Elsevier Editorial System(tm) for Computers & Industrial Engineering
Manuscript Draft

Manuscript Number: CAIE-D-14-00168R1

Title: Efficient Heuristic Algorithms for the Blocking Flow Shop Scheduling Problem with Total Flow Time Minimization

Article Type: Research Paper

Keywords: Scheduling; flow shop; blocking; total flow time; heuristics

Corresponding Author: Prof. Imma Ribas, Ph.D

Corresponding Author's Institution: Universitat Politecnica de Catalunya

First Author: Imma Ribas, Ph.D

Order of Authors: Imma Ribas, Ph.D; Ramon Companys, professor

Abstract: This paper proposes two constructive heuristics, i.e. HPF1 and HPF2, for the blocking flow shop problem in order to minimize the total flow time. They differ mainly in the criterion used to select the first job in the sequence since, as it is shown, its contribution to the total flow time is not negligible. Both procedures were combined with the insertion phase of NEH to improve the sequence. However, as the insertion procedure does not always improve the solution, in the resulting heuristics, named NHPF1 and NHPF2, the sequence was evaluated before and after the insertion to keep the best of both solutions. The structure of these heuristics was used in Greedy Randomized Adaptive Search Procedures (GRASP) with variable neighborhood search in the improvement phase to generate greedy randomized solutions. The performance of the constructive heuristics and of the proposed GRASPs was evaluated against other heuristics from the literature. Our computational analysis showed that the presented heuristics are very competitive and able to improve 68 out of 120 best known solutions of Taillard's instances for the blocking flow shop scheduling problem with the total flow time criterion.

Efficient Heuristic Algorithms for the Blocking Flow Shop Scheduling Problem with Total Flow Time Minimization

Abstract

This paper proposes two constructive heuristics, i.e. HPF1 and HPF2, for the blocking flow shop problem in order to minimize the total flow time. They differ mainly in the criterion used to select the first job in the sequence since, as it is shown, its contribution to the total flow time is not negligible. Both procedures were combined with the insertion phase of NEH to improve the sequence. However, as the insertion procedure does not always improve the solution, in the resulting heuristics, named NHPF1 and NHPF2, the sequence was evaluated before and after the insertion to keep the best of both solutions. The structure of these heuristics was used in Greedy Randomized Adaptive Search Procedures (GRASP) with variable neighborhood search in the improvement phase to generate greedy randomized solutions. The performance of the constructive heuristics and of the proposed GRASPs was evaluated against other heuristics from the literature. Our computational analysis showed that the presented heuristics are very competitive and able to improve 68 out of 120 best known solutions of Taillard's instances for the blocking flow shop scheduling problem with the total flow time criterion.

***Title Page including Author Details**

Efficient Heuristic Algorithms for the Blocking Flow Shop Scheduling Problem with Total Flow Time Minimization

Imma Ribas[1,a], Ramon Companys[b]

a Departament d'Organització d'Empreses, DOE – ETSEIB - Universitat Politècnica de Catalunya. BarcelonaTech, Avda. Diagonal, 647, 7th Floor, 08028 Barcelona, Spain

b EPSEB - Universitat Politècnica de Catalunya. BarcelonaTech, Av. Doctor Marañón, 44-50, 3rd floor 08028 Barcelona, Spain

[1] Corresponding author.
E-mail address: imma.ribas@upc.edu
Fax: +34 93 401 60 54

Highlights

- Two efficient constructive heuristics for the Fm | block | ΣC_i are proposed.
- We show that the insertion phase of heuristic NEH can worsen the solution.
- The structure of each constructive method is used in a GRASP combined with VNS.
- The computational evaluation shows the good performance of these algorithms.

***Detailed Response to Reviewers**

Reviewer #1:

Thank you for your comments and suggestions. According to them, we have prepared a revised version of the paper, which we hope improves on the previous one. The changes (including four new references) have been coloured in yellow. Figure 9 and 10 has also been reworked to facilitate its understanding.

The rational for using the flowtime criterion is not made in this paper. From my understanding, makespan is a better representation for this type of problem. The authors have to reinforce this point in Introduction with a decent literature search, showing its relevance especially in industry.

According to your comment we have extended this part, in page 2, to show the relevance of using the flow time minimization in industry. We have included four references to reinforce this point.

For statistical analysis, was paired or unpaired test used? Also, it would be complete to give the t-values alongside p-values.

We understand that, since we are comparing 7 algorithms, your question of whether we used a paired or unpaired test, really means if we were using a blocked design or not. The question is pertinent because we want the comparisons to be "fair". That is the reason to include the number of jobs (*n*) and the number of machines (*m*) in the design. Further this allows analyzing the interactions, and therefore to see if all algorithms behave equally independently of the difficulty; in other words, to see if some of them are better or worse depending on *n* and *m*.
Since we are comparing 7 algorithms the appropriate technique is to conduct an ANOVA, in our case the *F* and *p* values are provided in Table 10, and it can be seen that all main effects and interactions are highly significant $(p \sim 0)$. One possible way to identify the reasons for rejection would be to conduct multiple comparisons and, as you say, provide the *p*-values. A simpler and in this case, we believe, that better alternative is to provide 95% confidence intervals for the average ARPD of each algorithm. Both for simplicity and clarity we have used this alternative (Figure 8).

Tables are poorly formatted. Caption should be above the table. Column data should be properly aligned (eg. Table 10).

Tables have been revised and Table 3 (see page 9) and Table 10 (see page 15) have been reworked according to your suggestion. Note that in the revised version the captions are above the table.

Efficient Heuristic Algorithms for the Blocking Flow Shop Scheduling Problem with Total Flow Time Minimization

Abstract

This paper proposes two constructive heuristics, i.e. HPF1 and HPF2, for the blocking flow shop problem in order to minimize the total flow time. They differ mainly in the criterion used to select the first job in the sequence since, as it is shown, its contribution to the total flow time is not negligible. Both procedures were combined with the insertion phase of NEH to improve the sequence. However, as the insertion procedure does not always improve the solution, in the resulting heuristics, named NHPF1 and NHPF2, the sequence was evaluated before and after the insertion to keep the best of both solutions. The structure of these heuristics was used in Greedy Randomized Adaptive Search Procedures (GRASP) with variable neighborhood search in the improvement phase to generate greedy randomized solutions. The performance of the constructive heuristics and of the proposed GRASPs was evaluated against other heuristics from the literature. Our computational analysis showed that the presented heuristics are very competitive and able to improve 68 out of 120 best known solutions of Taillard's instances for the blocking flow shop scheduling problem with the total flow time criterion.

Keywords: Scheduling; flow shop; blocking; total flow time; heuristics

1 Introduction

Many industrial systems can be modeled as a flow shop with zero capacity buffers between consecutive machines. In this type of production configuration, a machine can be blocked by the job it has processed if the next machine is not available. To avoid or minimize machine blocking and idle time, accurate scheduling is necessary. Examples of blocking flow shop scheduling can be found in the production of concrete blocks, where storage is not allowed in some stages of the manufacturing process (Grabowski & Pempera, 2000); in the iron and steel industry (Gong et al., 2010); in the treatment of industrial waste and manufacture of metallic parts (Martínez et al., 2006); or in a robotic cell, where a job may block a machine while waiting for the robot to pick it up and move it to the next stage (Sethi et al., 1992) .

This paper deals with the blocking flow shop scheduling problem to minimize the total flow time of jobs, denoted as $Fm|block|\sum C_i$ according to the notation proposed by (Graham et al., 1979), if jobs and machines are available at instant zero, a hypothesis considered here. If jobs' release time are zero, this objective is equivalent to (total or average) flow time minimization, which, according to Rajendran (1993) and (Framinan et al. 2005), has been found to reduce the scheduling costs. Additionally, it has been found to be an important real-life objective in industries since it results in the even utilization of resources, even turn-over of finished jobs and reduced in-process inventory. Therefore, it is considered to be more relevant and meaningful for today's dynamic production environment (Liu & Reeves, 2001). Pan and Ruiz (2013) remark that the need to reduce Work In Process (WIP) or in-process inventory has fostered the study of the total flow time.

In the $Fm|block|\sum C_i$ problem, n jobs have to be processed by m machines. All jobs follow the same route, implying that a job sequence determined for machine 1 is kept throughout the system. The processing time of job $i \in \{1,2,..., n\}$ on machine j, $j \in \{1,2,..., m\}$ is $p_{j,i} > 0$.

Although the blocking flow shop scheduling problem has not been as extensively studied as the permutation flow shop problem, the number of published papers concerning the former in order to minimize makespan has increased in recent years (Grabowski & Pempera, 2007; Wang et al., 2006; Liu et al., 2008; Qian et al., 2009; Wang et al., 2010; Ribas et al., 2011; Davendra & Bialic-Davendra, 2013). However, little research has been done on total flow time criterion. From the best our knowledge, only Wang et al. (2010), who proposed a hybrid Harmony Search (HS) algorithm, Deng et al. (2012), who proposed a Discrete Artificial Bee Colony (DABC) algorithm and Moslehi and Khorasanian (2013), who presented a branch and bound algorithm that can be used in small instances, have addressed the $Fm|block|\sum C_i$ problem. Therefore, it is interesting to intensify research to develop efficient heuristics for this problem, especially simple algorithms which are easy to adapt and implement in practical applications.

In this paper, we present two constructive procedures and two versions of an efficient Greedy Randomized Adaptive Search Procedure (GRASP) combined with variable neighborhood search in the improvement phase, which use the structure of these constructive heuristics to generate greedy randomized solutions. Computational

evaluation against other algorithms from the literature has shown the effectiveness of the constructive heuristic and good performance of the proposed GRASP.

The rest of the paper is organized as follows. In section 2, blocking flow shop scheduling is presented. Sections 3 and 4 describe the constructive procedures and the GRASP, respectively. Section 5 shows the computational evaluation of the algorithms and section 6 concludes.

2 Problem definition

In the blocking flow shop problem, a set of n jobs must be processed by m machines in the same order, from the first machine to machine m. Each job i, $i\epsilon\{1, 2, ..., n\}$ requires a fixed positive processing time $p_{j,i}$ on every machine j, $j\epsilon\{1, 2, ..., m\}$. Jobs and machines are available from time zero onwards. Our objective is to find a job processing sequence that minimizes the total flow time. $Fm|block|\Sigma C_i$ can be modeled with the following equations, where $[k]$ is the index of the job in the k-th position in the permutation, $e_{j,k}$ denotes the time at which the job $[k]$ starts to be processed by machine j and $c_{j,k}$ is the departure time of job $[k]$ from machine j. Note that if job $[k]$ can leave machine j when it is completed, which depends on the availability of machine j+1, then $c_{j,k}$ is not only the departure time but also the completion time of job $[k]$ on machine j:

$$e_{j,k} + p_{j,[k]} \leq c_{j,k} \quad j=1, 2, ..., m \quad k=1, 2, ..., n \qquad (1)$$

$$e_{j,k} \geq c_{j,k-1} \quad j=1, 2, ..., m \quad k=1, 2 ,..., n \qquad (2)$$

$$e_{j,k} \geq c_{j-1,k} \quad j=1, 2, ..., m \quad k=1, 2, ..., n \qquad (3)$$

$$c_{j,k} \geq c_{j+1,k-1} \quad j=1, 2, ..., m \quad k=1, 2, ..., n \qquad (4)$$

$$TF = \sum_{k=1}^{n} c_{m,k} \qquad (5)$$

with $c_{j,0} = 0 \quad \forall j \quad , \; c_{0,k} = 0, \quad c_{m+1,k} = 0 \quad \forall k$ being the initial conditions.

If equations (2) and (3) are summarized as (6) and equation (1) and (4) as (7), the schedule obtained is semi-active, which is interesting because an optimal solution can be found in the subset of the semi-active set of solutions.

$$e_{j,k} = \max\{c_{j,k-1}; c_{j-1,k}\} \quad j=1, 2, ..., m \quad k=1, 2, ..., n \qquad (6)$$

$$c_{j,k} = \max\{e_{j,k} + p_{j,[k]}, c_{j+1,k-1}\} \quad j=1, 2, ..., m \quad k=1, 2, ..., n \qquad (7)$$

3 Constructive heuristics

The constructive heuristics proposed are based on the *profile fitting* (PF) procedure by McCormick et al. (1989) for solving sequencing problems in an assembly line with blocking to minimize the cycle time. The PF technique tries to sequence jobs in order to minimize machine timeout, which can be due to idle time, blocking time or the sum of both (see Figure 1). This is an adequate objective for the blocking flow shop problem. The PF method, combined with the insertion phase of the heuristic NEH (Nawaz et al., 1983), was successfully used by Ronconi (2004) for scheduling jobs in a blocking flow shop to minimize makespan. Thus, this procedure considers the blocking constraint; however, it does not address total flow time minimization. To include this objective in the construction of sequences, two variants are proposed.

(please insert Figure 1 near here)

First, we describe the PF heuristic to make clear the procedure and its variants.

Let σ the partial sequence and σ^*i the partial sequence with job i added at the end of σ.

- Step 1: Select the job with the minimum sum of all operations of a job (P_i) and put it in the first position in sequence σ. Set k=1. In case of ties, select the job with minimum $p_{1,i}$.
- Step 2: While $k<n$, calculate the machine timeout for each unscheduled job i when job i is added to the partial sequence σ, denoted as σ^*i. Select the job that leads to the minimum timeout and add it to σ; $k=k+1$. The timeout is calculated according to equation (8). In case of ties, select the job which leads to the partial sequence with minimum total flow time.

$$Ind0\ (i, k) = \sum_{j=1}^{m} (c_{j,k+1}(\sigma * i) - c_{j,k}(\sigma) - p_{j,i}) \qquad (8)$$

In order to consider the total flow time criterion during the scheduling of jobs, a new term was added to index (8) to measure the contribution of the evaluated job i to total flow time of the partial sequence. Therefore, the first heuristic proposed, named HPF1, can be defined as follows:

- Step 1: Select the job with minimum P_i and put it in the first position in sequence σ. Set k=1. In case of ties, select the job with minimum $p_{1,i}$.
- Step 2: While $k<n$, calculate index (*ind1*) as in equation (9) for each unscheduled job i, where C_i is the completion time of job i. Select the job with minimum *ind1*. In

case of ties, select the job which leads to the partial sequence with minimum total flow time.

$$ind1(i,k) = \mu \cdot (\sum_{j=1}^{m}(c_{j,k+1}(\sigma * i) - c_{j,k}(\sigma) - p_{j,i})) + (1-\mu) \cdot (C_i - C_{[k-1]}) \qquad (9)$$

Both procedures, PF and HPF1, schedule job i with the minimum sum of processing time on machines m (P_i) in the first position, but this rule may not always be effective in minimizing the total flow time. Consider the following 3-job, 3-machine problem where jobs J1, J2 and J3 have the following processing time on each machine: J1=(2,3,4), J2=(3,2,4) and J3=(4,3,2). The sum of processing times of each job is 9, but the total flow time of schedules {J1. J2, J3}, {J2, J1, J3} and {J3, J1, J2} is 38, 39 and 41, respectively. These differences are due to the front delay induced by the first job scheduled (grey parts in Figure 2).

(please insert Figure 2 near here)

Hence, it is interesting to schedule in the first position not only the job with the minimum completion time (minimum P_i) but also the one that generates the minimum front delay. Front delay can be measured in several ways, one of which is by estimating the slope of the line from the starting point of the first operation scheduled to the middle point of the starting points of all operations. In Figure 3, we can observe that the slope can be measured by calculating the tangent of angle α ($tan(\alpha)$) by dividing the x-axis value at the endpoint of this line:

$x_p = \sum_{j=1}^{m}(m-j) \cdot p_{j,i} \Big/ m$ by the y-axis value, which is $y_p = (m-1)/2$. Hence, $tan(\alpha) = 2 \cdot \sum_{j=1}^{m}(m-j) \cdot p_{j,i} \Big/ m \cdot (m-1)$.

A job which generates minor front delay has a low value of tan(α). Thus, scheduling the job with minimum $tan(\alpha)$ in the first position of the sequence can help to minimize the total flow time of the whole sequence.

(please insert Figure 3 near here)

However, as said before, the choice of the first job in a sequence depends on its contribution to the total flow time (P_i) and resulting front delay. Therefore, in the

second procedure, named HPF2, we created a bicriteria index (R(i)) to measure both terms. Note in equation (10) that the first term was corrected by multiplying it by *m* because it had a different magnitude from the sum of processing time of a job. Observe that, with the correction introduced in the first term, if the processing time in all stages is 1, both terms are equal to *m*, which demonstrates that both have the same magnitude.

$$R(i) = \lambda \cdot \left(2 \cdot \sum_{j=1}^{m} (m - j) \cdot p_{j,i} \Big/ (m-1) \right) + (1-\lambda) \cdot \sum_{j=1}^{m} p_{j,i} \qquad (10)$$

Therefore, HPF2 can be defined as follows:

- Step 1: Select the job with minimum R(*i*) and put it in the first position in sequence σ. Set k=1. In case of ties, select the job with minimum $p_{1,i}$.
- Step 2: While $k<n$, calculate index (*ind1*) as in equation (9) for each unscheduled job *i*. Select the job with minimum *ind1*. In case of ties, select the job which leads to the partial sequence with minimum total flow time.

The flow time calculation in an *n*-job, *m*-machine flow shop for a given sequence is of complexity O(*nm*). Hence, as *k* flow times in *k* jobs and *m* machines must be calculated in step 2 of PF, HPF1 and HPF2, we can conclude that the complexity of these procedures is $O(n^2m)$.

Finally, the insertion phase of NEH, adapted to the total flow time criterion, is applied to the sequence given by PF, HPF1 and HPF2 to try to improve them. We name these combinations NPF, NHPF1 and NHPF2, respectively. Therefore, the third step of these heuristics is defined as follows:

- Step 3: In accordance with the order established in step 2, take the first two jobs and schedule them in such a way that they minimize the total flow time of the partial sequence, considering an instance with only two jobs. Then, for k=3 up to *n*, insert the *k*-th job into one of the possible *k* positions of the partial sequence. Keep the partial sequence with minimum total flow time. In case of ties, select the partial sequence with minimum makespan.

Ribas et al. (2013) showed that, for the blocking flow shop problem with makespan minimization, the insertion phase of NEH can worsen the solution, i.e. C_{max} value obtained by the initial sequence. This can also be observed in the sequences given by PF, HPF1and HPF2 for the total flow time criterion, which is analyzed in Section 5. Thus, in NPF, NHPF1 and NHPF2 the obtained sequence is evaluated before and after the insertion phase to keep the best of both solutions. The complexity of NPF, NHPF1

and NHPF2 is O(n^3m), which corresponds to the original complexity of NEH, because now it is Step 3 which determines the complexity of these algorithms.

3.1 Experimental adjustment of heuristic parameters

Certain parameters of the NHPF1 and NHPF2 heuristics must be adjusted. In particular, parameter μ in NHPF1and parameters λ and μ in NHPF2 require proper calibration. Calibration was done on a test-bed created *ad hoc* to separate the calibration benchmark from the final testing benchmark. Each algorithm was tested with 140 randomly generated instances, grouped in 28 sets of size n x m, where n= {20, 50, 80, 110, 140, 170, 200} and m = {5, 10, 15, 20} on a 2 GHz Intel Core 2 Duo E8400 CPU with 2 GB of RAM. To compare the solution given by each parameter value, the relative percentage deviation (RPD) from a reference solution was calculated as in (11):

$$RPD = \frac{TF_{k,h} - TFref_k}{TFref_k} \cdot 100 \qquad (11)$$

where $TF_{k,h}$ is the total flow time given by heuristic h with a fixed value of parameters in instance k, and $TFref_k$ is the best value of the total flow time obtained in this instance.

The test results are summarized in Tables 1 and 2. The former shows the overall average RPD obtained by NHPF1 for several values of μ. It can be observed that the average RPDs for each μ are quite similar, but a slight improvement is obtained for μ=0.7 or μ=0.75. Hence, this parameter was set to 0.75, but it could also have been set to 0.7.

Table 1. Average RPD value of NHPF1 for different values of μ

μ	**0.65**	**0.70**	**0.75**	**0.80**	**0.85**
NHPF1	0.285	**0.284**	**0.284**	0.285	0.285

Table 2 shows the overall average RPD obtained by NHPF2 for all tested combinations of λ and μ values. The best values are obtained by setting μ to 0.75 and λ to 0.65. For this reason, we used these values in the computational evaluation.

Table 2. Average RPD value of NHPF2 for different values of λ and μ

λ \ μ	**0.65**	**0.70**	**0.75**	**0.80**	**0.85**
0.55	0.533	0.463	0.378	0.427	0.525
0.6	0.520	0.448	0.373	0.423	0.511
0.65	0.510	0.441	**0.362**	0.414	0.499
0.7	0.508	0.439	0.367	0.407	0.503
0.75	0.510	0.445	0.374	0.413	0.507

4 Proposed Greedy Randomized Adaptive Search Procedure

A Greedy Randomized Adaptive Search Procedure (GRASP), first presented by Feo and Resende (1989), is a metaheuristic algorithm commonly applied to combinatorial optimization problems. GRASP consists of iterations made up of successive constructions of a greedy randomized solution and a following local search which tries to improve the solution until a specified stopping criterion is reached.

Two variants of a GRASP, each of which uses one of the proposed constructive heuristics, i.e. NHPF1 and NHPF2, were implemented to construct greedy randomized solutions. We name them GRASP(NHPF1) and GRASP(NHPF2) to specify the initial constructive heuristic used. In both algorithms, the value of μ was randomly selected from a uniform distribution between μ_{min} and μ_{max} (U[μ_{min}, μ_{max}]) at each iteration. Moreover, in GRASP(NHPF2), the value of λ was set to 0.65, i.e. the value obtained in the calibration test of NHPF2, to select the first job because, as shown in section 5.1, this influences the quality of the generated sequence.

A general scheme of GRASP(NHPF1) and GRASP(NHPF2) is given in Figure 4.

(please insert Figure 4 near here)

Although the constructive randomized procedure used in each GRASP is slightly different, the remaining structure is the same. The greedy constructive solution goes to the local search, which consists of a variable neighborhood search made up of two neighborhood structures: swap and insertion. The procedures for exploring them are named LS1 and LS2, respectively.

In LS1, neighbors are generated for each job in the sequence by swapping one job with all jobs that follow it in the sequence. If the best neighbor (σ') is better than the current solution (σ), it becomes the new current solution σ and the process continues until all jobs have been considered. To avoid always exploring neighborhoods in the same order, jobs are selected randomly.

In LS2, neighbors are generated for each job in the sequence by removing the job from its position and inserting it into all other possible positions. If the best neighbor (σ') is better than the current solution (σ), it becomes the new current solution σ and the process continues until all jobs have been considered. As in LS1, jobs are selected randomly.

The implemented local search (Figure 5) uses both structures, one after the other, at each iteration. The first neighborhood to be explored is selected randomly with a probability of 50%. After exploring the neighboring solutions of the current solution σ, the local optimum σ' is compared with σ. If the solution has improved, σ' replaces σ and the search continues in the other neighborhood. This process goes on until the current solution is no longer improved. Next, the local optimum σ' is compared with the best solution σ* in terms of quality. If TF(σ') is less than TF(σ*), then σ' replaces σ*.

(Please insert figure 5 near here)

5 Computational evaluation

Prior to the evaluation of the proposed GRASP, a computational test was done to select the value of μ_{min} and μ_{max} used by the GRASP to construct greedy randomized solutions. This test was conducted on the same test-bed used before for the calibration of NHPF1 and NHPF2. Eight intervals of μ were tested and the results were analyzed by an ANOVA. The hypotheses were tested by a residual analysis, which showed small departures from normality mainly due to a low level of skewness and three borderline outliers. However, the ANOVA method is robust to violations of this assumption. This, together with the clarity of the results, validates the conclusions and makes a deeper analysis unnecessary.

Table 3. ANOVA: ARPD versus *interval*, *n* and *m*

Source	DF	SS	MS	F	P
Main Effects					
interval	7	1.022	0.146	3.63	0.001
n	6	28.873	4.812	119.62	0.000
m	3	0.224	0.074	1.86	0.135
Interactions					
*Interval*n*	42	3.847	0.091	2.28	0.000
*Interval*m*	21	0.252	0.012	0.30	0.999
*n*m*	18	3.919	0.217	5.41	0.000
Error	1022	41.113	0.040		
Total	1119	79.253			

The statistical analysis of results (see Table 3) indicates a significant difference between *intervals*, *n* and their interaction, as shown in the interval plot in Figure 6. The interval number corresponds to the following values of μ_{min} and μ_{max} according to this order: [0,1], [0.1,1], [0.2,1], [0.3,1], [0.4,1], [0.5,1], [0.6,0.9], [0.7,0.8]. As can be seen, the interval range depends on *n*; specifically, the range decreases when the number of jobs

increases. According to these results, for $n \leq 50$ the interval can be set to [0,1], [0.1,1], [0.2,1] or [0.3,1]. Nevertheless, when n increases up to 140 jobs, the best interval becomes narrower ([0.4,1], [0.5,1]) and for n bigger than 140, the required interval is further reduced [0.6,0.9]. This can be explained by the compromise between the diversity of the greedy randomized solutions and their quality. For small values of n, a narrow interval could lead to very similar solutions, causing the algorithm to be trapped in the same local minimum. On the other hand, a narrower interval is required for higher values of n because a huge interval could result in poor quality solutions, far from the optimum, that would not be able to improve enough in a limited CPU time. Therefore, we set the interval depending on n according to the values in Table 4.

Table 4. Values of μ_{min} and μ_{max} for each range of n

n	μ_{min}	μ_{max}
$0 < n \leq 50$	0	1
$75 < n \leq 140$	0.5	1
$140 > n$	0.6	0.9

(Please insert figure 6 near here)

In the following sections, the performance of the constructive heuristics and of the GRASPs proposed is evaluated. Two tests, one for the former and one for the latter, were done using Taillard's (1993) benchmark for the blocking flow shop scheduling problem with the total flow time criterion as in Wang et al. (2010) and Deng et al. (2012), although in the latter the authors use only the first 90 instances. Taillard's test-bed is composed of 120 instances, 12 sets of 10 instances each, from 20 jobs and 5 machines to 500 jobs and 20 machines where $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$, but not all combinations of n and m are available. In particular, sets 200x5, 500x5 and 500x10 are missing but they were added, as in (Pan & Ruiz, 2012), to maintain the orthogonality of the experiment.

5.1 Evaluation of the constructive heuristics

In the first test we compared the performance of NPF, NHPF1 and NHPF2 against NSPT, which, according to our nomenclature, indicates that the SPT rule is used to sequence the jobs in the first step.

First, we analyzed the performance of the proposed constructive procedures with and without the insertion phase to show that, in some instances, the insertion phase worsens

the solution. The results are summarized in the graphs of Figure 7, which show the behavior of the constructive procedures before and after the insertion phase. Note that we add "_i" to the procedure's name to show the results after the insertion phase. As can be seen in the first graph, the insertion phase always improves the SPT sequence. On the other hand, in the PF_i, HPF1_i and HPF2_i the solution is sometimes worse than in PF, HPF1 and HPF2, respectively. This is so because these three procedures use good constructive methods to sequence jobs which are able to find good solutions for the problem at hand. Therefore, the solution before the insertion phase is quite good, whereas the solution obtained by the SPT rule is poor.

(please, insert figure 7 near here)

It is also interesting that the more effective the constructive heuristic, the less advisable the insertion phase. In general, the solutions obtained by PF are better than those obtained by PF_i from 200 jobs onwards, whereas those obtained by HPF1 and HPF2 are better than by HPF1_i and HPF2_i, respectively, from 100 jobs. But this behavior is not easy to predict because it can also be observed in instances with fewer jobs. Therefore, when using the insertion procedure it is recommended to evaluate the sequence before and after the insertion phase in order to retain the best of both sequences. We implemented NPF, NHPF1 and NHPF2 according to this suggestion.

Second, these procedures were compared with the RPD index calculated as in (11), considering ($TFref_k$) the best known solution so far as the reference value. These reference values are summarized at the end of section 5.2 because some were improved during the research.

The algorithms were coded in the same language (QuickBASIC) and tested on the same computer, a 3 GHz Intel Core 2 Duo E8400 CPU with 2 GB of RAM.

The comparison of all procedures with the overall ARPD value (Table 6) indicates that NPF is better than NSPT but worse than NHPF1 and NHPF2, which exhibit very good performance. In particular, NHPF2 performs slightly better than NHPF1, suggesting that the effect of selecting the first job in the sequence is not negligible. We can therefore conclude that, in addition to the total processing time, the minimization of front delay is a good criterion to select the first job in the sequence.

Table 6. Average RPD of constructive heuristics

n	*m*	NSPT	NPF	NHPF1	NHPF2
20	5	3.264	2.948	2.928	3.059
20	10	3.071	2.625	2.719	2.340
20	20	3.581	2.840	2.857	2.766
50	5	6.329	5.195	3.903	3.528
50	10	5.525	4.105	4.191	3.665
50	20	4.748	4.450	4.398	4.237
100	5	7.504	7.364	3.757	3.668
100	10	6.134	5.197	4.450	3.964
100	20	5.138	4.175	4.428	4.539
200	10	6.275	3.279	2.505	1.915
200	20	4.454	2.392	2.676	2.478
500	20	5.220	1.178	1.464	1.533
200	5	8.353	6.761	2.260	1.936
500	5	10.039	7.645	1.330	1.027
500	10	7.950	2.650	1.399	1.307
Overall		5.839	4.187	3.018	**2.797**

5.2 Evaluation of the GRASP heuristics

In the second test, we compared the proposed NHPF2, GRASP(HPF1), GRASP(NHPF1), GRASP(HPF2) and GRASP(NHPF2) procedures against two benchmark algorithms from the literature, the Harmony Search (HS) algorithm by Wang et al. (2010) and the Discrete Artificial Bee Colony (DABC) algorithm by Deng et al. (2012). As we showed that the insertion phase can worsen the obtained solution, we included GRASP(HPF1) and GRASP(HPF2), which use HPF1 and HPF2 to generate greedy randomized solutions, to see whether it is better to use the available CPU time to add the insertion phase or to do a few iterations more.

All algorithms were coded in the same language (QuickBASIC) and tested on the same computer, a 3 GHz Intel Core 2 Duo E8400 CPU with 2 GB of RAM. To make a fair comparison, all algorithms adopted the CPU time limit as a stopping criterion, which was fixed to $k \cdot n^2 \cdot m \; 10^{-5}$ seconds, with k set to 10 and 30 to analyze the performance of these algorithms for two levels of CPU time. Five runs were carried out for each algorithm for all 150 instances.

As in the other tests, the relative percentage deviation (RDP) was calculated from the best known solution as in (11), taking as $TF_{h.k}$ the average total flow time obtained in the 5 runs for heuristic h in instance k, and as $TFref_k$, the best known solution reported in Table 11, at the end of this section.

Tables 7 and 8 show the average relative percentage deviation (ARPD) for each set of *n*x*m* instances for *k*=10 and *k*=30, respectively. The comparison of both tables indicates that the ranking between these algorithms is the same for the two levels of CPU time and that they converge quite fast because the results for *k*=10 are not very different from those obtained with an increased CPU time (*k*=30). From these results, we can say that the proposed GRASPs outperform the other algorithms for those instances with more than 20 jobs. For instances with 20 jobs, the best performing algorithm is DABC, but its efficiency decreases for larger instance sizes, probably because this algorithm requires much more time than the others to reach good solutions.

Table 7. Average ARPD value of heuristics for *k*=10

*n*x*m*	NHPF2	HS	DABC	GRASP (HPF1)	GRASP (NHPF1)	GRASP (HPF2)	GRASP (NHPF2)
20x5	3.059	0.515	0.090	0.151	0.138	0.145	0.172
20x10	2.340	0.202	0.067	0.248	0.157	0.198	0.310
20x20	2.766	0.112	0.039	0.183	0.205	0.161	0.138
50x5	3.528	5.576	2.358	1.866	1.755	1.695	1.673
50x10	3.665	4.777	2.320	1.760	1.790	1.748	1.743
50x20	4.237	3.360	1.577	1.340	1.416	1.385	1.328
100x5	3.668	7.622	3.253	1.911	1.985	1.928	1.925
100x10	3.964	6.394	3.804	2.404	2.387	2.046	2.085
100x20	4.539	5.093	2.871	2.306	2.159	2.142	2.183
200x10	1.915	6.109	2.948	1.040	1.061	0.999	0.994
200x20	2.478	4.408	2.491	1.083	0.942	0.909	0.940
500x20	1.533	5.235	3.043	0.546	0.465	0.438	0.469
200x5	1.936	8.430	2.913	0.809	0.882	0.756	0.750
500x5	1.027	10.217	2.861	0.613	0.610	0.481	0.588
500x10	1.307	8.018	3.730	0.733	0.690	0.626	0.670
Overall	2.797	5.071	2.291	1.133	1.110	**1.044**	1.065

The performance of NHPF2 is worth noting. For instances with more than 20 jobs, this algorithm is better than HS and performs similarly to DABC, and for more than 100 jobs it is even more efficient because it obtains better solutions in a considerably shorter CPU time. This means that, despite its simplicity, it is a good heuristic for the blocking flow shop scheduling problem with flow time minimization.

GRASP(HPF2) and GRASP(NHPF2) perform very similarly although the latter is slightly more advantageous. The same is true of GRASP(HPF1) and GRASP(NHPF1). Therefore, for simplicity, it is better to use HPF1 and HPF2 to construct greedy randomized solutions. GRASP(HPF2) performs slightly better than GRASP(HPF1) for

instances with more than 20 jobs, a difference that grows with the number of jobs. The same behavior is observed for NHPF1 and NHPF2**.**

Table 8. Average ARPD value of heuristics for *k*=30

nxm	NHPF2	HS	DABC	GRASP (HPF1)	GRASP (NHPF1)	GRASP (HPF2)	GRASP (NHPF2)
20x5	3.059	0.228	0.027	0.105	0.115	0.095	0.135
20x10	2.340	0.148	0.021	0.034	0.135	0.087	0.186
20x20	2.766	0.058	0.003	0.160	0.137	0.109	0.107
50x5	3.528	5.021	1.703	1.631	1.643	1.459	1.514
50x10	3.665	4.311	1.510	1.545	1.561	1.555	1.568
50x20	4.237	2.999	0.984	1.223	1.196	1.240	1.226
100x5	3.668	7.594	2.920	1.757	1.741	1.700	1.762
100x10	3.964	6.437	3.398	2.151	2.227	1.939	1.928
100x20	4.539	4.930	2.548	1.909	2.052	1.964	2.019
200x10	1.915	6.059	2.926	0.864	0.827	0.827	0.838
200x20	2.478	4.366	2.486	0.835	0.878	0.786	0.766
500x20	1.533	5.248	3.043	0.433	0.409	0.382	0.373
200x5	1.936	8.298	2.856	0.655	0.641	0.561	0.569
500x5	1.027	10.230	2.861	0.439	0.453	0.351	0.460
500x10	1.307	7.997	3.730	0.568	0.585	0.507	0.518
Overall	2.797	4.928	2.068	0.954	0.973	**0.904**	0.931

To validate the above observation, the results were analyzed by three-way completely randomized ANOVA with interactions (Wu & Hamada, 2000) where the factors were algorithm, number of jobs (*n*) and number of machines (*m*). The interactions allow us to analyze the behavior of algorithms depending on the instance size. As the ranking and the gap between algorithms are the same for both levels of CPU time, a statistical analysis was conducted for the results obtained with *k*=30.

The statistical analysis confirmed a significant difference between algorithms, *n*, *m* and their interaction. These results were analyzed using three graphs. Figure 8 shows the interval plot of average RPD (ARPD) per algorithm, which makes clear that GRASP(HF1), GRASP(HPF2), GRASP(NHPF1) and GRASP(NHPF2) are better than the others. However, there is no statistical significant evidence that one is better but since GRASP(HPF2) has an ARPD slightly lower than the others, we recommend this algorithm for the problem at hand.

Table 10. Three-way ANOVA: ARPD versus *Algorithm, n and m*

Source	DF	SS	MS	F	P
Main Effects					
n	4	688.267	172.067	515.70	0.000
m	2	25.169	12.585	37.72	0.000
Algoritm	6	2038.140	339.690	1.18.08	0.000
Interactions					
*n*m*	8	11.857	1.482	4.44	0.000
*n*Algorithm*	24	1021.643	42.568	127.58	0.000
*m*Algorithm*	12	179.737	14.978	44.89	0.000
Error	993	331.323	0.334		
Total	1049	4296.136			

(please insert Figure 8 near here)

Figure 9 shows the interaction between the algorithms and *n*. It can be observed that for *n*=20 NHPF2 performs worse than the others. For *n*=50 the algorithms are separated in two groups: NHPF2 and HS, with lower performance, and the rest of algorithms. From *n*=100 onwards, the four GRASPs are more efficient than the other algorithms and NHPF2 performs better than DABC when *n* increases. Notice that the interaction is due to the behavior of HS which, contrary to the others algorithms, performs worse when *n* increases.

(please insert Figure 9 near here)

Figure 10 shows the interaction between the algorithms and *m*. The algorithms have similar performance in all cases, irrespective of *m*. However, the interaction is due to the performance of HS that decreases when *m* increases.

(please insert figure 10 near here)

Finally, the new best solutions found during this research are summarized in Table 11, which can be used as a basis of comparison for future research. The number in columns "Source" indicates the paper that reported the values: "1" for Wang et al. (2010), "2" for Moslehi and Khorasanian (2013), "3" for Deng et al. (2012) and "4" for this research. For simplicity, the numbering of Taillard's instances is kept. Therefore, instances from 1 to 120 belong to Taillard's test-bed and those from 121-150 are the 30 instances added.

Table 11. Best solutions for the blocking flow shop with flow time criterion

Set	Best	Source	Set	Best	Source	Set	Best	Source
20×5			20×10			20×20		
1	14953	1,2,3,4	11	22358	1,2,3,4	21	34683	1,2,3,4
2	16343	1,2,3,4	12	23881	1,2,3,4	22	32855	1,2,3,4
3	14297	1,2,3,4	13	20873	1,2,3,4	23	34825	1,2,3,4
4	16483	1,2,3,4	14	19916	1,2,3,4	24	33006	1,2,3,4
5	14212	1,2,3,4	15	20196	1,2,3,4	25	35328	1,2,3,4
6	14624	1,2,3,4	16	20126	1,2,3,4	26	33720	1,2,3,4
7	14936	1,2,3,4	17	19471	1,2,3,4	27	33992	1,2,3,4
8	15193	1,2,3,4	18	21330	1,2,3,4	28	33388	1,2,3,4
9	15544	1,2,3,4	19	21585	1,2,3,4	29	34798	1,2,3,4
10	14392	1,2,3,4	20	22582	1,2,3,4	30	33174	1,2,3,4
50×5			50×10			50×20		
31	72672	3,4	41	99674	3,4	51	136865	3,4
32	78140	3	42	95608	4	52	129958	4
33	72913	4	43	91791	3,4	53	127617	3,4
34	77399	4	44	98454	3,4	54	131889	3,4
35	78353	4	45	98164	3	55	130967	3
36	75402	3,4	46	97246	4	56	131760	4
37	73842	4	47	99953	3,4	57	134217	3,4
38	73442	4	48	98027	4	58	132990	4
39	70871	3,4	49	96708	3,4	59	132599	3,4
40	78729	4	50	98019	4	60	135710	4
100x5			100x10			100×20		
61	288627	4	71	354524	4	81	425304	4
62	280491	4	72	335609	4	82	436360	3
63	276576	4	73	344090	4	83	430634	3
64	261278	4	74	359491	4	84	432344	4
65	274638	4	75	338537	3	85	427150	4
66	267554	4	76	327254	4	86	430532	3
67	275823	4	77	336360	4	87	437739	4
68	269872	4	78	343368	4	88	441173	3
69	285428	4	79	344563	4	89	432876	3
70	282828	4	80	347845	4	90	437785	4
200×10			200×20			500x20		
91	1282396	4	101	1502049	4	111	8733885	4
92	1284743	4	102	1542868	4	112	8854894	4
93	1283521	4	103	1556987	4	113	8793747	4
94	1283126	4	104	1549491	4	114	8839615	4
95	1283888	4	105	1517943	4	115	8797812	4
96	1252880	4	106	1530159	4	116	8849661	4
97	1304158	4	107	1532090	4	117	8786821	4
98	1304187	4	108	1547372	4	118	8808920	4
99	1279766	4	109	1527564	4	119	8792132	4
100	1278516	4	110	1545061	4	120	8862934	4
200×5			500x5			500x10		
121	1077132	4	131	6390125	4	141	7556997	4
122	1026709	4	132	6418782	4	142	7673115	4
123	1066136	4	133	6467532	4	143	7630163	4
124	1051122	4	134	6336628	4	144	7627243	4
125	1065882	4	135	6374245	4	145	7507780	4
126	1028241	4	136	6286829	4	146	7538161	4
127	1083187	4	137	6268190	4	147	7513119	4
128	1051034	4	138	6360466	4	148	7577516	4
129	1065897	4	139	6329512	4	149	7551199	4
130	1039941	4	140	6321622	4	150	7637538	4

6 Conclusions

This paper focuses on the blocking flow shop problem in order to minimize the total flow time of jobs. First, two constructive algorithms based on the PF algorithm, NHPF1 and NHPF2, are presented. They consist of three steps: selection of the first job in the sequence, construction of the remaining sequence in order to minimize machine timeout, and insertion phase of NEH to try to improve the sequence. However, as the insertion phase can worsen the solution, especially in NHPF1 and NHPF2, the procedures evaluate the sequences obtained before and after the insertion phase to retain the best of both algorithms.

The main difference between NHPF1 and NHPF2 is the selection of the first job in the sequence. NHPF1 chooses the job with a shorter processing time whereas in NHPF2 the front delay generated by the jobs is also considered. The computational evaluation showed not only the good performance of the two algorithms but also the significant influence of the selection of the first job to be scheduled on the quality of the resulting sequence.

Second, four versions of a GRASP are described. The main difference between them is the constructive procedure used to obtain greedy randomized solutions, i.e. HPF1, NHPF1, HPF2 and NHPF2. The GRASPs was combined with a variable neighborhood search that uses the insertion and swap neighborhood. These algorithms were tested against two algorithms proposed for the problem at hand, i.e. the HS algorithm (Wang et al., 2010) and a DABC procedure (Deng et al., 2012), and against the NHPF2 algorithm here proposed. The comparison between them indicated that the presented GRASPs outperform the other algorithms in those sets of more than 20 jobs and that, despite its simplicity, NHPF2 performs better than HS for instances with more than 20 jobs, and even better than DABC for instances with more than 100 jobs.

Finally, the new best known solutions found during this research for most of the Taillard's instances used in the blocking flow shop with total flow time minimization are reported. These new solutions could serve as a basis for comparison for future studies.

One future research direction involves the application of the above algorithms to more complex scheduling problems considering other constraints like setup times, parallel machines or multicriteria scheduling problems. Their simplicity and good performance make them real candidates for adaptation to and implementation in real situations.

References

Davendra, D., & Bialic-Davendra, M. (2013). Scheduling flow shops with blocking using discrete self-organising migration algorithm. *International Journal of Production Research, 51*(8), 2200-2218.

Deng, G., Xu, Z., & GU, X. (2012). A discrete artificial bee colony algorithm for minimizing the total flow time in the blocking flow shop scheduling. *Chinese Journal of Chemical Engineering, 20*(6), 1067-1073.

Feo, T. A., & Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operatons Research Letters, 8*, 67-71.

Framinan, J. M., Leisten, R., & Ruiz-Usano, R. (2005). Comparison of heuristics for flowtime minimisation in permutation flowshops. *Computers & Operations Research, 32*(5), 1237–1254.

Gong, H., Tang, L., & Duin, C. W. (2010). A two-stage flow shop scheduling problem on a batching machine and a discrete machine with blocking and shared setup times. *Computers & Operations Research, 37*(5), 960-969.

Grabowski, J., & Pempera, J. (2000). Sequencing of jobs in some production system. *European Journal of Operational Research, 125*(3), 535-550.

Grabowski, J., & Pempera, J. (2007). The permutation flow shop problem with blocking. A tabu search approach. *Omega, 35*(3), 302-311.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan A.H.G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287-326.

Jeff Wu, C. F., & Hamada, M. (2000). *Experiments. planning, analysis, and parameter design optimization* (Wiley Series in Probability and Statistics ed.). New York: Wiley-Interscience.

Liu, B., Wang, L., & Jin, Y. (2008). An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. *Computers & Operations Research, 35*(9), 2791-2806.

Liu, J., & Reeves, C. R. (2001). Constructive and composite heuristic solutions to the P//∑Ci scheduling problem. *Data Envelopment Analysis, 132*(2), 439–452.

Martinez, S., Dauzère-Pérès, S., Guéret, C., Mati, Y., & Sauer, N. (2006). Complexity of flowshop scheduling problems with a new blocking constraint. *European Journal of Operational Research, 169*(3), 855-864.

McCormick, S. T., Pinedo, M. L., Shenker, S., & Wolf, B. (1989). Sequencing in an assembly line with blocking to minimize cycle time. *Operations Research, 37*, 925-936.

Moslehi, G., & Khorasanian, D. (2013). Optimizing blocking flow shop scheduling problem with total completion time criterion. *Computers & Operations Research, 40*(7), 1874-1883.

Nawaz, M., Enscore Jr, E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega, 11*(1), 91-95.

Pan, Q., & Ruiz, R. (2012). Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research*, *222*(1), 31-43.

Pan, Q.-K., & Ruiz, R. (2013). A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers & Operations Research*, *40*(1), 117–128.

Qian, B., Wang, L., Huang, D. X., & Wang, X. (2009). An effective hybrid DE-based algorithm for flow shop scheduling with limited buffers. *International Journal of Production Research, 47*(1), 1-24.

Qian, B., Wang, L., Huang, D. X., Wang, W., & Wang, X. (2009). An effective hybrid DE-based algorithm for multi-objective flow shop scheduling with limited buffers. *Computers & Operations Research, 36*(1), 209-233.

Rajendran, C. (1993). Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. *International Journal of Production Economics*, *29*(1), 65–73.

Ribas, I., Companys, R., & Tort-Martorell, X. (2013). A competitive variable neighbourhood search algorithm for the blocking flowshop problem. *European J. of Industrial Engineering*, 7(6):729-754

Ribas, I., Companys, R., & Tort-Martorell, X. (2011). An iterated greedy algorithm for the flowshop scheduling problem with blocking. *Omega, 39*(3), 293-301.

Ronconi, D. P. (2004). A note on constructive heuristics for the flowshop problem with blocking. *International Journal of Production Economics*, *87*(1), 39-48.

Sethi, S. P., Sriskandarajah, C., Sorger, G., Blazewicz, J., & Kubiak, W. (1992). Sequencing of parts and robot moves in a robotic cell. *International Journal of Flexible Manufacturing Systems, 4*, 331-358.

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

Wang, L., Pan, Q., & Fatih Tasgetiren, M. (2010). Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms. *Expert Systems with Applications, 37*(12), 7929-7936.

Wang, L., Pan, Q., Suganthan, P. N., Wang, W., & Wang, Y. (2010). A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. *Computers & Operations Research, 37*(3), 509-520.

Wang, L., Zhang, L., & Zheng, D. (2006). An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. *Computers & Operations Research, 33*(10), 2960-2971.

Figure 1. Sequence for a 4-job, 4-machine blocking flow shop

Figure 2. Completion time of jobs J1, J2 or J3 when scheduled in the first position of a sequence

Figure 3. Measurement of front delay

Figure 4. Pseudocode of GRASP(NHPF1) and GRASP(NHPF2)

Figure 5. Pseudocode of the Local Search

Figure 6. Interval plot of ARPD values for each interval and *n* values

Figure 7. ARPD values of each constructive procedure by *n*x*m*

Figure 8. Interval plot of ARPD values of algorithms for *k*=30

Figure 9. Interval plot of ARPD values of algorithms and *n*

Figure 10. Interval plot of ARPD values of algorithms and *m*.

Figure1

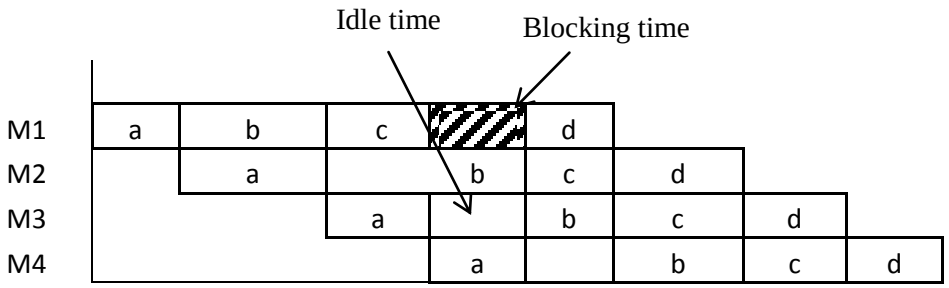


Figure 2

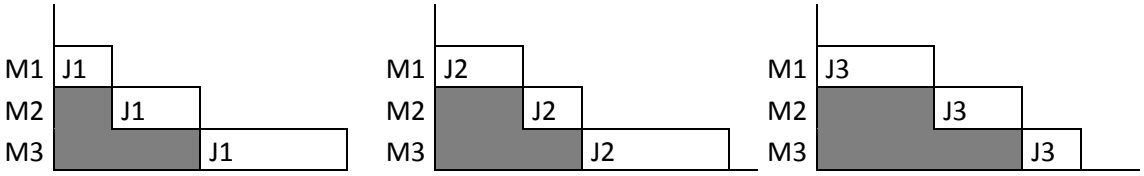


Figure 3

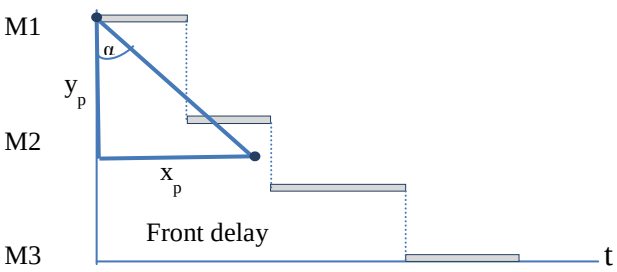


Figure 4

Procedure GRASP(NHPF1) $\sigma_{best} = \infty$ **repeat** **μ**=μ_{min}+(μ_{max}-μ_{min})*random(0,1) σ' ← **N**HPF1(μ) σ'←local_search (σ') **if** cost(σ')< cost(σ_{best}) σ_{best}← σ' **end** **until** stopping_condition met **end**	**Procedure** GRASP(NHPF2) $\sigma_{best} = \infty$ **repeat** **λ**=0.65 **μ**=μ_{min}+(μ_{max}-μ_{min})*random(0,1) σ' ← NHPF2(λ, μ) σ'←local_search (σ') **if** cost(σ')< cost(σ_{best}) σ_{best}← σ' **end** **until** stopping_condition met **end**

Figure 5

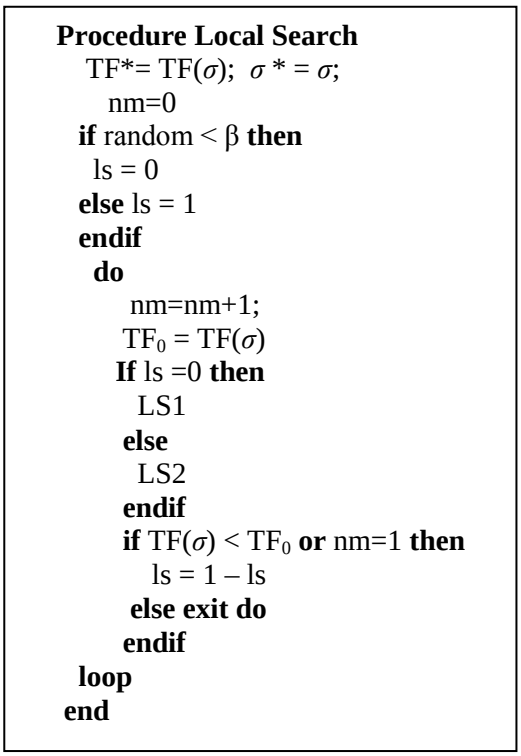

Procedure Local Search
TF*= TF(σ); σ * = σ;
nm=0
if random < β then
ls = 0
else ls = 1
endif
do
nm=nm+1;
TF0 = TF(σ)
If ls =0 then
LS1
else
LS2
endif
if TF(σ) < TF0 or nm=1 then
ls = 1 – ls
else exit do
endif
loop
end

Figure 6

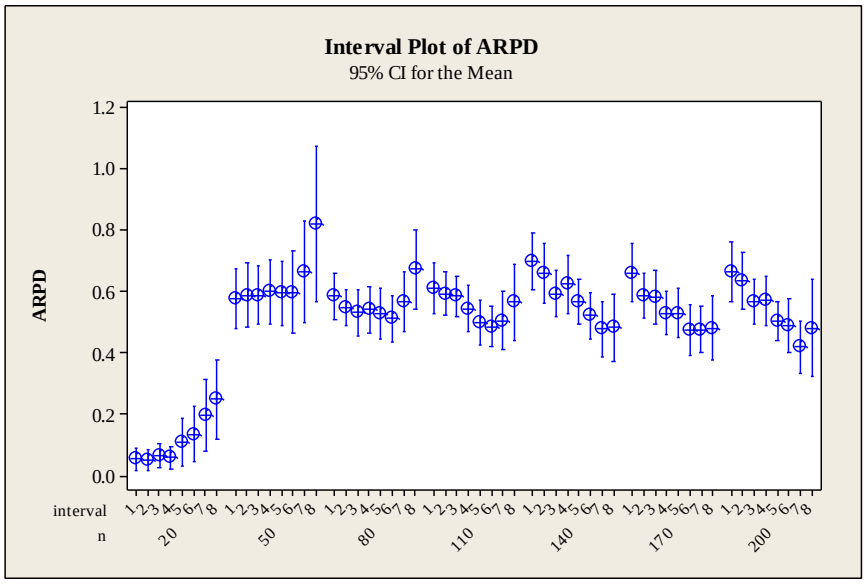

Interval Plot of ARPD
95% CI for the Mean
ARPD
1.2
1.0
0.8
0.6
0.4
0.2
0.0
interval
n
20
50
80
110
140
170
200

Figure 7

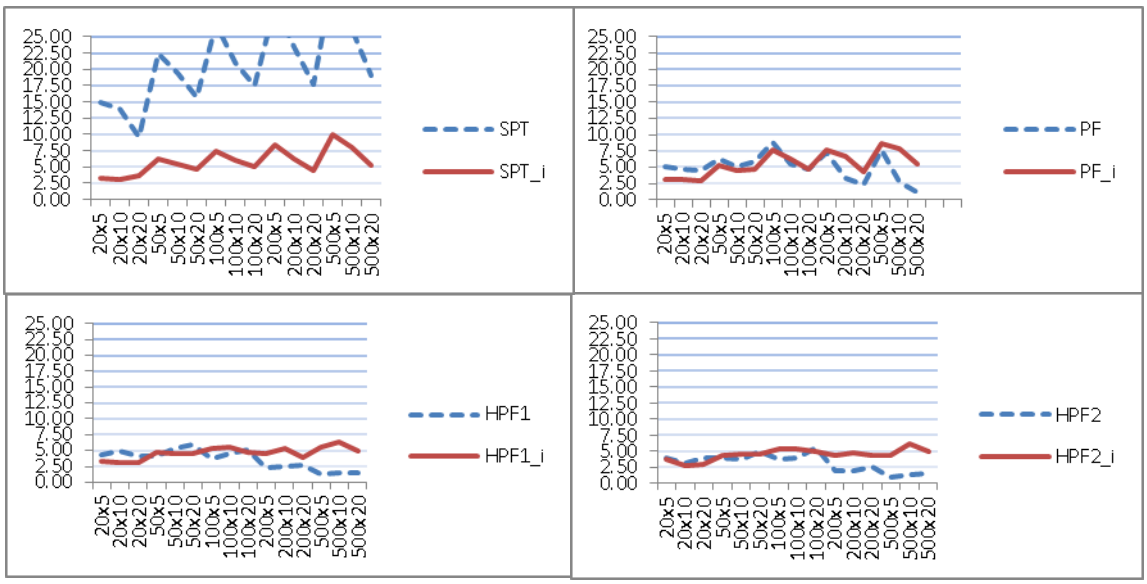

25.00
22.50
20.00
17.50
15.00
12.50
10.00
7.50
5.00
2.50
0.00
20x5
20x10
20x20
50x5
50x10
50x20
100x5
100x10
100x20
200x5
200x10
200x20
500x5
500x10
500x20
SPT
SPT_i
PF
PF_i
HPF1
HPF1_i
HPF2
HPF2_i

Figure 8

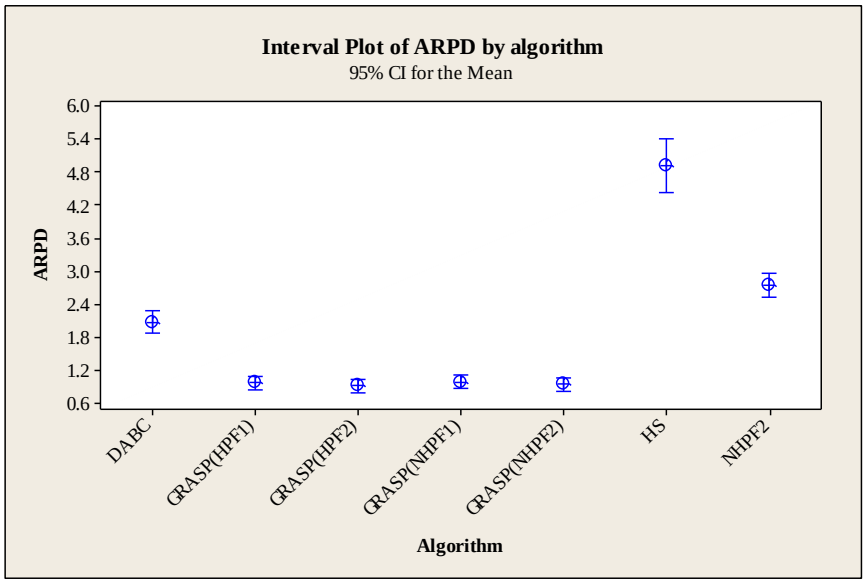

Interval Plot of ARPD by algorithm
95% CI for the Mean
ARPD
6.0
5.4
4.8
4.2
3.6
3.0
2.4
1.8
1.2
0.6
DABC
GRASP(HPF1)
GRASP(HPF2)
GRASP(NHPF1)
GRASP(NHPF2)
HS
NHPF2
Algorithm

Figure 9

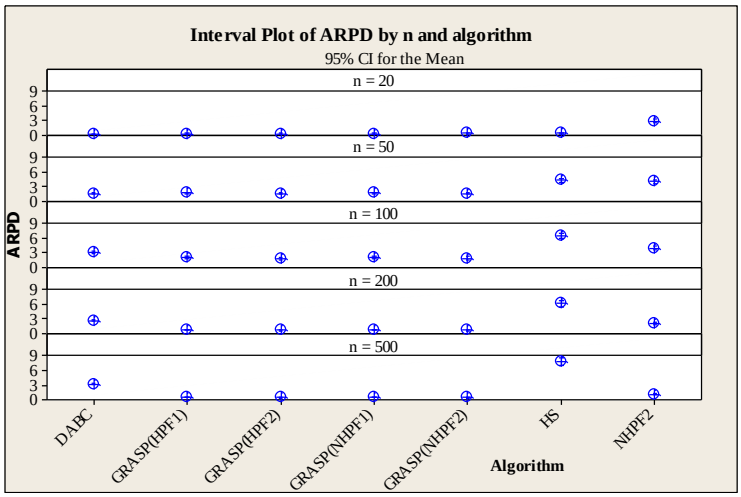
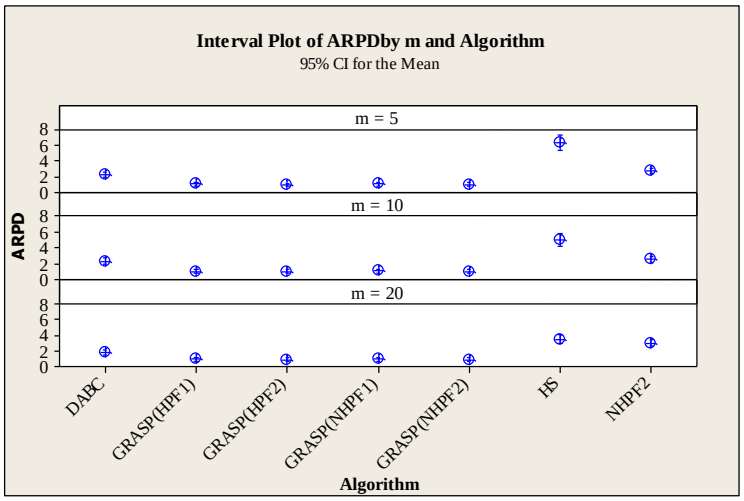

Interval Plot of ARPD by n and algorithm
95% CI for the Mean
n = 20
n = 50
n = 100
n = 200
n = 500
ARPD
0
3
6
9
DABC
GRASP(HPF1)
GRASP(HPF2)
GRASP(NHPF1)
GRASP(NHPF2)
HS
NHPF2
Algorithm

Figure 10


Interval Plot of ARPDby m and Algorithm
95% CI for the Mean
m = 5
m = 10
m = 20
ARPD
0
2
4
6
8
DABC
GRASP(HPF1)
GRASP(HPF2)
GRASP(NHPF1)
GRASP(NHPF2)
HS
NHPF2
Algorithm

Acknowledgments

This work was partially supported by the Spanish Ministry of Science and Innovation under the project RESULT - Realistic Extended Scheduling Using Light Techniques with reference DPI2012-36243-C02-01.