A beam-search-based constructive heuristic for the PFSP to minimise total flowtime*

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

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

Camino de los Descubrimientos s/n, 41092 Seville, Spain, {vfernandezviagas,framinan}@us.es

January 9, 2017

Abstract

In this paper we present a beam-search-based constructive heuristic to solve the permutation flowshop scheduling problem with total flowtime minimisation as objective. This well-known problem is NP-hard, and several heuristics have been developed in the literature. The proposed algorithm is inspired in the logic of the beam search, although it remains a fast constructive heuristic.

The results obtained by the proposed algorithm outperform those obtained by other constructive heuristics in the literature for the problem, thus modifying substantially the state-of-the-art of efficient approximate procedures for the problem. In addition, the proposed algorithm even outperforms two of the best metaheuristics for many instances of the problem, using much lesser computation effort. The excellent performance of the proposal is also proved by the fact that the new heuristic found new best upper bounds for 35 of the 120 instances in Taillard's benchmark.

Keywords: Scheduling, Flowshop, Heuristics, Flowtime, PFSP, Beam Search, total completion time

*Preprint submitted to Computers & Operations Research. http://dx.doi.org/10.1016/j.cor.2016.12.020

†Corresponding author.

1 Introduction

The permutation flowshop scheduling problem, denoted as PFSP, is one of the most studied optimization problems in the literature. In this problem, n jobs must be processed on a shop of m machines following the same order. Since the sequence of jobs must be the same for all machines, the goal of the problem is to find a sequence of jobs optimizing one or several objectives. Traditionally, the most common criteria are: minimisation of makespan (see e.g. Fernandez-Viagas and Framinan, 2014; Ruiz and Stützle, 2007; Nawaz et al., 1983; Dong et al., 2008), minimisation of total flowtime (see e.g. Allahverdi and Aldowaisan, 2002; Framinan et al., 2005; Dong et al., 2013; Rajendran, 1993), and minimisation of total tardiness (see e.g. Vallada et al., 2008; Armentano and Ronconi, 1999; Framinan and Leisten, 2008; Fernandez-Viagas and Framinan, 2015a). Among these, PFSP with makespan minimisation as objective was initially proposed by Johnson (1954), and has been employed in many works since (see e.g. the reviews by Framinan et al., 2004; Ruiz and Maroto, 2005; Reza Hejazi and Saghafian, 2005). Here we focus on total flowtime minimisation, which is considered to be among the most relevant and meaningful for today's dynamic production environments (Liu and Reeves, 2001).

The PFSP to minimise total flowtime is denoted as $Fm|prmu|\sum C_j$ according to the standard notation for scheduling problems (see e.g. Framinan et al., 2014). Since this problem was shown to be strongly NP-hard for two or more machines by Garey et al. (1976), numerous heuristics and metaheuristics have been proposed in the literature trying to achieve good solutions in reasonable CPU times. In an exhaustive analysis, Pan and Ruiz (2013) evaluate the existing algorithms for the problem in order to obtain a so-called efficient set of heuristics assuming as criteria the quality of solutions obtained by each heuristic, and its computational requirements. This efficient set was later improved by Fernandez-Viagas and Framinan (2015b) by means of a constructive heuristic of complexity $O(n^2 \cdot m)$ that can be used as initial solution in composite heuristics.

The goal of this paper is to substantially improve the existing efficient set of heuristics for the $Fm|prmu|\sum C_j$ problem by proposing a new beam-search-based constructive heuristic. The proposed heuristic is inspired by the beam search which was first used in artificial intelligence problems by Lowerre (1976). The beam search is a derivation of the branch-and-bound method

where only a subset of the most promising nodes are kept in each iteration and has been successfully adapted to several scheduling problems in the literature (see e.g. Della Croce and T'kindt, 2002; Valente and Alves, 2005; Valente and Alves, 2008; Valente, 2010). Its performance is highly scalable with the decision interval, thus serving to obtain fast solutions in very short times, or to yield very good-quality solutions if longer CPU times are allowed.

The remainder of the paper is organised as follows: the problem under consideration is described and the state-of-the-art is presented in Section 2. In Section 3, the proposed heuristic is explained in detail and compared with the state-of-the-art heuristics in Section 4. Finally, conclusions are discussed in Section 5.

2 Problem Statement and State of the Art

The problem under study can be stated as follows: n jobs have to be scheduled in a flowshop with m machines. A job j has a processing time p_{ij} on machine i. The completion time of job j on machine i is denoted as C_{ij}, whereas $C_{i[j]}$ indicates the completion time of the job scheduled in position j on machine i. C_{mj} represents the completion time of job j.

As mentioned in Section 1, many heuristics have been proposed for the problem, and an excellent review on these heuristics is provided by Pan and Ruiz (2013). In the following, we just outline the basic aspects of the main heuristics and refer the interested reader to the paper by Pan and Ruiz (2013) for a more detailed description of all existing heuristics.

Among the so-found efficient heuristics, the fastest one is the Raj heuristic by Rajendran (1993), where a sequence is constructed by iteratively trying to insert a non-scheduled job in several positions of an existing partial sequence. More specifically, given a partial sequence of k jobs, positions from $\lceil \frac{k}{2} \rceil$ to $k+1$ are tried. The list of non-scheduled jobs is arranged in non descending order of indicator T_j (1):

$$T_j = \sum_{i=1}^{m} (m - j + 1) \cdot p_{ij} \tag{1}$$

A different approach is adopted by the $LR(x)$ heuristic by Liu and Reeves (2001) where x final sequences are constructed by iteratively adding jobs one by one at the end of x partial sequences.

The job to be inserted in iteration k is chosen so its value of indicator ξ_{jk} –see Equation (2)– is the minimum among the unscheduled jobs. The first job of the i final sequence (with $i \in \{1, \ldots, x\}$) is the job with ith minimal indicator ξ_{j0}.

$$\xi_{jk} = (n - k - 2) \cdot IT_{jk} + AT_{jk} \tag{2}$$

In Equation (2), IT_{jk} estimates the weighted idle time induced if job j is scheduled in the last position of the partial sequence (i.e. $k+1$). AT_{jk} is the artificial flowtime, which is the sum of the completion time of job j plus the completion time of an artificial job p whose processing time on machine i equal to the average processing time of the unscheduled jobs on that machine (excluding job j). More specifically, IT_{jk} and AT_{jk} are defined as:

$$IT_{jk} = \sum_{i=2}^{m} \frac{m \cdot max\{C_{i-1,j} - C_{i,[k]}, 0\}}{i + k \cdot (m - i)/(n - 2)} \tag{3}$$

$$AT_{jk} = C_{mj} + C_{mp}$$

The NEH heuristic was originally proposed by Nawaz et al. (1983) for the $Fm|prmu|C_{max}$ problem and lately adapted for the $Fm|prmu|\sum C_j$ problem by Framinan et al. (2002). In the NEH, jobs are initially sorted according to a non-descending sum of their processing times. Using this order, each unscheduled job is inserted in the partial sequence in the position that minimises its total flowtime.

In view of the good performance of both $LR(x)$ and NEH, Pan and Ruiz (2013) propose the composite $LR-NEH(x)$ heuristic which schedules the first $3 \cdot n/4$ jobs of x sequences according to a $LR(x)$ procedure, and the remaining jobs according to the NEH.

The rest of the efficient heuristics in the set identified by Pan and Ruiz (2013) include a local search method after the construction of the initial solution. More specifically, regarding local search methods based on job insertion, the RZ heuristic by Rajendran and Ziegler (1997) uses the ascending order of total processing times as the initial sequence and improves that sequence by inserting each job of the sequence in the rest of positions and updating the sequence if better solution is found (this improvement phase is denoted in the following as RZ). The $IC1$ heuristic

by Li et al. (2009) implements the RZ local search method after the LR heuristic until no further improvement (denoted as iRZ) is found.

Regarding local search methods based on the exchange of positions among jobs, Liu and Reeves (2001) propose the heuristic $LR(x) - FPE(y)$, in which x sequences are generated according to the $LR(x)$ procedure and then, the solutions are improved by employing a Forward Pairwise Exchange (FPE) procedure, i.e. each job in position k in the sequence is exchanged with each one of the y jobs in positions $k+1, k+2, \ldots, k+y$. The procedure is repeated until there are no more improvements in a complete iteration. The $IC2$ and $IC3$ heuristics, proposed by Li et al. (2009), are similar to $IC1$, but at the end of each iteration the FPE and $FPE-R$ procedures are performed, being $FPE-R$ a variant of FPE where the insertion procedure is restarted after an improvement of the solution. Finally, regarding the combination of insertion and exchange movements, Pan and Ruiz (2013) propose two variants –denoted as $PR2(x)$ and $PR4(x)$– of a VND local search method where the resulting solutions are embedded in $LR-NEH$ procedures until x iterations are reached, or the CPU time exceeds a given value. Other variations, such as the PR1 heuristic, which performs the iRZ procedure instead of the VND method to improve each sequence obtained by the $LR-NEH$ procedure were not found to be efficient. Recently, Abedinnia et al. (2016) present a new simple heuristic which outperforms the simple heuristic of Laha and Sarin (2009). However, their results in term of quality of solution and computational effort are still far from this set of efficient heuristics.

All aforementioned heuristics have at least a complexity of $O(n^3 \cdot m)$, and most of them use the LR heuristic to generate a seed solution. Using a similar procedure to that of the LR heuristic, Fernandez-Viagas and Framinan (2015b) propose a fast constructive heuristic –denoted FF in the following– inserting, step by step, jobs at the end of the sequence according to the index ξ'_{jk} (see Equation 4) in order to reduce the complexity to $O(n^2 \cdot m)$:

$$\xi'_{jk} = \frac{(n-k-2)}{4} \cdot IT'_{jk} + C_{mj} \tag{4}$$

where IT'_{jk} is defined by (5):

$$IT'_{jk} = \sum_{i=2}^{m} \frac{m \cdot max\{C_{i-1,j} - C_{i,[k]}, 0\}}{i - 1 + k \cdot (m - i + 1)/(n - 2)} \quad (5)$$

By means of this new FF heuristic, it is possible to obtain a completely new set of efficient heuristics by replacing LR by FF in the rest of heuristics. More specifically, the new set includes the FF, $FF-FPE$, $FF-IC1$, $FF-IC2$, $IC2$, $IC3$ and $PR1$ heuristics obtained by replacing LR by FF in the corresponding heuristics. In the next section, we propose a new heuristic which can substantially improve the above described set of efficient heuristics.

3 Proposed Heuristic

In this section, we propose a Beam-Search-based Constructive Heuristic –denoted $BSCH$–, for the PFSP to minimise total flowtime. $BSCH$ works with several candidate nodes in parallel in each iteration. The number of selected nodes is controlled by the parameter x (beam width). The heuristic operates performing $n-1$ iterations. At iteration k, each selected node l ($l \in \{1, \ldots, x\}$) is formed by a set, $\mathcal{S}_k^l$, of k scheduled jobs (s_{jk}^l denotes the job placed in position j of selected node l in iteration k). Consequently, for each selected node l in iteration k there is a set $\mathcal{U}_k^l$ of $n-k$ unscheduled jobs. Let us denote u_{jk}^l the jth unscheduled job of selected node l in iteration k.

For each iteration $k \in \{1, \ldots, n-1\}$, $|\mathcal{U}_k^l|$ candidate nodes can be obtained from each selected node l by inserting each one of the jobs in $\mathcal{U}_k^l$ in position $k+1$ of $\mathcal{S}_k^l$. In total, $(n-k) \cdot x$ candidate nodes can be obtained. The idea is to retain the most promising x candidate nodes for the next iteration (selected nodes). The rest of the nodes are discarded for the next iterations. However, comparing candidate nodes may be or may be not straightforward depending on the specific situation:

- If the candidate nodes to be compared have been obtained by appending different jobs in $\mathcal{U}_k^l$ to a same node l, then their corresponding partial sequences are identical with the exception of the last job. Therefore, they can be compared in terms of the completion time of the added job, or of the new idle time induced by the added job.

- If the candidate nodes to be compared have been obtained from different nodes –e.g. one candidate node is the subsequence $(1, 2)$, and other candidate node is subsequence $(2, 3)$–, both the scheduled and the unscheduled jobs are different for each candidate node. In such case, it is useless to perform a straightforward comparison among candidate nodes taking into account either the job to be inserted, or just the scheduled jobs.

Clearly, the key to select the best x candidate nodes is to be able to compare partial sequences composed of different jobs. Since in iteration k, a candidate node is formed by partial sequence $\mathcal{S}_k^l$ of selected node l plus a job inserted in position $k+1$, both l and the inserted job would contribute to the value of the flowtime of a final sequence obtained from this candidate.

Regarding the contribution of the inserted job, there are two elements that largely influence the value of the sum of completion times in the complete sequence (Fernandez-Viagas and Framinan, 2015b), i.e.: the weighted idle time induced by the new job u_{jk}^l inserted, and the completion time of the new job u_{jk}^l. Note that the evaluation of these elements can be done in $O(m)$.

Regarding the contribution of each selected node l in iteration k to the flowtime of the final sequence –denoted F_{kl} or forecast index in the following–, it is clear that such contribution is related to both scheduled and unscheduled jobs. On the one hand, the contribution due to the scheduled jobs can be computed by means of a function of the idle times and completion times of the previous jobs. On the other hand, an 'artificial' completion time, denoted as $CT\lambda_{kl}$, can be used to identify the contribution of the unscheduled jobs. The computation of F_{kl} is developed in Section 3.5.

Hence, steps of the constructive heuristic can be summarised as follows:

- Obtain a set of nodes
- During n iterations:
  - Generate candidate nodes
  - Evaluate candidate nodes
  - Select the best x candidate nodes
  - Update forecast index

These steps are elaborated in the next subsections.

3.1 Generation of the Initial Nodes

Jobs are initially sorted in non descending order of indicator $\xi'_{j,0}$ (see Section 2) breaking ties in favor of jobs with lower $IT'_{j,0}$. Let us denoted by α_i ($\alpha := (\alpha_1, ..., \alpha_i, ..., \alpha_n)$) the component i of that sorted list. Hence, to obtain the first x nodes (consisting of one job), job in position l of the sorted list is placed in the first position of the partial sequence $s^l_{1,1}$ of the selected node l ($s^l_{1,1} = \alpha_l$). The rest of the jobs forms the unscheduled jobs of this selected node l, i.e. $u^l_{j,1}$ with $j \in \{1, \ldots, n-1\}$.

3.2 Candidate Nodes Generation

New candidate nodes are formed by adding an unscheduled job at the end of the partial sequence of each selected node. More specifically, from each selected node $l \in \{1, \ldots, x\}$, $n-k$ candidate nodes are obtained at iteration k where each candidate j is obtained from selected node l by adding the jobs in $\mathcal{U}^l_k$ at the end of the scheduled jobs.

3.3 Candidate Nodes Evaluation

Once candidate nodes are formed, they are evaluated. This evaluation is performed taking into account two factors:

- Influence from the selected node: As already discussed, the influence of selected node l in iteration k is measured by means of the forecast index F_{kl} which is explained further in Section 3.5.

- Influence from the inserted job: This influence is due to the new job inserted at the end of the scheduled jobs and is measured by CT_{jkl} the completion time of the unscheduled job u^l_{jk}, which is the additional completion time incurred when inserting job u^l_{jk} in the selected node, i.e.:

$$CT_{jkl} = C_{mu^l_{jk}}$$

and by IT_{jkl} the weighted idle time induced by the insertion of job u^l_{jk}:

$$IT_{jkl} = \sum_{i=2}^{m} \frac{m \cdot max\{C_{i-1,u^l_{jk}} - C_{i,[k]}, 0\}}{i - 1 + k \cdot (m - i + 1)/(n - 2)} \tag{6}$$

Hence, in iteration k, given a selected node l, the insertion of unscheduled job u^l_{jk} is evaluated according to the following index:

$$B_{jkl} = F_{kl} + c \cdot CT_{jkl} + IT_{jkl} \cdot (n - k - 2) \tag{7}$$

The parameter c has been introduced in the equation in order to balance the completion time and the idle time of the new introduced job (in Section 3.6, the calibration of this parameter is addressed). Additionally, the idle time is weighted by $(n - k - 2)$ to reduce its importance as indicator as the sequence contains more jobs.

In the beam search literature (see e.g. Sabuncuoglu and Bayiz, 1999), this type of evaluation method where each unscheduled job is taken into account is denoted as *total cost evaluation function*. However, note that, in our case, in order to speed up the computation of the cost function, an *estimate* of the actual cost function is carried out.

3.4 Candidate Nodes Selection

The procedure to select the candidate nodes that would constitute the selected nodes of the next iteration is very simple: we adopt an elitist selection procedure where the x candidate nodes with the lowest values of B are selected, i.e. in iteration k we look for the combination of j and l achieving the lowest values of B_{jkl} as defined in Equation (7). The rest of candidate nodes are removed from the population, and the chosen candidate nodes are denoted as the selected nodes for the next iteration. Let us denote by $branch[l']$ and $job[l']$ the value of l and j respectively of the l'th best B_{jkl} in iteration k.

3.5 Forecasting Phase

The Forecast Index, F, is used to be able to compare candidate nodes with different un- and scheduled jobs. It balances the following indicators:

1. the idle time of each scheduled job in the candidate node,
2. the completion time of each scheduled job in the candidate node, and
3. the completion time of the unscheduled jobs in the candidate node.

The influence of 1) and 2) changes across the iterations of the algorithm. Recall that the influence of the idle time allows us to compare candidate nodes with different jobs. In the first iterations there are few scheduled jobs, and these scheduled jobs may be quite different. Therefore, the idle time between jobs is expected to have a larger influence in the comparison between nodes, as compared to the sum of completion times (which is strongly schedule-dependent). In contrast, in the last iterations the candidate nodes are almost complete sequences, so they are very similar in terms of scheduled jobs and therefore, a direct evaluation of the completion times of the jobs to compare nodes would be more related to the final objective. Thereby, in the equation of the forecast index, the cumulated idle time, denoted as SIT (8) is reduced with the number of scheduled jobs (it is multiplied by $n-k-2$), while the cumulated completion time, so-called SCT (9), remains the same along the iterations. More specifically:

$$SIT_{k,l'} = \frac{n-b}{n} \cdot \left[SIT_{k-1,branch[l']} + IT_{job[l'],k,branch[l']} \cdot (n-k-2) \right], \forall\, k = 1,\ldots,n-1, l' = 1,\ldots,x \tag{8}$$

$$SCT_{k,l'} = SCT_{k-1,branch[l']} + CT_{job[l'],k,branch[l']} + CT\lambda_{k,branch[l']}, \forall\, k = 1,\ldots,n-1, l' = 1,\ldots,x \tag{9}$$

where $SIT_{0,l'} = SCT_{0,l'} = 0, \forall\, l' = 1,\ldots,x$ and $CT\lambda_{k,l}$ is the completion time of an artificial job placed at the end of the sequence of the selected node l in the iteration k. The processing times of this artificial job are equal to the average processing times of the unscheduled jobs ($u^l_{j,k}$ $\forall\, j$).

Taking these indicators into account, the forecast index can be then defined as follows:

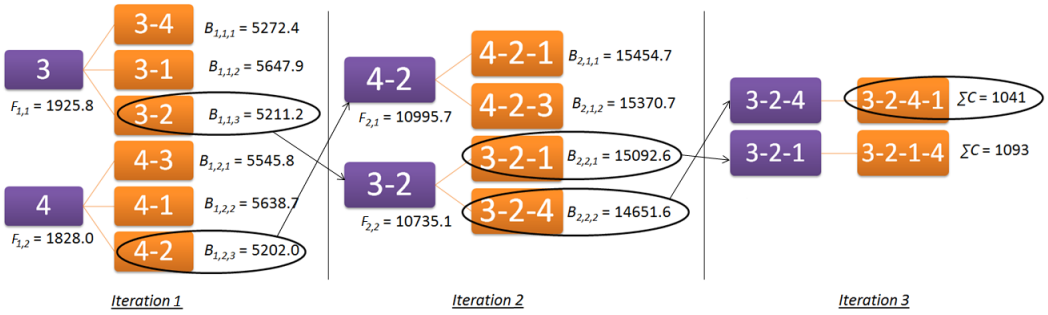

Figure 1: Example of BSCH

$$F_{k,l'} = a \cdot SCT_{k,l'} + SIT_{k,l'}, \forall\, k = 1, \ldots, n-1, l' = 1, \ldots, x \tag{10}$$

where a, and b are parameters designed to better balance the components of the forecast index. Parameter a balances the influence of SIT and SCT. Parameter b is introduced in fraction $(n-b)/n$ of SIT in order to diminish the weight of idle time with the increase of iterations, given that 1) the idle time of the last jobs is less important than that of the first ones given the flowtime objective, and 2) the importance of the cumulated idle time as indicator also decreases as the number of scheduled jobs is higher.

The calibration of a and b is discussed in Section 3.6. An example of the algorithm is presented in Figure 1. We use the third instance of the benchmark by Taillard (1993) where the last 16 jobs have been removed, and consider only the first 4 jobs. Selected nodes are shown in lilac while candidate nodes are shown in orange. The pseudo-code of the algorithm is shown in Figure 2.

3.6 Experimental parameter tuning

Parameters a, b and c have been included to better adjust the performance of the proposed heuristic. In this subsection, a full factorial design of experiments is performed to set up proper values for these parameters. For each of them, the following levels are tested

- $a \in \{1, 3, 5, 7, 9, 11, 13\}$
- $b \in \{0, 1, 2, 3, 4, 5, 6\}$
- $c \in \{1, 3, 5, 7, 9, 11, 13\}$

Procedure *BSCH(x)*

//Initial Order

Determination of $IT'_{j,0}$, $CT'_{j,0}$ and ξ'_{j0};

$IT_{j,0,l} = IT'_{j,0}$ and $CT_{j,0,l} = CT'_{j,0}$ $\forall\, l$;

$\alpha :=$ Jobs ordered according to non-decreasing $\xi'_{j,0}$ breaking ties in favor of jobs with lower $IT'_{j,0}$;

Update $\mathcal{S}_1^l$ $(s_{1,1}^l = \alpha_l)$ $\forall\, l$ and $\mathcal{U}_1^l$ with the remaining jobs.

Determination of $CT\lambda_{0,l}$ $\forall\, l$. Note that the processing times of the artificial job for selected node l is equal to the average processing times of all jobs with the exception of $s_{1,1}^l$;

for $l = 1$ **to** x **do**

$SIT_{1,l} = \frac{n-b}{n} \cdot \big(IT_{alpha[l],0,l} \cdot (n - 0 - 2)\big)$;

$SCT_{1,l} = CT_{alpha[l],0,l} + CT\lambda_{,0,l}$;

$F_{1,l} := a \cdot SCT_{1,l} + SIT_{1,l}$;

end

for $k = 1$ **to** $n - 1$ **do**

//Candidate Nodes Creation

Determination of IT_{jkl}, CT_{jkl};

//Candidate Nodes Evaluation

$B_{jkl} := F_{kl} + c \cdot CT_{jkl} + IT_{jkl}$, $\forall\, l = 1, \ldots, x$ and $\forall\, j = 1, \ldots, n - x$;

//Candidate Nodes Selection

for $l' = 1$ **to** x **do**

Determination of the l'-th best candidate node according to non-decreasing B_{jkl} in iteration k. Denote by $branch[l']$ the value of the index l of that candidate node and by $job[l']$ the value of j;

end

//Forecasting Phase. Update of the Forecast Index

for $l' = 1$ **to** x **do**

Update $\mathcal{S}_{k+1}^{l'}$ and $\mathcal{U}_{k+1}^{l'}$ by removing job $u_{job[l'],k}^{branch[l']}$ from $\mathcal{U}_k^{branch[l']}$ and including in $\mathcal{S}_k^{branch[l']}$.

Determination of $CT\lambda_{k,branch[l']}$ for new selected node l' formed by the old selected node $branch[l']$ with job $job[l']$. Note that the processing times of the artificial job are equal to the average processing times of all unscheduled jobs $(U_{k+1}^{l'})$;

$SIT_{k+1,l'} = \frac{n-b}{n} \cdot \Big(SIT_{k,branch[l']} + IT_{job[l'],k,branch[l']} \cdot (n - k - 2)\Big)$;

$SCT_{k+1,l'} = SCT_{k,branch[l']} + CT_{job[l'],k,branch[l']} + CT\lambda_{k,branch[l']}$;

$F_{k+1,l'} = a \cdot SCT_{k+1,l'} + SIT_{k+1,l'}$;

end

end

//Final evaluation

Evaluate the flowtime of the scheduled jobs of each selected node and return the least one.

end

Figure 2: BSCH

Source	Significance
Parameter a	0.000
Parameter b	1.000
Parameter c	0.007

Table 1: Kruskal-Wallis for the parameters a, b and c

representing 343 combinations of values. For each combination, five instances have been generated for several values of n and m, $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$, where the processing times of each job in each machine is uniformly distributed between 1 and 99. A non-parametric Kruskal-Wallis analysis is performed since normality and homoscedasticity assumptions required for ANOVA were not fulfilled. In the experiments, $x = n/10$ in order to avoid excessive CPU time requirements for parameter tuning. Results are shown in Table 1, indicating that there are significant differences between the levels of parameters a and c, but not for parameter b. The best combination is obtained for $a = 9$, $b = 3$ and $c = 7$. These values are used for the BSCH heuristic in the next section regardless the value of x.

4 Computational Evaluation

The proposed heuristic is compared with the current set of efficient heuristics formed by 17 heuristics (see Section 2). In order to have a fair comparison, each heuristic is again implemented under the same computers conditions which means:

- Using the same computer in the computational evaluation (a Intel Core i7-3770 PC with 3.4 GHz and 16GB RAM),
- the same programming language (C# under Visual Studio 2013), and
- the same libraries and common functions for all heuristics.

Experiments have been performed for the 120 instances of the benchmark by Taillard (1993) which is composed of 12 problem sizes varying the number of jobs and machines according to $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 15, 20\}$ respectively, with 10 instances for each size.

Processing times are uniformly distributed from 1 to 99 in this testbed. To better fit the computational time of each heuristic, 5 runs are carried out for each instance and the average values are collected.

Additionally, the parameter x of the proposed heuristic must be set. As shown in Section 3, x indicates the number of selected nodes in each iteration and therefore, it is proportional to the CPU time required by the heuristic. For $x > n$, additional indications in the first iteration of the algorithm would have to be provided (i.e. at least it should be indicated which is the first job of the last $x - n$ selected nodes after the first iteration), so here we restrict to $x \in \{1, n\}$. More specifically, we use the values of x also employed in the literature for the LR heuristic, i.e. $x \in \{2, 5, 10, 15, n/10, n/m, n\}$ (see e.g. Liu and Reeves, 2001, Pan and Ruiz, 2013, and Fernandez-Viagas and Framinan, 2015b). Note that $x = 1$ has been removed from the analysis since $BSCH(1)$ is equivalent to $FF(1)$ (with a different combination of parameters), so it is already included in the computational evaluation.

4.1 Comparison between BSCH and the efficient heuristics

The comparison among the heuristics is performed in terms of quality of the solutions and computational effort. On the one hand, the former is commonly evaluated by means of the Relative Percentage Deviation $RPD1$, which is defined for heuristic h in instance i as:

$$RPD1_{ih} = \frac{C_{sum}^{ih} - \min_{1 \leq h \leq H} C_{sum}^{ih}}{\min_{1 \leq h \leq H} C_{sum}^{ih}} \cdot 100, \forall\, i = 1, \ldots, I, h = 1, \ldots, H \tag{11}$$

where H is the number of heuristics considered in the evaluation, I is the number of instances in the test bed, and C_{sum}^{ih} is the total flowtime obtained by heuristic h in instance i. Note that $ARPD1$ indicates Average $RPD1$. On the other hand, the most common indicator for the computational effort is the average CPU time. However, Fernandez-Viagas and Framinan (2015b) detected that this indicator presents several problems when used to evaluate heuristics with different stopping criteria, and proposed the RPT' (Relative Percentage Time) indicator instead:

$$RPT'_{ih} = \frac{T_{ih} - ACT_i}{ACT_i}, \forall\, i = 1, \ldots, I, h = 1, \ldots, H \tag{12}$$

where T_{ih} is the CPU time required by heuristic h in instance i and:

$$ACT_i = \sum_{h=1}^{H} T_{ih}/H, \forall\, i = 1, \ldots, I \tag{13}$$

In this paper, a slightly different indicator, denoted as RPT, is used to be able to graphically represent the results in logarithmic scale:

$$RPT_{ih} = \frac{T_{ih} - ACT_i}{ACT_i} + 1, \forall\, i = 1, \ldots, I, h = 1, \ldots, H \tag{14}$$

The average value of RPT, i.e. $ARPT$, can be defined as follows:

$$ARPT_h = \sum_{i=1}^{I} \frac{RPT_{ih}}{I}, \forall\, h = 1, \ldots, H \tag{15}$$

Nevertheless, in order to provide additional information of the experiments, raw CPU times are also used together with $ARPT$.

The $RPD1$ values obtained for each algorithm are shown in Tables 2 and 3. The last row indicates the average value, i.e. the $ARPD1$ for each algorithm. As it can be seen, the $ARPD1$ of the actual set of efficient heuristics ranges from 3.84 to 1.22, where the best one (1.22) is obtained by FF-PR1(15). Regarding $BSCH$, the worst $ARPD1$ is 2.51 while the best one is 0.19. In order to be able to perform a fair comparison among heuristics, CPU times (in seconds) are summarised in Tables 4 and 5 (the last two rows represent the average CPU time and the $ARPT$ respectively). The average values are indicated in Table 6 and graphically shown in Figure 4 using $ARPT$ as a measure of the computational effort, as well as in Figure 3 using Average CPU times.

Considering $ARPT$, the actual set of efficient heuristics is updated by including a complete new set of heuristics, all of them including $BSCH$ for different values of x. The following conclusions can be obtained:

- $BSCH(2)$ (with $ARPD1 = 2.51$) improves heuristics $FF(n/m)$, $FF(n/10)$ and $FF(n/10)-$

$FPE(1)$ with $ARPD1$ equal to 3.11, 3.02 and 2.70 respectively, while using less $ARPT$.

- $BSCH(n/m)$, $BSCH(5)$ and $BSCH(n/10)$ with $ARPD1$ 1.46, 1.35 and 1.21 respectively outperform $FF(2) - FPE(n/10)$ and $FF(n/10) - FPE(n/10)$ using less $ARPT$.

- $BSCH(10)$, with $ARPD1$ and $ARPT$ equal to 0.88 and 0.13, clearly outperforms $FF(15) - FPE(n/10)$, which has an $ARPD1$ of 2.35 and an $ARPT$ of 0.17.

- $BSCH(15)$ ($ARPD1 = 0.64$) outperforms with less computational effort $FF(n/10) - FPE(n)$, $FF(n/m) - FPE(n)$, $FF - IC1$ and $FF - IC2$ which have a minimal $ARPD1$ of 1.61.

- The best heuristic, $BSCH(n)$, with $ARPD1 = 0.19$ clearly outperforms heuristics $IC2$, $FF - IC3$, $IC3$, $FF - PR1(5)$, $FF - PR1(10)$ and $FF - PR1(15)$.

As a consequence, it can be stated that our proposal outperforms the up-to-now efficient heuristics for the problem.

In order to establish the statistical significance of these results, Holm's procedure (Holm, 1979) is used where each hypothesis is analysed using a non-parametric Mann-Whitney test (see e.g. Pan et al., 2008). In Holm's procedure, the hypotheses are sorted in non-descending order of the p-values found in the Mann-Whitney test. The hypothesis i is rejected if its p-value is lower than $\alpha/(k - i + 1)$ where k is the total number of hypotheses. The results of the Holm's procedure are shown in Table 7, where the fourth and sixth columns indicate if the hypothesis is rejected (denoted as R in such case) by Mann-Whitney and/or Holm's procedure. As can be seen, hypothesis $BSCH(2) = FF(n/10) - FPE(1)$ is the only one that cannot be rejected by Holm's procedure, but it has to be noted that the computational effort required by $FF(n/10) - FPE(1)$ is much higher to that by $BSCH(2)$. In summary, it can be concluded that $BSCH(n/10)$, $BSCH(10)$, $BSCH(15)$ and $BSCH(n)$ are statistically efficient and that $BSCH(2)$ is not inefficient. Note that $BSCH(2)$ would be statistically efficient when considering the Pareto frontier using the average CPU time instead of the $ARPT$.

Instance	FF(1)	FF(2)	FF($n/10$)	FF(n/m)	FF(2)-FPE($n/10$)	FF(15)-FPE($n/10$)	FF($n/10$)-FPE(1)	FF($n/10$)-FPE($n/10$)	FF($n/10$)-FPE(n)	FF(n/m)-FPE(n)	FF-IC1
20 x 5	3.20	2.71	2.71	2.45	1.90	1.54	1.95	1.90	1.24	1.42	1.30
20 x 10	3.20	3.38	3.38	3.38	2.37	1.94	2.49	2.37	1.76	1.76	1.07
20 x 20	3.06	2.68	2.68	3.26	1.91	2.24	2.13	1.91	1.54	1.79	1.02
50 x 5	2.28	2.10	2.03	2.01	1.51	1.55	1.91	1.62	1.38	1.36	1.05
50 x 10	3.49	3.60	3.12	3.12	2.82	2.64	2.84	2.63	2.26	2.26	1.59
50 x 20	3.19	3.44	3.15	3.44	2.76	2.48	2.83	2.48	1.96	2.16	1.50
100 x 5	1.92	1.61	1.55	1.55	1.41	1.41	1.50	1.41	1.25	1.25	1.25
100 x 10	3.63	3.70	3.34	3.34	2.76	2.55	3.17	2.59	2.41	2.41	2.20
100 x 20	5.55	5.24	4.06	4.42	4.02	3.13	3.76	3.28	2.40	2.53	1.84
200 x 10	3.59	3.24	2.95	2.95	2.41	2.37	2.85	2.35	2.25	2.25	2.14
200 x 20	5.93	5.46	4.13	4.32	4.33	3.42	3.89	3.42	3.03	3.07	2.69
500 x 20	4.12	3.83	3.12	3.15	3.21	2.90	3.05	2.75	2.54	2.55	2.46
Average	3.84	3.42	3.02	3.11	2.62	2.35	2.70	2.39	2.00	2.07	1.68

Table 2: RPD1 of heuristics (I)

Instance	FF-IC2	FF-IC3	IC2	IC3	FF-PR1(5)	FF-PR1(10)	FF-PR1(15)	BSCH(2)	BSCH(5)	BSCH(10)	BSCH(15)	BSCH($n/10$)	BSCH(n/m)	BSCH(n)
20 x 5	1.20	1.20	0.65	0.57	0.46	0.34	0.26	2.44	1.44	0.82	0.61	2.44	1.23	0.96
20 x 10	1.16	1.16	1.09	0.98	0.67	0.53	0.29	2.28	1.03	0.82	0.43	2.28	2.28	0.46
20 x 20	1.02	1.01	1.13	1.20	0.24	0.12	0.06	1.96	0.81	0.62	0.44	1.96	3.34	0.46
50 x 5	0.91	0.91	1.06	1.05	0.95	0.88	0.88	1.74	0.81	0.54	0.45	0.81	0.54	0.12
50 x 10	1.69	1.66	1.59	1.58	1.33	1.27	1.20	2.50	1.38	0.65	0.44	1.38	1.38	0.06
50 x 20	1.45	1.45	1.12	1.08	0.86	0.81	0.71	2.65	1.15	0.96	0.73	1.15	2.65	0.10
100 x 5	1.12	1.11	1.15	1.15	1.20	1.16	1.14	1.39	0.93	0.54	0.50	0.54	0.39	0.07
100 x 10	1.96	2.00	1.82	1.76	1.86	1.76	1.74	2.92	1.54	0.80	0.53	0.80	0.80	0.04
100 x 20	1.81	1.79	2.18	2.04	1.67	1.48	1.41	3.89	2.20	1.29	0.87	1.29	2.20	0.00
200 x 10	2.01	1.99	2.07	2.04	2.06	1.92	1.92	2.34	1.31	0.87	0.71	0.58	0.58	0.00
200 x 20	2.61	2.56	2.60	2.59	2.50	2.41	2.33	3.36	1.84	1.43	1.09	0.80	1.43	0.00
500 x 20	2.40	2.36	2.49	2.48	2.66	2.66	2.66	2.69	1.70	1.19	0.92	0.49	0.69	0.00
Average	1.61	1.60	1.58	1.54	1.37	1.28	1.22	2.51	1.35	0.88	0.64	1.21	1.46	0.19

Table 3: RPD1 of heuristics (II)

Instance	FF(1)	FF(2)	FF($n/10$)	FF(n/m)	FF(2)-FPE($n/10$)	FF(15)-FPE($n/10$)	FF($n/10$)-FPE(1)	FF($n/10$)-FPE($n/10$)	FF($n/10$)-FPE(n)	FF(n/m)-FPE(n)	FF-IC1
20 x 5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20 x 10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20 x 20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50 x 5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01
50 x 10	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.02	0.03
50 x 20	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.01	0.03	0.03	0.04
100 x 5	0.00	0.00	0.01	0.01	0.01	0.02	0.01	0.01	0.06	0.07	0.08
100 x 10	0.00	0.00	0.01	0.01	0.03	0.04	0.01	0.03	0.15	0.14	0.23
100 x 20	0.00	0.00	0.02	0.01	0.05	0.08	0.03	0.06	0.33	0.33	0.60
200 x 10	0.01	0.01	0.09	0.09	0.22	0.27	0.12	0.30	1.30	1.31	2.13
200 x 20	0.01	0.02	0.19	0.09	0.54	0.57	0.23	0.63	3.26	3.17	5.69
500 x 20	0.06	0.12	2.85	1.42	9.14	9.30	3.12	10.43	57.79	55.96	81.24
Average	0.01	0.01	0.26	0.14	0.83	0.86	0.29	0.96	5.25	5.09	7.51
ARPT	0.01	0.01	0.02	0.03	0.06	0.17	0.04	0.08	0.34	0.34	0.58

Table 4: Computational times of heuristics I

Instance	FF-IC2	FF-IC3	IC2	IC3	FF-PR1(5)	FF-PR1(10)	FF-PR1(15)	BSCH(2)	BSCH(5)	BSCH(10)	BSCH(15)	BSCH($n/10$)	BSCH(n/m)	BSCH(n)
20 x 5	0.00	0.00	0.00	0.00	0.01	0.02	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20 x 10	0.00	0.00	0.00	0.00	0.01	0.03	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20 x 20	0.00	0.00	0.00	0.00	0.02	0.03	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50 x 5	0.02	0.03	0.03	0.04	0.06	0.13	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.03
50 x 10	0.04	0.05	0.06	0.07	0.13	0.27	0.39	0.00	0.00	0.01	0.01	0.00	0.00	0.03
50 x 20	0.07	0.08	0.14	0.15	0.25	0.53	0.82	0.00	0.00	0.01	0.02	0.00	0.00	0.06
100 x 5	0.16	0.26	0.30	0.35	0.52	1.03	1.56	0.00	0.01	0.01	0.02	0.01	0.03	0.31
100 x 10	0.36	0.79	0.45	0.94	1.07	2.20	3.35	0.00	0.01	0.02	0.03	0.02	0.02	0.40
100 x 20	0.81	1.64	0.83	1.91	2.56	5.05	7.58	0.01	0.02	0.04	0.06	0.04	0.02	0.68
200 x 10	3.17	14.54	4.39	15.01	10.21	19.59	28.92	0.02	0.04	0.08	0.13	0.18	0.19	7.25
200 x 20	7.66	32.59	8.43	23.84	22.74	45.59	68.98	0.03	0.07	0.15	0.23	0.33	0.15	8.57
500 x 20	120.86	1079.65	174.03	1090.81	378.95	390.46	391.29	0.19	0.44	0.95	1.62	7.32	2.88	215.44
Average	11.10	94.14	15.72	94.43	34.71	38.74	41.93	0.02	0.05	0.11	0.18	0.66	0.27	19.40
ARPT	0.85	2.04	1.14	2.26	2.82	5.57	8.12	0.02	0.05	0.13	0.20	0.05	0.05	1.02

Table 5: Computational times of heuristics II

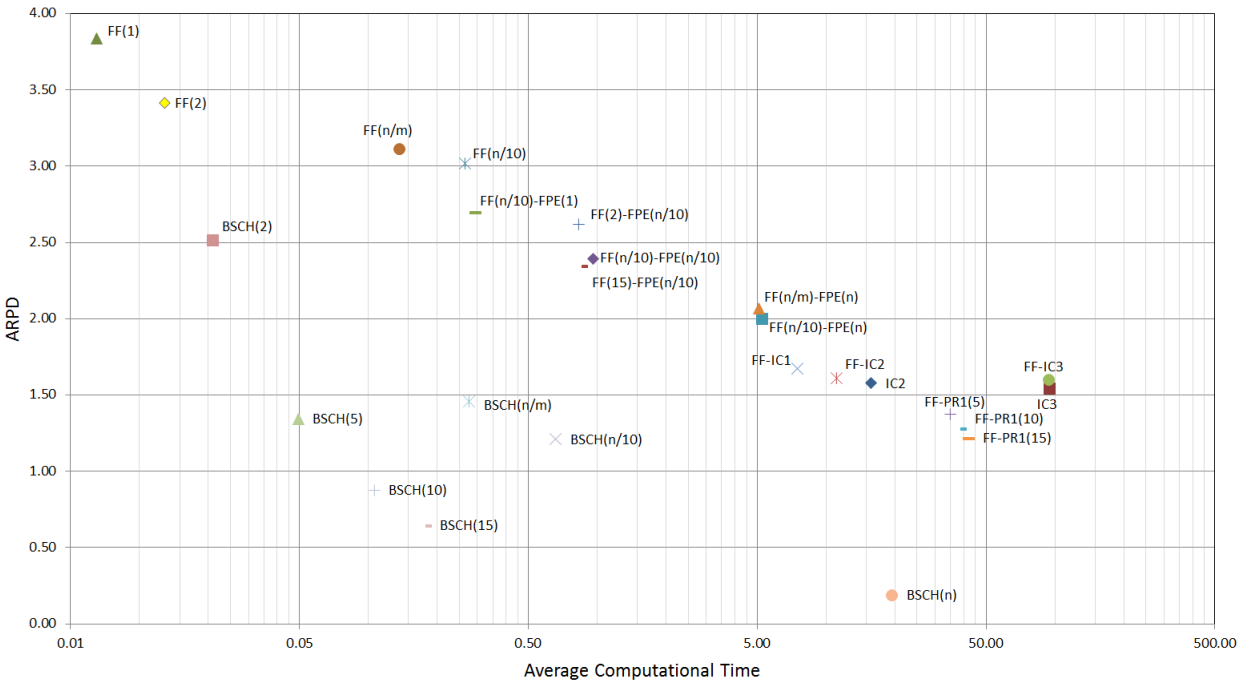

Figure 3: $ARPD1$ versus average CPU times. Average computational time (X-axis) is shown in logarithmic scale.

4.2 Comparison between BSCH and metaheuristics

An additional series of experiments have been conducted to compare the $BSCH$ heuristic with an iterated local search (denoted as $MRSILS$) and an iterated greedy algorithm (denoted as IG_{RIS}). These two are among the best metaheuristics for the problem (see Dong et al., 2013 and Pan et al., 2008). In order to analyse the impact of $BSCH$, we separately run both metaheuristics until the stopping criterion $60 \cdot n \cdot m/2$ milliseconds. For each instance, five runs are considered and the average flowtime values are recorded. Both metaheuristics have been again implemented under the same computer conditions and the comparison has been performed for all instances of the benchmark. Results in terms of $ARPD2$ and average CPU times are shown in Table 8. Note that the last column indicates the ratio between the CPU time needed by the metaheuristics and the $BSCH(n)$ heuristic for each size of instance. $ARPD2$ is the average $RPD2$ which is calculated by Equation (16):

$$RPD2_{ih} = \frac{C^{ih}_{sum} - UB}{UB} \cdot 100, \forall\, i = 1, \ldots, I, h = 1, \ldots, H \tag{16}$$

Heuristic	$ARPD1$	$ARPT$	Avg. Time
$FF(1)$	3.84	0.01	0.01
$FF(2)$	3.42	0.01	0.01
$FF(n/10)$	3.02	0.02	0.26
$FF(n/m)$	3.11	0.03	0.14
$FF(2) - FPE(n/10)$	2.62	0.06	0.83
$FF(15) - FPE(n/10)$	2.35	0.17	0.86
$FF(n/10) - FPE(1)$	2.70	0.04	0.29
$FF(n/10) - FPE(n/10)$	2.39	0.08	0.96
$FF(n/10) - FPE(n)$	2.00	0.34	5.25
$FF(n/m) - FPE(n)$	2.07	0.34	5.09
$FF - IC1$	1.68	0.58	7.51
$FF - IC2$	1.61	0.85	11.10
$FF - IC3$	1.60	2.04	94.14
$IC2$	1.58	1.14	15.72
$IC3$	1.54	2.26	94.43
$FF - PR1(5)$	1.37	2.82	34.71
$FF - PR1(10)$	1.28	5.57	38.74
$FF - PR1(15)$	1.22	8.12	41.93
$BSCH(2)$	2.51	0.02	0.02
$BSCH(5)$	1.35	0.05	0.05
$BSCH(10)$	0.88	0.13	0.11
$BSCH(15)$	0.64	0.20	0.18
$BSCH(n/10)$	1.21	0.05	0.66
$BSCH(n/m)$	1.46	0.05	0.27
$BSCH(n)$	0.19	1.02	19.40

Table 6: Summary of results of the heuristics.

i	H_i	p-value	Mann-Whitney	$\alpha/(k-i+1)$	Holm's Procedure
1	$BSCH(2)$=$FF(n/m)$	0.000	R	0.0031	R
2	$BSCH(n/10)$=$FF(2) - FPE(n/10)$	0.000	R	0.0033	R
3	$BSCH(n/10)$=$FF(n/10) - FPE(n/10)$	0.000	R	0.0036	R
4	$BSCH(10)$=$FF(15) - FPE(n/10)$	0.000	R	0.0038	R
5	$BSCH(15)$= $FF(n/10) - FPE(n)$	0.000	R	0.0042	R
6	$BSCH(15)$=$FF(n/m) - FPE(n)$	0.000	R	0.0045	R
7	$BSCH(15)$=$FF - IC1$	0.000	R	0.0050	R
8	$BSCH(15)$=$FF - IC2$	0.000	R	0.0056	R
9	$BSCH(n)$=$IC2$	0.000	R	0.0063	R
10	$BSCH(n)$=$FF - IC3$	0.000	R	0.0071	R
11	$BSCH(n)$=$IC3$	0.000	R	0.0083	R
12	$BSCH(n)$=$FF - PR1(5)$	0.000	R	0.0100	R
13	$BSCH(n)$=$FF - PR1(10)$	0.000	R	0.0125	R
14	$BSCH(n)$=$FF - PR1(15)$	0.000	R	0.0167	R
15	$BSCH(2)$=$FF(n/10)$	0.001	R	0.0250	R
16	$BSCH(2)$=$FF(n/10) - FPE(1)$	0.163		0.0500	

Table 7: Holm's procedure.

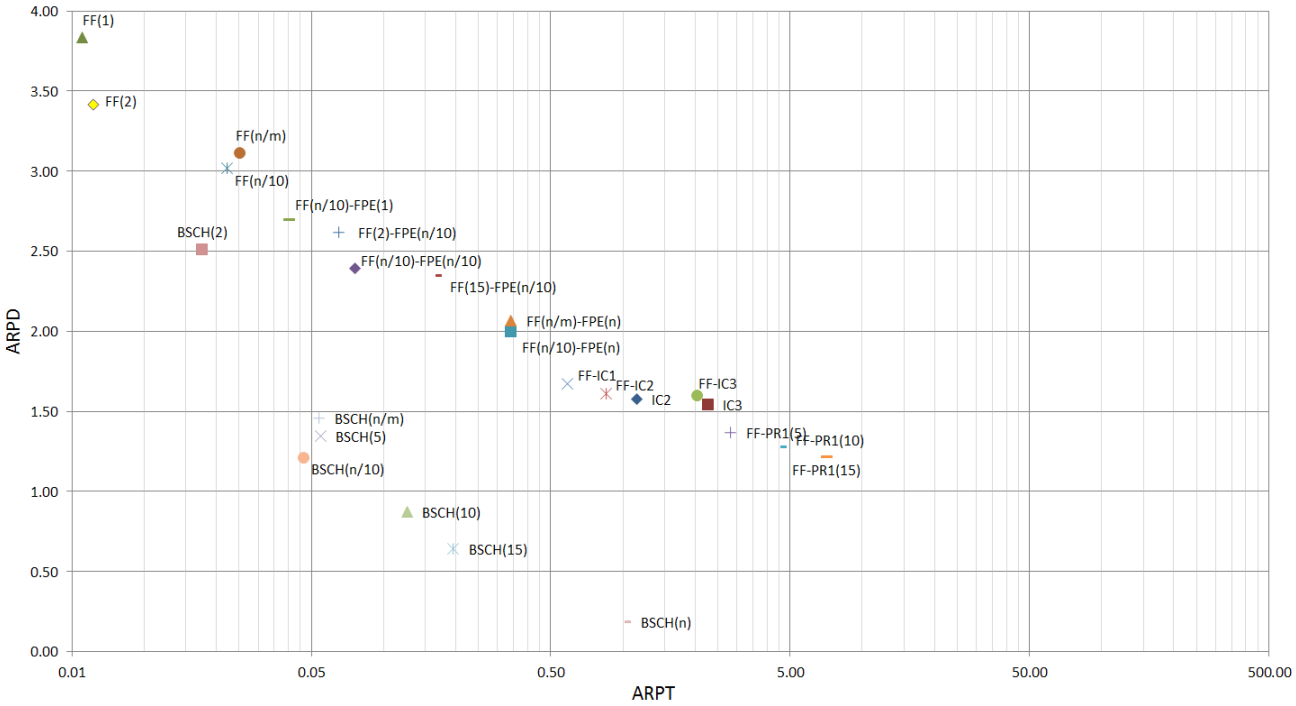

Figure 4: $ARPD1$ versus $ARPT$. $ARPT$ (X-axis) is shown in logarithmic scale

where UB is the best known upper bound for instance i taken from Pan and Ruiz (2012).

As it can be seen, both $ARPD2$ values and average CPU times of the metaheuristics are clearly improved by the proposed constructive heuristic. One the one hand, the best $ARPD2$ value of the metaheuristics is 0.76 while the $ARPD2$ value of the $BSCH(n)$ heuristic is 0.40 (there are statistical differences between the algorithms when a non-parametric Mann-Whitney test is used as p-value equals to 0.004). Additionally, 35 new best upper bounds have been found in the instances (see Table 9). This fact clearly highlights the excellent performance of the proposed heuristic since e.g. only 12 upper bounds were updated when Pan and Ruiz (2012) ran the several metaheuristics until a stopping criterion of $400 \cdot m \cdot n$ milliseconds (i.e. an average CPU time of 731.7 seconds). On the other hand, big differences are found when analysing the average CPU time between the algorithms, which are 19.4 seconds for the $BSCH(n)$ heuristic and 54.88 seconds for the metaheuristics. Although the differences in average CPU time are not so relevant, it is due to the use of an instance-size dependent indicator to compare algorithms with different stopping criteria (see Fernandez-Viagas and Framinan, 2015b for a more detailed explanation). In fact, regarding the ratio of the CPU time between the metaheuristics and the

	$ARPD2$				Avg. time		
Instance	$MRSILS$	IG_{RIS}	$BSCH(n)$	$MRSILS(BSCH)$	$MRSILS, IG_{RIS}$	$BSCH(n)$	$\frac{MRSILS, IG_{RIS}}{BSCH(n)}$
20 x 5	0.01	0.05	1.25	0.01	3.00	0.00	1704.55
20 x 10	0.00	0.08	0.75	0.00	6.00	0.00	2500.00
20 x 20	0.00	0.01	0.75	0.00	12.00	0.00	3508.77
50 x 5	0.57	0.69	0.75	0.28	7.50	0.03	291.60
50 x 10	0.70	0.90	1.04	0.47	15.00	0.03	438.34
50 x 20	0.69	0.99	1.48	0.63	30.00	0.06	529.10
100 x 5	1.11	1.17	0.30	0.22	15.00	0.31	48.49
100 x 10	1.44	1.60	0.57	0.27	30.00	0.40	74.63
100 x 20	1.50	1.89	1.14	0.83	60.00	0.68	87.60
200 x 10	1.10	1.35	-0.61	-0.71	60.00	7.25	8.28
200 x 20	1.24	1.46	-0.76	-0.83	120.00	8.57	14.01
500 x 20	0.79	0.85	-1.87	-1.90	300.00	215.44	1.39
Average	0.76	0.92	0.40	-0.06	54.88	19.40	767.23

Table 8: $ARPD2$ and average CPU time, for each instance size, required by the $BSCH(n)$ heuristic and the metaheuristics $MRSILS$ and IG_{RIS}.

proposed heuristic, the computational effort for the metaheuristics is 767.23 times bigger than for the proposed heuristic. This also serves to explain the good performance of the metaheuristics in the 60 smallest instances as compared with the proposed constructive heuristic since a huge computational effort is used for the former (e.g. approximately 3,500 times higher in instances Ta21-Ta-30). In contrast, the CPU time of the proposed heuristic is always less than 1 second, and its average CPU times for the first 90 instances is 0.17 seconds against 19.83 seconds required by the metaheuristics.

Finally, the excellent behavior of the proposed heuristic is also confirmed in a last experiment. We measure the variation in the quality of the solution in the metaheuristic $MRSILS$ when the $BSCH(n)$ heuristic is used as the initial sequence of the metaheuristic, denoted as $MRSILS(BSCH)$. Results are shown in the fifth column of Table 3. The $ARPD2$ found by $MRSILS(BSCH)$ is –0.06 as compared to 0.76 ($ARPD$ found of by the original $MRSILS$).

5 Conclusions

In this paper, we have presented $BSCH(x)$, a beam-search-based constructive heuristic to solve the PFSP to minimise total flowtime. The algorithm constructs sequences, and at the same time, it combines them and selects the best x ones. Since the nodes are formed by partial sequences, a forecast index is introduced in order to be able to compare nodes with different un- and scheduled

Instance	Best Bound	Instance	Best Bound	Instance	Best Bound	Instance	Best Bound
TA1	14033	TA31	64802	TA61	**253232**	TA91	**1042494**
TA2	15151	TA32	68051	TA62	**242093**	TA92	**1028957**
TA3	13301	TA33	63162	TA63	237832	TA93	**1043467**
TA4	15447	TA34	68226	TA64	227738	TA94	**1029244**
TA5	13529	TA35	69351	TA65	240301	TA95	**1029384**
TA6	13123	TA36	66841	TA66	232342	TA96	**999241**
TA7	13548	TA37	66253	TA67	240366	TA97	**1042663**
TA8	13948	TA38	64332	TA68	230945	TA98	**1035981**
TA9	14295	TA39	62981	TA69	**247677**	TA99	**1015389**
TA10	12943	TA40	68770	TA70	242933	TA100	**1022277**
TA11	20911	TA41	87114	TA71	298385	TA101	**1223860**
TA12	22440	TA42	82820	TA72	**273826**	TA102	**1234081**
TA13	19833	TA43	79931	TA73	288114	TA103	**1259866**
TA14	18710	TA44	86446	TA74	301044	TA104	**1228060**
TA15	18641	TA45	86377	TA75	**284279**	TA105	**1219886**
TA16	19245	TA46	86587	TA76	269686	TA106	**1219432**
TA17	18363	TA47	88750	TA77	279463	TA107	**1234366**
TA18	20241	TA48	86727	TA78	290908	TA108	**1240627**
TA19	20330	TA49	85441	TA79	301970	TA109	**1220873**
TA20	21320	TA50	87998	TA80	291283	TA110	**1235462**
TA21	33623	TA51	125831	TA81	365463	TA111	**6558547**
TA22	31587	TA52	119247	TA82	372449	TA112	**6679507**
TA23	33920	TA53	116459	TA83	370027	TA113	**6624893**
TA24	31661	TA54	120261	TA84	372393	TA114	**6649855**
TA25	34557	TA55	118184	TA85	368915	TA115	**6590021**
TA26	32564	TA56	120586	TA86	370908	TA116	**6603691**
TA27	32922	TA57	122880	TA87	373408	TA117	**6576201**
TA28	32412	TA58	122489	TA88	384525	TA118	**6629393**
TA29	33600	TA59	121872	TA89	374423	TA119	**6589205**
TA30	32262	TA60	123954	TA90	379296	TA120	**6626342**

Table 9: New best bounds (in bold) found by the proposed algorithm.

jobs.

Under the same computer conditions, the proposed heuristic improves each other efficient heuristic for the problem both in quality of the solutions and in computational effort (e.g. the $ARPD1$ and $ARPT$ of the $BSCH(n)$ heuristic is 0.19 and 0.02 respectively which are much less than those obtained by the most efficient heuristic so far, $FF - PR1(15)$ with 1.22 and 7.13). When comparing $BSCH(x)$ with the so-far most efficient heuristics in the literature, there are statistical differences for each new efficient heuristic with the only exception of $BSCH(2)$. Thereby, the set of efficient heuristics for the problem has been reduced from 17 heuristics to seven heuristics of only two types of heuristics, the existing FF for parameters 1 and 2 which is efficient for the smallest CPU times, and our proposal with $x \in \{2, n/10, 10, 15, n\}$.

The excellent performance of the proposed heuristic is also shown by means of its comparison against two of the best metaheuristics for the problem. Our proposal statistically outperforms both metaheuristics (i.e. the $ARPD2$ of $BSCH(n)$ is 0.40 against 0.76 of the best metaheuristic) using much less computational effort for each instance of the benchmark. Additionally, the proposed heuristic found new best upper bounds for 35 of the 120 instances in Taillard's benchmark.

Acknowledgements

This research has been funded by the Spanish Ministry of Science and Innovation, under projects "ADDRESS" with reference DPI2013-44461-P and "PROMISE" with reference DPI2016-80750-P.

References

Abedinnia, H., Glock, C., and Brill, A. (2016). New simple constructive heuristic algorithms for minimizing total flow-time in the permutation flowshop scheduling problem. *Computers and Operations Research*, 74:165–174.

Allahverdi, A. and Aldowaisan, T. (2002). New heuristics to minimize total completion time in m-machine flowshops. *International Journal of Production Economics*, 77(1):71–83.

Armentano, V. and Ronconi, D. (1999). Tabu search for total tardiness minimization in flowshop scheduling problems. *Computers and Operations Research*, 26(3):219–235.

Della Croce, F. and T'kindt, V. (2002). A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem. *Journal of the Operational Research Society*, 53(11):1275–1280.

Dong, X., Chen, P., Huang, H., and Nowak, M. (2013). A multi-restart iterated local search algorithm for the permutation flow shop problem minimizing total flow time. *Computers and Operations Research*, 40(2):627–632.

Dong, X., Huang, H., and Chen, P. (2008). An improved NEH-based heuristic for the permutation flowshop problem. *Computers and Operations Research*, 35(12):3962–3968.

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

Fernandez-Viagas, V. and Framinan, J. M. (2015b). A new set of high-performing heuristics to minimise flowtime in permutation flowshops. *Computers and Operations Research*, 53(0):68 – 80.

Framinan, J., Gupta, J., and Leisten, R. (2004). A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55(12):1243–1255.

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.

Framinan, J., Leisten, R., and Ruiz-Usano, R. (2002). Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation. *European Journal of Operational Research*, 141(3):559–569.

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

Garey, M., Johnson, D., and Sethi, R. (1976). Complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129.

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

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1:61–68.

Laha, D. and Sarin, S. (2009). A heuristic to minimize total flow time in permutation flow shop. *Omega*, 37(3):734–739.

Li, X., Wang, Q., and Wu, C. (2009). Efficient composite heuristics for total flowtime minimization in permutation flow shops. *Omega*, 37(1):155–164.

Liu, J. and Reeves, C. (2001). Constructive and composite heuristic solutions to the $P||\sum c_i$ scheduling problem. *European Journal of Operational Research*, 132:439–452.

Lowerre, B. T. (1976). *The HARPY speecch recognition system*. PhD thesis, Carnegie-Mellon University, USA.

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.

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

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.

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

Rajendran, C. and Ziegler, H. (1997). An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. *European Journal of Operational Research*, 103:129–138.

Reza Hejazi, S. and Saghafian, S. (2005). Flowshop-scheduling problems with makespan criterion: A review. *International Journal of Production Research*, 43(14):2895–2929.

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.

Sabuncuoglu, I. and Bayiz, M. (1999). Job shop scheduling with beam search. *European Journal of Operational Research*, 118(2):390–412.

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

Valente, J. (2010). Beam search heuristics for quadratic earliness and tardiness scheduling. *Journal of the Operational Research Society*, 61(4):620–631.

Valente, J. and Alves, R. (2005). Filtered and recovering beam search algorithms for the early/tardy scheduling problem with no idle time. *Computers and Industrial Engineering*, 48(2):363–375.

Valente, J. and Alves, R. (2008). Beam search algorithms for the single machine total weighted tardiness scheduling problem with sequence-dependent setups. *Computers and Operations Research*, 35(7):2388–2405.

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.