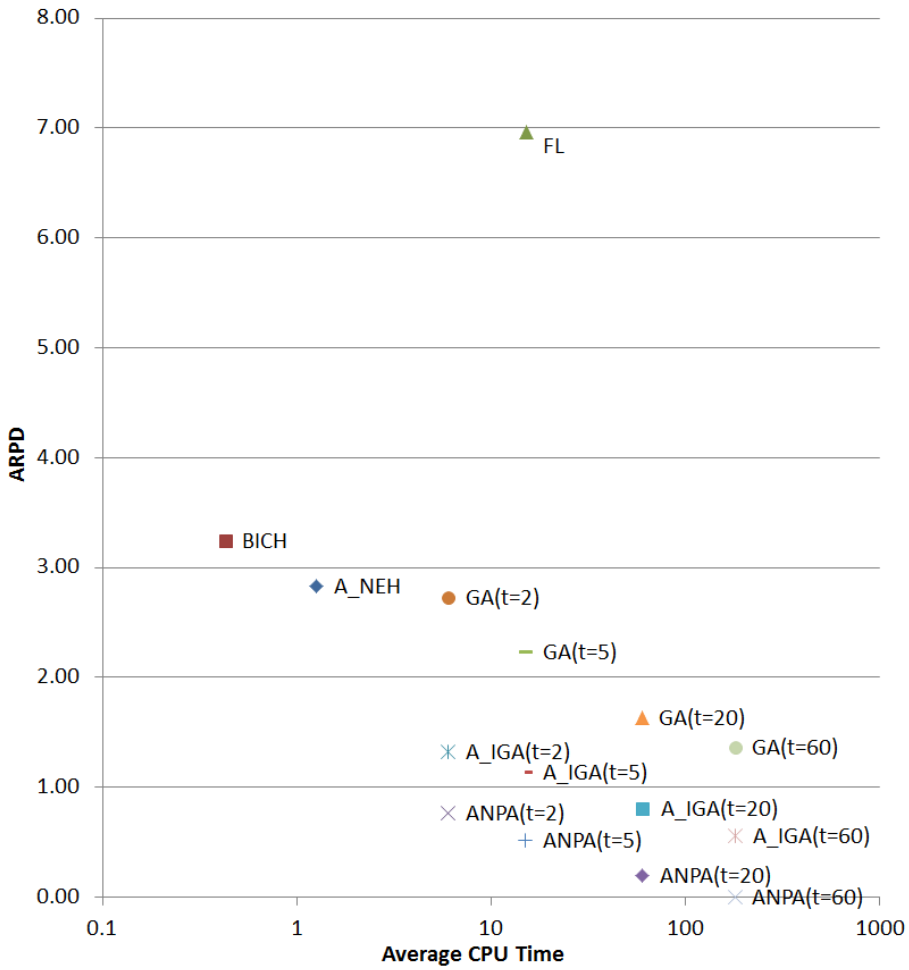

Figure 8: *ARPD* vs Average CPU times for each algorithm. X-Axis is shown in logarithmic scale.

Table 2: Holm's procedure for multiple hypotheses. R indicate that hypothesis is reject by Mann-Whitney and/or Holm's procedure

i	H_i	p-value	Mann-Whitney	$\alpha/(k-i+1)$	Holm's Procedure
1	$ANPA(t=2) = GA(t=2)$	0.000	R	0.0056	R
2	$ANPA(t=2) = A\_IGA(t=2)$	0.000	R	0.0063	R
3	$ANPA(t=5) = GA(t=5)$	0.000	R	0.0071	R
4	$ANPA(t=5) = A\_IGA(t=5)$	0.000	R	0.0083	R
5	$ANPA(t=5) = FL$	0.000	R	0.0100	R
6	$ANPA(t=20) = GA(t=20)$	0.000	R	0.0125	R
7	$ANPA(t=20) = A\_IGA(t=20)$	0.000	R	0.0167	R
8	$ANPA(t=60) = GA(t=60)$	0.000	R	0.0250	R
9	$ANPA(t=60) = A\_IGA(t=60)$	0.000	R	0.0500	R

- H_3: $ANPA(t=5) = GA(t=5)$
- H_4: $ANPA(t=5) = A\_IGA(t=5)$
- H_5: $ANPA(t=5) = FL$
- H_6: $ANPA(t=20) = GA(t=20)$
- H_7: $ANPA(t=20) = A\_IGA(t=20)$
- H_8: $ANPA(t=60) = GA(t=60)$
- H_9: $ANPA(t=60) = A\_IGA(t=60)$

The p-value of each hypothesis is calculated using a non-parametric Mann-Whitney test (see Pan et al., 2008). Then, Holm's procedure orders the hypotheses according to these p-values in non-decreasing order. The procedure rejects hypothesis i if its p-value is lower than $\alpha/(k-i+1)$ where k is the number of hypotheses. The results of this statistical analysis are shown in Table 2. As the p-values are always lower than $\alpha/(k-i+1)$, each hypothesis is rejected justifying the statement regarding the efficient algorithms in terms of their $ARPD$. In fact, each p-value of the non-parametric Mann-Whitney analysis is 0.000.

5.3 Number of instances with the best makespan

The third indicator used in this paper is the number of instances where each algorithm finds the best solution. This indicator is related to both the feasibility and the makespan in each instance of the algorithm. Results are shown in Table 1 for each algorithm. On the one hand, regarding the constructive heuristics, the $BICH$ algorithm finds the best solution in 14 instances as compared to the 9 and 1 of the A_NEH and FL heuristics respectively. On the other hand, the $ANPA$ algorithm is clearly the best with 69, 80, 112 and 491 instances for the stopping criteria $t = 2, 5, 20$ and 60 respectively. Regarding the other two iterative improvement algorithms (A_IGA and GA), A_IGA slightly improves GA for each stopping criterion. Taking into account this indicator, the efficient algorithms would be $BICH$, $ANPA(t = 2)$, $ANPA(t = 5)$, $ANPA(t = 20)$ and $ANPA(t = 60)$, as shown in Figure 9.

5.4 Summary

Although the determination of the best algorithms for the problem under study is not trivial due to the existence of infeasible solutions, the proposed algorithms $BICH$ and $ANPA$ have been found to be the most efficient algorithms for each one of the three indicators analysed. With respect to the rest of the algorithms, it is not clear whether A_IGA outperforms GA or vice versa, since the latter A_IGA is better for the last two indicators, but finds less feasible solutions. The same happens when comparing FL and A_NEH.

Regarding the computational time requirements, note that the average CPU time is both instance- and instance-size- dependent indicator (see Fernandez-Viagas and Framinan, 2015b). However, similar results are also found when using a dimensionless time indicator. Particularly, the average relative percentage computation time described in Fernandez-Viagas and Framinan (2015b) has been tested with similar results. In order not to excessively increase the extension of the paper, these findings are not detailed.

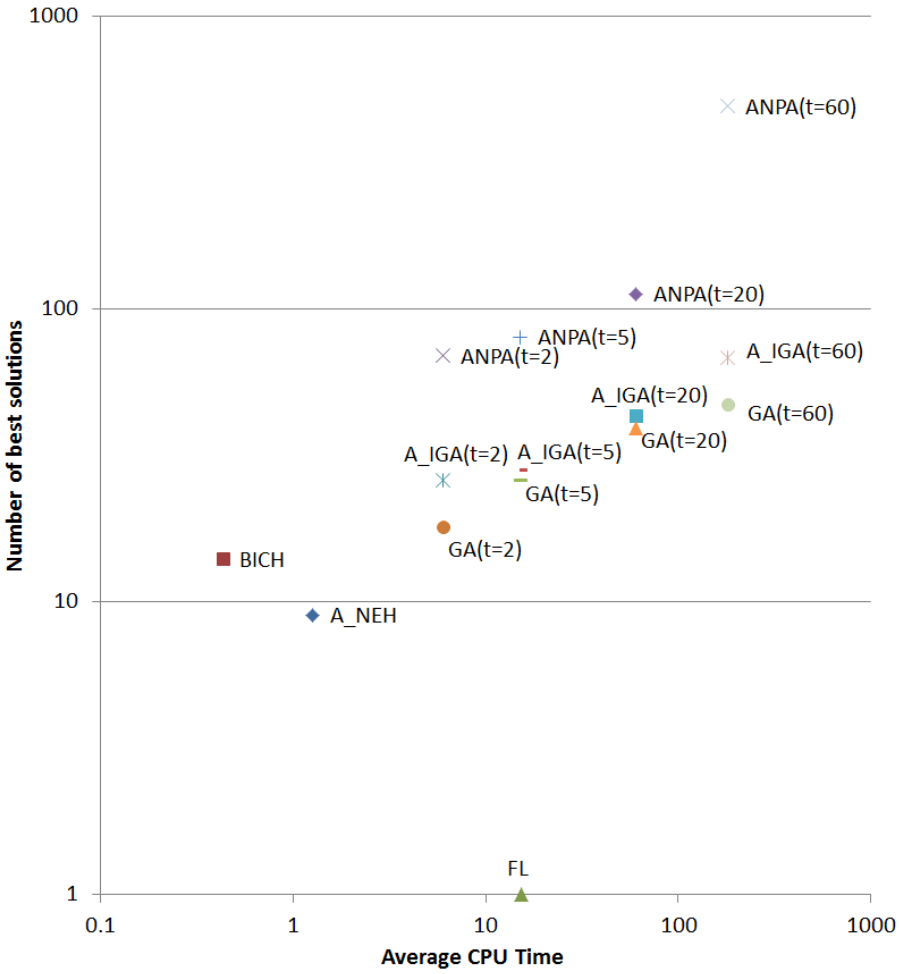

Figure 9: Amount of best solutions vs Average CPU times for each algorithm. X-Axis and Y-Axis are shown in logarithmic scale.

5.5 Different Distributions for the Processing Times

The above analyses have been performed with processing times following a uniform distribution, as it is usual in the literature for the PFSP (see e.g. the benchmarks of Taillard, 1993 and Demirkol et al., 1998). In this Section, three additional benchmarks have been generated using different distributions for the processing times in order to evaluate the robustness of the results. The procedure to generate the three benchmarks is the same as in Vallada et al. (2008), with the exception of the distribution of the processing times, which follow Exponential (positive and negative) and Normal distributions, respectively. Hence a total of 540 instances are generated per benchmark representing a total of 1620 instances. In order to have homogeneous results, the same mean (i.e. 50.5) is chosen for those distribution. In the case of the normal distribution, the standard deviation is chosen to achieve a moderate variation of the processing times which means, according to Hopp and Spearman (2000), a coefficient of variation between 0.75 and 1.33. Therefore, a value of 1 is used for the coefficient of variation. Additionally, the distributions are truncated and the lower bound and upper bounds are set to 1 and 100, the same as in the uniform distribution. A summary of the results is shown in Table 3 for the aforementioned three indicators. The results are very similar to that found using the uniform distribution (see Table 1). Additionally, the excellent behaviour and the efficiency of the two proposed algorithms (for the three indicators) are also confirmed in these benchmarks being e.g. the ARPD of the $ANPA(t = 60)$ algorithms less than 0.01.

6 Conclusions

This paper addresses the permutation flow-shop scheduling problem to minimise the makespan subject to that the tardiness of jobs does not exceed a given maximum tardiness. After analysing the problem and deriving some properties, a constructive heuristic $BICH$ and a non-population based algorithm $ANPA$ are proposed. The performance of both algorithms has been evaluated against the FL and GA algorithms which are the (up to now) state-of-the-art algorithms for the problem under study on an extensive benchmark of 540 instances. Additionally, two of the most efficient algorithms for the $Fm|prmu|C_{max}$ problem are also included in the comparison. The

Table 3: Values of the three quality indicators and the average CPU time of each algorithm considering different distributions of the processing times. The exponential distribution (positive and negative) are denoted by EP and EN respectively, as well as the normal distribution is denoted by N.

Algorithm	#Feasible Solutions			# Best Instances			ARPD			Average CPU Time (s.)		
	EP	EN	N	EP	EN	N	EP	EN	N	EP	EN	N
A_NEH	354	253	316	3	17	8	2.63	2.90	2.82	1.34	1.34	1.23
$BICH$	537	458	495	3	15	9	2.73	3.57	3.10	0.58	0.75	0.36
FL	490	420	443	0	3	2	6.19	9.47	6.96	14.18	17.53	13.90
$ANPA(t=2)$	537	458	495	34	84	55	0.63	0.88	0.73	6	6	6
$A\_IGA(t=2)$	392	270	343	8	27	25	1.24	1.48	1.38	6	6	6
$GA(t=2)$	511	416	433	5	19	18	2.10	3.54	2.57	6	6	6
$ANPA(t=5)$	538	459	495	47	96	69	0.43	0.59	0.49	15	15	15
$A\_IGA(t=5)$	403	281	352	9	38	28	1.07	1.26	1.17	15	15	15
$GA(t=5)$	515	426	445	8	36	21	1.74	2.88	2.13	15	15	15
$ANPA(t=20)$	538	459	495	77	122	98	0.17	0.23	0.19	60	60	60
$A\_IGA(t=20)$	421	292	364	19	55	40	0.77	0.90	0.84	60	60	60
$GA(t=20)$	522	442	454	20	51	34	1.26	2.08	1.53	60	60	60
$ANPA(t=60)$	538	459	495	535	455	490	0.00	0.01	0.00	180	180	180
$A\_IGA(t=60)$	432	304	370	35	84	57	0.55	0.63	0.59	180	180	180
$GA(t=60)$	526	444	457	29	65	43	1.04	1.69	1.26	180	180	180

efficiency of the two algorithms proposed has been shown according to three different measures of the quality of the solutions: number of feasible solutions, average relative percentage deviation, and number of instances with the best solution. Among the 494 feasible instances found in the benchmark, $ANPA(t = 60)$ finds the best solution for 491 instances with an $ARPD$ equal to 0.00. The performance of $BICH$ is also noteworthy, as it improves several iterative improvement algorithms using much lesser CPU time. These results are also confirmed in other three different benchmarks (of 540 instances each one) generated using three different distributions for the processing times of the jobs.

Acknowledgements

The authors are sincerely grateful to the anonymous referees, who provide very valuable comments on the earlier version of the paper. This research has been funded by the Spanish Ministry of Science and Innovation, under project "ADDRESS" with reference DPI2013-44461-P/DPI.

References

Chakravarthy, K. and Rajendran, C. (1999). Heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization. *Production Planning and Control*, 10(7):707–714.

Daniels, R. and Chambers, R. (1990). Multiobjective flow-shop scheduling. *Naval Research Logistics*, 37(6):981–995.

Demirkol, E., Mehta, S., and Uzsoy, R. (1998). Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141.

Fernandez-Viagas, V. and Framinan, J. (2014a). A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*.

Fernandez-Viagas, V. and Framinan, J. (2015a). Neh-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness. *Computers and Operations Research*, 60:27–36.

Fernandez-Viagas, V. and Framinan, J. (2015b). A new set of high-performing heuristics to minimise flowtime in permutation flowshops. *Computers and Operations Research*, 53:68–80.

Fernandez-Viagas, V. and Framinan, J. M. (2014b). On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers and Operations Research*, 45(0):60 – 67.

Framinan, J. and Leisten, R. (2006). A heuristic for scheduling a permutation flowshop with makespan objective subject to maximum tardiness. *International Journal of Production Economics*, 99(1-2):28–40.

Framinan, J. and Leisten, R. (2008). Total tardiness minimization in permutation flow shops: A simple approach based on a variable greedy algorithm. *International Journal of Production Research*, 46(22):6479–6498.

Framinan, J., Leisten, R., and Ruiz, R. (2014). *Manufacturing Scheduling Systems: An Integrated View on Models, Methods, and Tools*. Springer.

Gelders, L. F. and Sambandam, N. (1978). Four simple heuristics for scheduling a flow-shop. *International Journal of Production Research*, 16(3):221–231.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70.

Hopp, W. and Spearman, M. (2000). *Factory Physics*. McGraw-Hill/Irwin, Boston, Massachusetts.

Nawaz, M., Enscore Jr., E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, The International Journal of Management Science*, 11(1):91–95.

Osman, I. and Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. *Omega*, 17(6):551–557.

Pan, Q.-K., Tasgetiren, M., and Liang, Y.-C. (2008). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers and Industrial Engineering*, 55(4):795–816.

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. and Allahverdi, A. (2009). Minimizing the bicriteria of makespan and maximum tardiness with an upper bound on maximum tardiness. *Computers and Operations Research*, 36(4):1268–1283.

Ruiz, R. and Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

Sun, Y., Zhang, C., Gao, L., and Wang, X. (2011). Multi-objective optimization algorithms for flow shop scheduling problem: A review and prospects. *International Journal of Advanced Manufacturing Technology*, 55(5-8):723–739.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285.

T'Kindt, V. and Billaut, J.-C. (2006). *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer, New York, second edition.

Vallada, E. and Ruiz, R. (2010). Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *Omega*, 38(1-2):57–67.

Vallada, E., Ruiz, R., and Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers and Operations Research*, 35(4):1350–1373.