A computational evaluation of constructive and improvement heuristics for the blocking flow shop to minimise total flowtime*

Victor Fernandez-Viagas[1]†, Rainer Leisten[2], 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

[2] Industrial Engineering, Faculty of Engineering, University of Duisburg-Essen

Bismarckstr. 90, 47057 Duisburg, Germany, rainer.leisten@uni-due.de

May 31, 2016

Abstract

This paper focuses on the blocking flow shop scheduling problem with the objective of total flowtime minimisation. This problem assumes that there are no buffers between machines and, due to its application to many manufacturing sectors, it is receiving a growing attention by researchers during the last years. Since the problem is NP-hard, a large number of heuristics have been proposed to provide good solutions with reasonable computational times. In this paper, we conduct a comprehensive evaluation of the available heuristics for the problem and for related problems, resulting in the implementation and testing of a total of 35 heuristics. Furthermore, we propose an efficient constructive heuristic which successfully combines a pool of partial sequences in parallel, using a beam-search-based approach. The computational experiments show the excellent performance of the proposed heuristic as compared to the best-so-far algorithms for the problem, both in terms of quality of the solutions and of computational requirements. In fact, despite being a relative fast constructive heuristic, new best upper bounds have been found for more than 27% of Taillard's instances.

Keywords: Scheduling, Flowshop, Blocking, Heuristics, PFSP, Total Flowtime, Computational Evaluation, Beam Search

*Preprint submitted to Expert Systems With Applications. http://dx.doi.org/10.1016/j.eswa.2016.05.040
†Corresponding author.

1 Introduction

The flowshop is a layout employed in many manufacturing scenarios (see e.g. Krajewski et al., 1987 and Storer et al., 1992) where n jobs must follow the same processing route on m machines. The so-called flowshop scheduling problem consists in finding the best sequence of jobs on each machine according to some objective(s). In its classical formulation, unlimited buffers capacity between two consecutive machines is considered (see the full list of assumptions in Section 3). However, zero-buffer flowshops are very common in several industrial sectors, such as iron and steel industry, chemical and pharmaceutical industries, just-in-time production lines and in-line robotic cells (see e.g. Reklaitis, 1982, Sethi et al., 1992, Hall and Sriskandarajah, 1996 and Gong et al., 2010). This problem is usually denoted as blocking flowshop scheduling problem (BFSP) since a job blocks a machine until the next machine is available. Therefore, interest in this problem is increasing over the past years (Ribas and Companys, 2015), although there are not many algorithms as compared to the number of heuristics and metaheuristics for the traditional permutation flowshop scheduling problem –denoted as PFSP– (see e.g. Ruiz and Maroto, 2005, Pan and Ruiz, 2013, Ferrer et al., 2016, and Fernandez-Viagas and Framinan, 2015c), which is one of the most studied problems in Operations Research.

Commonly used objectives for the BFSP include makespan and total flowtime (see e.g. Ribas et al., 2011 and Ronconi, 2004). Among them, the total flowtime stands out as the most relevant and meaningful for today's dynamic production environment (Liu and Reeves, 2001). This problem is denoted as $Fm|block|\sum C_j$ according to the notation by Graham et al. (1979). Note that, as there are zero-capacity buffers between two consecutive machines, several jobs cannot wait at the same time before the machine and the job sequence must therefore be the same on every machine. As a conclusion, $n!$ schedules have to be considered, i.e. the number of solutions is the permutation of n jobs.

In this paper, we propose an efficient constructive heuristic for the BFSP with flowtime objective based on beam search that outperforms existing heuristics for the problem. Additionally, we test several adaptations of the most efficient algorithms for the $Fm|block|C_{max}$ problem as well as for the PFSP to minimise makespan and total flowtime (respectively denoted as $Fm|prmu|C_{max}$

and $Fm|prmu|\sum C_j$ according to Graham et al., 1979). We include them in the comparison since many algorithms originally implemented for the PFSP –such as the the iterated greedy proposed by Ruiz and Stützle (2007) (see e.g. Pan et al., 2008b) or the NEH heuristic by Nawaz et al. (1983) (see e.g. Leisten, 1990)– have turned to be efficient for several decision problems. The resulting computational evaluation is composed of a total of 36 heuristics which are fully recoded and exhaustively compared under the same computer conditions. Additionally, we introduce a speed-up method to accelerate the insertion phases of all algorithms.

The rest of the paper is organised as follows. The state of the art is analysed in Section 2. In Section 3, the problem and the notation are described. The beam-search-based constructive heuristic is proposed in Section 4. In Section 5, a complete comparison of heuristics is performed. Finally, the conclusions are discussed in Section 6.

2 Literature Review

The classification of algorithms to solve operations research problems is ambiguous, as different classifications have been proposed in the literature (see e.g. Zäpfel et al., 2010, Zanakis et al., 1989). In this paper, we adopt the classification used e.g. by Framinan et al. (2005) and Ruiz and Maroto (2005), thus dividing the algorithms between ad hoc heuristics and general-purpose metaheuristics. We focus exclusively on heuristics, which can be classified into constructive heuristics and improvement heuristics (Ruiz and Maroto, 2005). Constructive heuristics obtain the final sequence by integrating jobs –usually in an iterative manner– on an incomplete sequence. Improvement heuristics are usually composed of two phases: a construction phase where a complete sequence is formed, and an improvement phase where the solution is improved by means of some method, typically by specific knowledge of the problem. Using this classification, heuristics (constructive and improvements) naturally stop after a predefined number of steps independent of a time limit, whereas metaheuristics require a termination criterion (usually related to the CPU time effort) as an input parameter. Furthermore, metaheuristics require a fast initial solution to start their procedures, and the quality of this initial solution is known to influence the performance of the metaheuristic, so usually the initial solution for the metaheuristics is provided by (constructive and/or improvement) heuristics. The aim of our paper is precisely to assess the state-of-the-art of the heuristics -in the aforementioned sense- for the problem, therefore we have not considered in the computational evaluation very interesting and well-performing procedures as the ones of Ribas et al. (2015) and Khorasanian and Moslehi (2012). Furthermore, note that it is problematic to include these procedures in the comparison since these procedures a) require the other heuristics to provide an initial solution, and b) allow different stopping criteria, whereas that of the heuristics is fixed.

In this section, a review of the literature on the problem under consideration is presented. Since there are several heuristics for related scheduling problems that can be adapted to our problem, we also review these contributions. More specifically, we review:

- Heuristics for the (classical) permutation flowshop scheduling problem to minimise makespan.
- Heuristics for the permutation flowshop scheduling problem to minimise total flowtime.
- Heuristics for the blocking flowshop scheduling problem, both with makespan and flowtime objectives.
- Speed-up procedures developed for related flowshop scheduling problems.

Regarding heuristics for the $Fm|prmu|C_{max}$ problem, we focus on the most promising ones and refer the reader to Framinan et al. (2004), Reza Hejazi and Saghafian (2005) or Ruiz and Maroto (2005) for more extensive reviews. Among the available heuristics, the NEH heuristic (Nawaz et al., 1983) is, without doubt, the most efficient heuristic for the problem. Its excellent performance –established by Ruiz and Maroto (2005)– probably lies in the low computational cost of carrying out its insertion phases due to the speed-up by Taillard (1990). Therefore, several papers have focused on improving some of the phases of the NEH, or on proposing NEH-based heuristics. More specifically, improvements in the initial order of the NEH have been proposed by e.g. Framinan et al. (2003) and Vasiljevic and Danilovic (2015). Regarding improvements in the insertion phase of the NEH, Rad et al. (2009) propose several heuristics (denoted as FRB1, FRB2, FRB3, FRB4_k and FRB5) where a partial insertion local search method after the insertion of a job is employed. In a similar way, Ying and Lin (2013) employ another partial local search based on the interchange of jobs. Several works address the problem of breaking the ties of the partial makespans when inserting a job, such as e.g. Ribas et al. (2010), Kalczynski and Kamburowski (2007), Kalczynski and Kamburowski (2008), Kalczynski and Kamburowski (2009), Dong et al. (2008) and Fernandez-Viagas and Framinan (2014).

Regarding the $Fm|prmu|\sum C_j$ problem, a recent evaluation carried out by Fernandez-Viagas and Framinan (2015c) shows that, in terms of average relative percentage deviation and average relative percentage computation time, the set of efficient heuristics is formed by the Raj heuristic by Rajendran (1993); the LR heuristic proposed by Liu and Reeves (2001); the RZ heuristic by Rajendran and Ziegler (1997); the RZ-LW proposed by Li and Wu (2005); the LR-NEH heuristic proposed by Pan and Ruiz (2013); the IC1, IC2 and IC3 improvement heuristics by Li et al. (2009); and finally, the PR1 heuristic proposed by Pan and Ruiz (2013). As most of these heuristics include the LR heuristic as initial or main procedure, Fernandez-Viagas and Framinan (2015c) propose an improvement of this method, denoted as FF, which heavily decreases the required CPU time. This procedure has been incorporated in each heuristic using the LR procedure, obtaining excellent results for the following heuristics: FF, FF-FPE (replacing LR by FF in the LR-FPE heuristic by Liu and Reeves, 2001), FF-ICi (ICi heuristics by Li et al., 2009 with FF instead of LR) and FF-PR1 (PR1 heuristic by Pan and Ruiz, 2013 using the FF procedure).

Regarding BFSP, several algorithms have been proposed for makespan minimisation. McCormick et al. (1989) implement a constructive heuristic, denoted as PF, to minimise cycle time, that constructs a sequence by progressively inserting an unsequenced job with minimal sum of idle and blocking time. Ronconi (2004) proposes three constructive heuristics (denoted as MM, MME and PFE) to solve the $Fm|block|C_{max}$. MME and PFE are variations of the original NEH heuristic where the initial order is replaced by the MM and PF heuristics respectively. In Ribas et al. (2011), several NEH-based heuristics are proposed using different mechanisms to break ties in the first and second phase of the NEH heuristic. The heuristics are compared with the MME and PFE heuristics. Pan and Wang (2012) propose eight heuristics (the wPF and PW constructive heuristics and the PF-NEH, wPF-NEH, PW-NEH, PF-NEH$_{\mathrm{LS}}$, wPF-NEH$_{\mathrm{LS}}$ and PW-NEH$_{\mathrm{LS}}$ improvement heuristics) based on NEH and LR. The heuristics clearly outperform MME and PFE in terms of quality of the solution and computational effort. In Ribas et al. (2013), these heuristics have been improved by evaluating the sequences before and after the insertion phase as well as using the reversibility property.

Regarding the minimisation of total flowtime in the BFSP, Wang et al. (2010) introduce an adaptation of the NEH algorithm using the non-decreasing sum of processing times as initial order. Note that this order outperforms the original one for the $Fm|prmu|\sum C_j$ problem. This heuristic is used as initial sequence for the metaheuristics proposed by Bao et al. (2012) and Deng et al. (2012). Han et al. (2012) and Han et al. (2013) adapt the MME heuristic to minimise total flowtime as well as propose two new NEH-based heuristics modifying the initial order (denoted as MME-A and MME-B). Finally, Ribas and Companys (2015) propose six new heuristics for the problem. Firstly, they adapt the PF heuristic (McCormick et al., 1989) to the problem and propose two new constructive heuristics denoted as HPF1 and HPF2 modifying the index to choose a job. Then, they propose three NEH-based heuristics (NPF, NHPF1 and NHPF2) using the previous heuristics as initial sequences of the NEH.

Finally, regarding speed up methods to accelerate algorithms, they have been successfully applied for several problems related to flowshop scheduling: the PFSP to minimise total flowtime (see e.g. Li et al., 2009); the PFSP to minimise total tardiness (see e.g. Vallada and Ruiz, 2010); the PFSP to minimise makespan subject to maximum tardiness (see e.g. Fernandez-Viagas and Framinan, 2015b); and the distributed PFSP to minimise makespan (see e.g. Naderi and Ruiz, 2010 and Fernandez-Viagas and Framinan, 2015a). However, to the best of our knowledge, they have not been applied to the BFSP so far.

3 Problem statement. Mathematical model

The problem under study can be stated as follows: a set $\mathcal{N}$ of n jobs have to be scheduled in a flow shop consisting on a set $\mathcal{M}$ of m machines without intermediate buffers. Each machine is always available and can process at most one job at the same time. Each job $j \in \mathcal{N}$ has a non preemptive processing time p_{ij} on each machine $i \in \mathcal{M}$. Set up times are sequence-independent and non-anticipatory (see Framinan et al., 2014) and thus can be included in the processing times of each job. Given a sequence $\Pi := (\pi_1, ...\pi_n)$, let $c_{ij}(\Pi)$ and $e_{ij}(\Pi)$ represent the departure and start time of job j from (on) machine i, respectively. Note that the departure time of a job must not necessarily be equal to its completion time, as the next machine can block this job after its completion (denoted as blocking time, see Figure 1). Similarly, $c_{i[r]}(\Pi)$ represents the departure time of job in position $r \in [1, n]$ of sequence Π from machine i, i.e. $c_{i[r]}(\Pi) = c_{i\pi_r}$. By means of these variables, the total flowtime, denoted as TF, can be defined by Expression (1).

$$TF = \sum_{r=1}^{n} c_{m[r]} \tag{1}$$

Where the departure times can be recursively obtained according to the following expressions (Ronconi, 2004):

$$c_{0[1]}(\Pi) = 0$$

$$c_{i[1]}(\Pi) = \sum_{q=1}^{i} p_{q\pi_1}, i = 1, ..., m-1$$

$$c_{0[r]}(\Pi) = c_{1,[r-1]}(\Pi), r = 2, ..., n$$

$$c_{i[r]}(\Pi) = \max\{c_{i-1,[r]}(\Pi) + p_{i\pi_r}, c_{i+1,[r-1]}(\Pi)\}, r = 2, ..., n, i = 1, ..., m-1$$

$$c_{m[r]}(\Pi) = c_{m-1,[r]}(\Pi) + p_{m\pi_r}, r = 1, ..., n$$

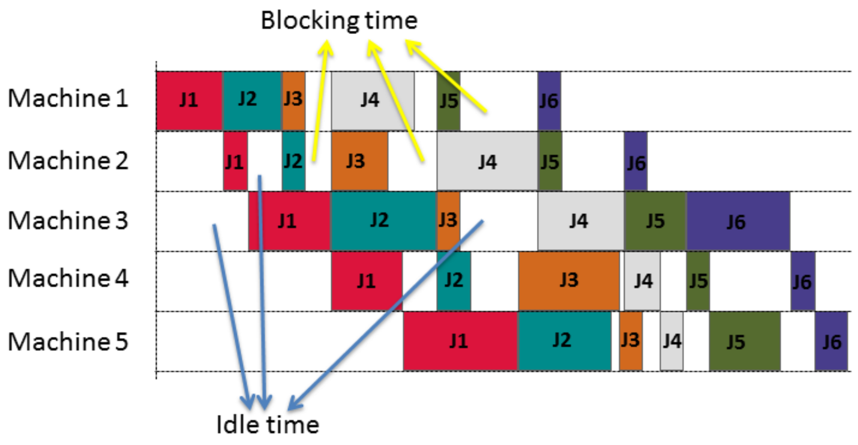

Figure 1: Example of the problem under study.

4 Proposed Heuristic

4.1 Introduction

In this section, a beam-search-based constructive heuristic, BS, is proposed to solve the $Fm|block|\sum C_j$ which successfully combines the diversification of population-based metaheuristics with the speed of constructive heuristics. The algorithm simultaneously constructs several partial sequences in each iteration (denoted as candidate nodes) by appending jobs one by one and keeping the best ones (denoted as selected nodes) over all candidates. A simple example of the algorithm with four jobs is shown in Figure 2. More specifically, the algorithm is composed of the following phases:

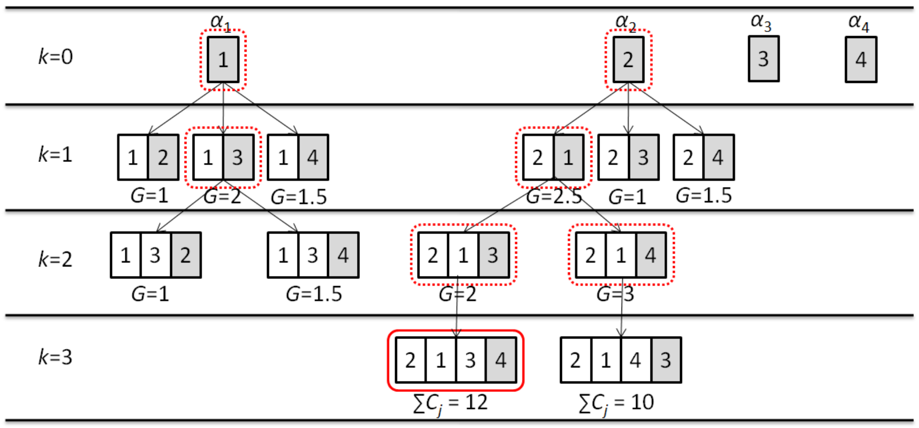

Figure 2: Example of the proposed algorithm.

- Obtain the initial selected nodes
- For n iterations:
    - Construct candidate nodes
    - Evaluate candidates nodes
    - Select the best candidates nodes (selected nodes)

These steps are detailed in the next subsections. However, we first must introduce in Section 4.2 some notation required.

4.2 Notation

The proposed heuristic appends the jobs at the end of a partial sequence. Note that, when a new job q is placed after the last position (position $k+1$) of a partial sequence $\Pi_k := (\pi_1, \ldots, \pi_k)$ of length k, the departure and the start times of job q can be computed according to Expressions (2) and (3), respectively. Obviously, $c_{i[k]}(\Pi)$ represents the departure time of the job placed in the last position of Π_k (i.e. before job q). Additionally, let $t_q(\Pi_k)$ and $b_q(\Pi_k)$ be the total idle and the total blocking time induced by job q, respectively, see Expressions (4) and (5).

$$c_{iq}(\Pi_k) = \begin{cases} c_{i[k]}(\Pi_k) + p_{iq}, & i = 1 \\ \max\{c_{i-1,q}(\Pi_k) + p_{iq}, c_{i+1,[k]}(\Pi_k)\}, & \forall\ i = \{2, \ldots, m-1\} \\ c_{i-1,q}(\Pi_k) + p_{iq}, & i = m \end{cases} \tag{2}$$

$$e_{iq}(\Pi_k) = \begin{cases} c_{i[k]}(\Pi_k), & i = 1 \\ c_{i-1,q}(\Pi_k), & \forall\ i = \{2, \ldots, m\} \end{cases} \tag{3}$$

$$t_q(\Pi_k) = \sum_{i=2}^{m} \max\{e_{i-1,q}(\Pi_k) + p_{i-1,q} - c_{i[k]}(\Pi_k), 0\} \tag{4}$$

$$b_q(\Pi_k) = \sum_{i=2}^{m} \max\{c_{i[k]}(\Pi_k) - (e_{i-1,q}(\Pi_k) + p_{i-1,q}), 0\} \tag{5}$$

Let us denote by x (beam width) the number of selected nodes in each iteration. At iteration k ($k = 1, \ldots, n$), selected node l ($l = 1, \ldots, x$) is composed of k sequenced jobs (partial sequence) denoted as $S_l^k := (s_{1l}^k, \ldots, s_{kl}^k)$, and a set of $n-k$ unsequenced jobs denoted as $\mathcal{U}_l^k := \{u_{1l}^k, \ldots, u_{n-k,l}^k\}$. Additionally, as the proposed algorithm is composed of a set of partial sequences in each iteration, let us extend the notation and denote by c_{iql}^k and e_{iql}^k the departure and start times of job q on machine i of selected node l in iteration k (i.e. $c_{iql}^k = c_{iq}(S_l^k)$ and $e_{iql}^k = e_{iq}(S_l^k)$), respectively. Analogously, t_{ql}^k and b_{ql}^k represent the total idle and total blocking times of job q on machine i of selected node l in iteration k (i.e. $t_{ql}^k = t_q(S_l^k)$ and $t_{ql}^k = t_q(S_l^k)$), respectively. Finally, let c_{ql}^k represent the departure time of job q from the last machine m, i.e. $c_{ql}^k := c_{mql}^k$.

4.3 Heuristic description

First, the algorithm sorts all jobs according to non-decreasing order of indicator ξ_j, see Expression (6). Let $\alpha := (\alpha_1, \ldots, \alpha_l, \ldots, \alpha_n)$ denote this order.

$$\xi_j := \frac{(n-2)}{4} \cdot w_j + \sum_{i=1}^{m} p_{ij}, \ \forall\, j \in [1, n] \tag{6}$$

where w_j is the weighted idle time defined by Expression (7) (see Liu and Reeves, 2001):

$$w_j := \sum_{i=2}^{m} \frac{m \cdot \sum_{i'=1}^{i-1} p_{i'j}}{i-1}, \ \forall\, j \in [1, n] \tag{7}$$

The nodes selected in the first iteration are constructed according to the indicator as follows: the partial sequence $S_l^1 = (s_{1l}^1)$ with $l \in \{1, \ldots, x\}$, is formed by the job in position l of the initial order, i.e. $s_{1l}^1 = \alpha_l$; the $\mathcal{U}_l^1$ set of unsequenced jobs contains all jobs with the exception of the job in S_l^1.

Once the initial selected nodes are obtained, in each iteration k, each selected node l forms $n-k$ candidate nodes for the next generation. Each candidate node $v \in \{1, \ldots, n-x\}$ is constructed from selected node l, appending each job in set $\mathcal{U}_l^k$ at the end of S_l^k. Let $\hat{S}_{vl}^k := (\hat{s}_{1vl}^k, \ldots, \hat{s}_{k+1,v,l}^k)$ and $\hat{\mathcal{U}}_{vl}^k$ be the corresponding partial sequence and set of unsequenced jobs, respectively. Then, the partial sequence of this candidate node and its set of unsequenced jobs are defined by Expression (8).

$$\begin{gathered} \hat{S}_{vl}^k = (\hat{s}_{1vl}^k, \ldots, \hat{s}_{k+1,v,l}^k) = (S_l^k, u_{vl}^k) = (s_{1l}^k, \ldots, s_{kl}^k, u_{vl}^k) \\ \hat{\mathcal{U}}_{vl}^k = \{\hat{u}_{1vl}^k, \ldots, \hat{u}_{k+1,v,l}^k\} = \mathcal{U}_l^k - u_{vl}^k \end{gathered} \tag{8}$$

Consequently, in iteration k, a total of $x \cdot (n-k)$ candidate nodes are formed. Among these candidate nodes, the best x ones are selected for the next iteration. Note that, as each new selected node l' in iteration $k+1$ (composed of partial sequence $S_{l'}^{k+1}$, $\forall l' \in \{1, \ldots, x\}$) is formed by adding job u_{vl}^k to selected node l (composed of partial sequence S_l^k), node l' selected in iteration $k+1$ does not have necessarily to come from the partial sequence $S_{l'}^k$ (i.e. l' may be different from l). Therefore, it may happen that one node l is selected in iteration k, but its partial sequence is not selected for the next iteration $(k+1)$. Let $branch[l']$ and $job[l']$ denote the selected node l and job u^k_{vl}, respectively, that forms selected node l' in iteration $k+1$. In order to select the candidate nodes for the next iteration $(k+1)$, three issues (typically different for each one) have to be considered to evaluate them which are:

- Influence of the chosen job, u^k_{vl}, i.e. the last job in the partial sequence $\hat{S}^k_{vl}$ $(\hat{s}^k_{k+1,v,l})$. Obviously, the departure time of this job on the last machine, $c^k_{u^k_{vl}l}$, has a direct influence on the final objective function. Additionally, the job may incur in idle and blocking times that may largely influence the departure times of the subsequent jobs to be inserted. This influence is higher at the beginning of the algorithm when the partial sequence is relative empty, and lower in the last iterations where the sequence is almost complete as it affects to a smaller number of jobs (in fact, it does not affect to any job in the last iteration). The index L^k_{vl} (see Expression (9)), which balances these three objectives, is used to measure the influence of inserting job u^k_{vl}.

$$\begin{gathered} L^k_{vl} = c^k_{u^k_{vl}l} + a \cdot \tfrac{n-k-2}{n} \cdot (t^k_{u^k_{vl}l} + b^k_{u^k_{vl}l}), \\ \forall k = \{1,\ldots,n-1\}, v = \{1,\ldots,n-k\}, l = \{1,\ldots,x\} \end{gathered} \tag{9}$$

  where a is a parameter to balance the influence of the departure time against that of blocking and idle time. $t^k_{u^k_{vl}l}$ and $b^k_{u^k_{vl}l}$ are the sum of idle and blocking times between position k (job $\hat{s}^k_{kvl}$) and $k+1$ (job $\hat{s}^k_{k+1,v,l} = u^k_{vl}$) over all machines, respectively (see Section 4.2). Note that $v^k_{u^k_{vl}l}$, $t^k_{u^k_{vl}l}$ and $b^k_{u^k_{vl}l}$ can be calculated by means of the start time, $e^k_{iu^k_{vl}l}$, of job u^k_{vl} (placed in the last position of the S^k_l sequence) on machine i and the departure time, $c^k_{i[k]l}$, of the previous job (i.e. the job in position k, $\hat{s}^k_{kvl}$ or equivalently s^k_{kl}) which was already computed in the previous iteration of the algorithm (this fact leads to a high reduction of computational effort since the calculation of the departure times of the complete sequence is avoided).

- Influence of sequenced (previous) jobs, i.e. S^k_l (or equivalently $\hat{s}^k_{rvl}$, $\forall r \leq k$). Due to the process employed to construct the candidate nodes, the first k sequenced jobs of candidate node v may be different to the first jobs of other candidate nodes (e.g. the first candidate node is formed by jobs 1 and 2, and the second candidate node is formed by jobs 3 and 4).

The comparison of these partial sequences is not trivial. Obviously, when the sequences are complete, the algorithm has to look for the minimisation of the total flowtime. However, in case of partial sequence composed of different jobs, several other aspects may have a greater influence. On the one hand, although the goal is the minimisation of total flowtime, a comparison of the partial sequences based only on this measure would obviously be influenced by the characteristics of the jobs of each partial sequence. It would prioritise jobs with low processing times regardless the idle or blocking times that the inserted job causes to the machines. On the other hand, the exclusive consideration of idle and/or blocking times would miss the relation with the objective of the scheduling problem: the minimisation of total flowtime. To cover both aspects, the proposed algorithm uses index F_l^k (see Expression 10) to measure the influence of the sequenced jobs of candidate node v in iteration k. Note that this index is identical for all candidate nodes coming from selected node l since it does not consider the contribution of the last job of the sequence ($\hat{s}_{k+1,v,l}^k$). Furthermore, the contribution of idle and blocking times decrease with the number of iterations, thus avoiding their high influence in the last iterations.

$$F_{l'}^k = \Delta c_{l'}^k + a \cdot (\Delta t_{l'}^k + \cdot \Delta b_{l'}^k),\ \forall\ k = \{2, \ldots, n-1\},\ \ l^{'} = \{1, \ldots, x\} \qquad (10)$$

where Δt, Δb and Δc are the accumulated idle, blocking and departure time, respectively, defined by the following expressions:

$$\Delta t_{l'}^{k+1} = \Delta t_{branch[l']}^k + t_{job[l'],branch[l']}^k \cdot \frac{n-k-2}{n},\ \forall\ k = \{1, \ldots, n-2\},\ l^{'} = \{1, \ldots, x\} \quad (11)$$

$$\Delta b_{l'}^{k+1} = \Delta b_{branch[l']}^k + b_{job[l'],branch[l']}^k \cdot \frac{n-k-2}{n},\ \forall\ k = \{1, \ldots, n-2\},\ l^{'} = \{1, \ldots, x\} \quad (12)$$

$$\Delta c_{l'}^{k+1} = \Delta c_{branch[l']}^k + c_{job[l'],branch[l']}^k + c_{\lambda,branch[l']}^k,\ \forall\ k = \{1, \ldots, n-2\},\ l^{'} = \{1, \ldots, x\} \quad (13)$$

where $\Delta t_{l'}^1 = \Delta c_{l'}^1 = F_{l'}^1 = 0$, $\forall\ l^{'} = \{1, \ldots, x\}$. $c_{\lambda l}^k$ is the departure time of an artificial job placed at the end of the sequence as an estimation of the unscheduled jobs (see the following item).

- Influence of the unsequenced jobs. These are the next jobs to be sequenced in the selected nodes and hence, they also influence the evaluation of the candidate node. However, their impact on the final total flowtime is diffused since they have not been scheduled yet. As a measure of its influence, we use an artificial departure time denoted as $c^k_{\lambda l}$, which is the departure time of an artificial job λ placed in the last position (position $k+2$) of the sequence (after the last job, u^k_{vl} or $\hat{s}^k_{k+1,v,l}$). The processing times of this job are equal to the average processing times of all unscheduled jobs of selected node l (i.e. $\mathcal{U}^k_l$). Note that the chosen job u^k_{vl} is also considered to have an artificial departure time. The main reason is that the calculation of this term can be then globally done for all candidate nodes of selected node l, thus decreasing the complexity of the procedure, which is one of the main advantages of the proposed algorithm (see Fernandez-Viagas and Framinan, 2015c for a more detailed explanation).

Therefore, each candidate node v is evaluated using index G^k_{vl}, (see Expression 14) where the nodes selected for the next iteration are those with the best x values of the index. The pseudo code of the algorithm is shown in Figure 1 (see e.g Lin et al., 2012 for similar description of algorithms). The complexity of the algorithm is bounded by the creation and selection of the candidate nodes, which have a complexity of $O(x \cdot n^2 \cdot m)$ and $O(x^2 \cdot n^2)$ respectively. In this manner, the global complexity of the algorithm is $O(\max\{x \cdot n^2 \cdot m, x^2 \cdot n^2\})$.

$$G^k_{vl} = F^k_l + L^k_{vl}, \forall k = \{1, \ldots, n-2\}, c = \{1, \ldots, n-k\}, l = \{1, \ldots, x\} \tag{14}$$

4.4 Speed Up Procedure

In this section, we introduce a simple speed up procedure to accelerate the insertion phases of the algorithms. This procedure is based on the speed up methods proposed by Li et al. (2009) and Vallada and Ruiz (2010). Basically, the proposed procedure stores the departure times, c_{ij}, of each job j on each machine i before inserting a job in each position. Then, when the job is tested in each position j_1, all completion times c_{ij} with $j < j_1$ stay the same and are not

Algorithm 1: Beam-search-based constructive heuristic

```
    Procedure BS(x)
STEP 1:    //Initial Order
           Determination of $w_j$ and $\xi_j$, $\forall j \in [1, n]$;
           $\alpha :=$ Jobs ordered according to non-decreasing $\xi_j$ breaking ties in favor of jobs
           with lower $w_j$;
           Update $S^1_l$ $(s^1_{1l} = \alpha_l)$ $\forall l$ and $\mathcal{U}^1_l$ with the remaining jobs;
           $\Delta t^1_l, \Delta c^1_l, F^1_l = 0$, $\forall l \in [1, x]$;
           for $k = 1$ to $n - 2$ do
STEP 2:        //Candidate Nodes Creation
               Determination of $t^k_{u^k_{vl}l}$, $b^k_{u^k_{vl}l}$, $c^k_{iu^k_{vl}l}$, $c^k_{u^k_{vl}l}$, $\forall v \in [1, n-k], l \in [1, x]$;
STEP 3:        //Candidate Nodes Evaluation
               $G^k_{vl} := F^k_l + c^k_{u^k_{vl}l} + a \cdot (t^k_{u^k_{vl}l} + b^k_{u^k_{vl}l})$, $\forall v \in [1, n-k], l \in [1, x]$;
STEP 4:        //Candidate Nodes Selection
               Determination of the $l'$-th best candidate node according to non-decreasing
               $G^k_{vl}$ in iteration $k$. Denote by $branch[l']$ the value of the index $l$ of that
               candidate node and by $job[l']$ the value of $u^k_{vl}$, $\forall l' \in [1, x]$;
               $c^{k+1}_{i,[k+1],l'} \longleftarrow c^k_{i,job[l'],branch[l']}$;
STEP 5:        //Forecasting Phase. Update of the Forecast Index
               for $l' = 1$ to $x$ do
                   Update $S^{k+1}_{l'}$ and $\mathcal{U}^{k+1}_{l'}$ by removing job $job[l']$ from $\mathcal{U}^{k+1}_{l'}$ and including
                   in $S^k_{l'}$;
                   Determination of $c^{k+1}_{\lambda,branch[l']}$ 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 $(\mathcal{U}^{k+1}_{l'})$;
                   $\Delta t^{k+1}_{l'} = \Delta t^k_{branch[l']} + t^k_{job[l'],branch[l']} \cdot \frac{n-k-2}{n}$;
                   $\Delta b^{k+1}_{l'} = \Delta b^k_{branch[l']} + b^k_{job[l'],branch[l']} \cdot \frac{n-k-2}{n}$;
                   $\Delta c^{k+1}_{l'} = \Delta c^k_{branch[l']} + c^k_{job[l'],branch[l']} + c^k_{\lambda,branch[l']}$;
                   $F^{k+1}_{l'} = \Delta c^{k+1}_{l'} + a \cdot (\Delta t^{k+1}_{l'} + \Delta b^{k+1}_{l'})$;
               end
           end
STEP 6:    //Final evaluation
           Evaluate the flowtime of the $x$ selected nodes and return the best one;
    end
```

computed again. Although the complexity of the insertion phase remains the same using this procedure, a strong CPU reduction of about 30%-50% has been achieved for similar procedures in the literature (see e.g. Li et al., 2009). Note that the procedures proposed by e.g. Taillard (1990) and Naderi and Ruiz (2010) cannot be adapted since they are based only on the calculation of the makespan and cannot be applied for the calculation of each departure time on last machine. The proposed speed up procedure has been incorporated in each insertion phase of all implemented heuristics.

5 Computational Experiments

In this section, a computational evaluation of heuristics is carried out. To perform the comparison we adopt the following procedure: in Section 5.1, the set of instances used are presented. The design of experiments is carried out in Section 5.2. In Section 5.3, the implemented heuristics are enumerated. The indicators to evaluate the heuristics are defined in Section 5.4. Finally, the computational results of heuristics are shown in Section 5.5.

5.1 Benchmarks

As explained above, two computational experiments are generated in this paper: the experimental parameter tuning of Section 5.2 and the computational evaluation of heuristics of Section 5.5. In order to avoid an overfitting of parameter a of the proposed heuristic if the same benchmark would be used for both cases, the experiments are performed on the following two different set of instances:

- Benchmark $\mathcal{B}_1$, used for the calibration of the proposed heuristics. It is composed of 340 instances generated according to the procedure described by Ruiz and Stützle (2007). This benchmark consists of 68 combinations of the parameters $n = \{20, 50, 80, ..., 410, 440, 470, 500\}$ and $m = \{5, 10, 15, 20\}$. Processing times are uniformly distributed between 1 and 99, and 5 instances are generated for each combination of n and m.

- Benchmark $\mathcal{B}_2$, used for comparison among the implemented heuristics. This benchmark is composed of the set of 120 instances of Taillard (1993), which is the most common benchmark for the studied problems (see e.g. Wang et al., 2010, Han et al., 2011, Han et al., 2012, Han et al., 2013, Ribas et al., 2015, Ribas and Companys, 2015). The benchmark consists of a set of 12 instance sizes for $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$, where the processing times are generated with an uniform distribution [1,99]. For each instance size, 10 instances are constructed.

5.2 Experimental Parameter Tuning

In this section, we perform an experimental tuning of parameter a in the proposed heuristic on set $\mathcal{B}_1$. Regarding the values for the parameter x, we consider $x \in \{2, 5, 15, n/10, n\}$ (see e.g. Liu and Reeves, 2001; Fernandez-Viagas and Framinan, 2015c for similar values of the parameters in other constructive heuristics working with a pool of partial sequences), since this parameter is directly proportional to the CPU time and complexity of the algorithm. The computational experiments for the parameter a are carried out for the proposed BS($x = 5$) and the same value is used for each other value of x. We use the following values for parameter $a \in \{1, 2, 3, ..., 23, 24, 25\}$.

The relationship between the levels of the parameters is evaluated by means of a non-parametric Kruskal-Wallis test since normality and homoscedasticity assumptions are not fulfilled. Note that the Relative Percentage Deviation $RPD1$ –Expression (15)– is used to measure the quality of the solution of the heuristic for each instance.

$$RPD1 = \frac{O - Base}{Base} \cdot 100 \tag{15}$$

As a result of the experiments, it turns out that there are statistically significant differences between the levels of the three parameters, since the p-values obtained for the parameters n, m and a are 0.000. The best value found for parameter a is 14, which is used in Section 5.5 in BS(x) $\forall x \in \{2, 5, 15, n/10, n\}$.

5.3 Implemented Heuristics

In this section, the heuristics included in the computational evaluation are listed. According to the literature review in Section 2, 11 heuristics have been published so far for this problem. Additionally, we adapt 8 and 18 heuristics for the $Fm|block|C_{max}$ and for the classical PFSP problem, respectively, given their excellent performance. Finally, the proposed beam-search-based constructive heuristic is added to the comparison. In summary, the heuristics implemented are:

- Heuristics of the $Fm|block|\sum C_j$ problem:
  - Heuristic NEH_WPT: Wang et al. (2010).
  - Heuristic MME: Han et al. (2011) (adapted from Ronconi, 2004 for $Fm|block|C_{max}$).
  - Heuristic MME-A: Han et al. (2013).
  - Heuristic MME-B: Han et al. (2013) (adapted from Han et al., 2012 for $Fm|block|C_{max}$).
  - Heuristic NEH-MK: Moslehi and Khorasanian (2013).
  - Heuristic PF: Ribas and Companys (2015) (adapted from Ronconi, 2004 for $Fm|block|C_{max}$).
  - Heuristics HPF1 and HPF2: Ribas and Companys (2015).
  - Heuristics NPF, NHPF1 and NHPF2: Ribas and Companys (2015).
  - Heuristics BS(x), $\forall x \in \{2, 5, 15, n/10, n\}$: Proposed heuristic.
- Heuristics adapted from the $Fm|block|C_{max}$ problem:
  - Heuristics wPF and PW: Pan and Wang (2012). These heuristics are implemented as the original ones. For the final sequence, the total flowtime is calculated.
  - Heuristics PF-NEH(x), wPF-NEH(x) and PW-NEH(x), $\forall x \in \{1, 2, 5\}$: Pan and Wang (2012). In the NEH-based phase of the algorithms, each evaluation of makespan is replaced by the evaluation of total flowtime. Note that these heuristics include the evaluation of the objective function before applying the NEH-based phase (proposed by Ribas et al., 2013). The other improvement proposed by Ribas et al. (2013) (reversibility property) cannot be applied for total flowtime minimisation.

  - Heuristics PF-NEH$_{\text{LS}}(x)$, wPF-NEH$_{\text{LS}}(x)$ and PW-NEH$_{\text{LS}}(x)$, $\forall x \in \{1, 2, 5\}$: Pan and Wang (2012). In both the NEH-based and the local search phases of the algorithms, each evaluation of makespan is replaced by the evaluation of total flowtime.

- Heuristics adapted from the traditional PFSP to minimize total flowtime ($Fm|prmu|\sum C_i$). To adapt the heuristics, each evaluation of the total flowtime of a partial sequence is replaced by the evaluation of the total flowtime with blocking. Note that the indexes of initial sequences and FF and LR-based heuristics are not changed since the objective is the same.

  - Heuristic LR(1): Liu and Reeves (2001).
  - Heuristic FF(x), $\forall x \in \{1, 2, n/10, n/m\}$: Fernandez-Viagas and Framinan (2015c).
  - Heuristic FF-FPE(x, y), $\forall (x, y) \in \{(2, n/10), (15, n/10), (n/10, 1), (n/10, 1), (n/10, n/10), (n/10, n), (n/m, n), (n, n)\}$: Liu and Reeves (2001) with FF(x) instead of LR(x) heuristic.
  - Heuristics FF-ICH1, FF-ICH2 and FF-ICH3: Li et al. (2009) with FF(x) instead of LR(x) heuristic.
  - Heuristic FF-NEH(x) for $x = 5, 10$: Pan and Ruiz (2013) with FF(x) instead of LR(x) heuristic.
  - Heuristic Raj: Rajendran (1993).
  - Heuristic RZ: Rajendran and Ziegler (1997).
  - Heuristic RZ_LW: Li and Wu (2005).
  - Heuristics FF-PR1(x) for $x = [5, 10, 15]$: Pan and Ruiz (2013) with FF(x) instead of LR(x) heuristic.

- Heuristics adapted from the traditional PFSP to minimize makespan ($Fm|prmu|C_{max}$). Given a partial sequence, each evaluation of the makespan of this sequence is replaced by the evaluation of total flowtime with blocking:

  - Heuristic NEH proposed by Nawaz et al. (1983).

  – Heuristics FRB2, FRB3, FRB4$_\mathrm{k}$ (with $k = [2, 4, 6, 8, 10, 12]$) and FRB5: Rad et al. (2009). Due to the good results found by the NEH_WPT as compared to the original NEH, these heuristics are initialized in a non-decreasing sum of processing times.

Hence, a total of 36 heuristics are compared in this paper. In order to have a fair comparison under the same conditions, the following items are fulfilled:

- All 36 heuristics are fully re-coded in C# under the same compiler. Some of them have been executed for different values of the parameters, yielding a total of 70 heuristics which are tested.
- The same libraries and common functions are used for all implemented heuristics.
- All heuristics are tested under the same computer, an Intel Core i7-3770 with 3.4 GHz and 16 GB RAM.

5.4 Indicators to evaluate heuristics

Heuristics are evaluated and compared according to the quality of their solutions and their computational effort. Traditionally, the former is measured by the Average Relative Percentage Deviation –denoted as $ARPD_h$ for heuristic h, see Equation (16)–, while the Average CPU time –denoted as $ACPU_h$ for heuristic h, see Equation (17)– is the indicator used to measure the latter.

$$ARPD_h = \frac{\sum_{\forall s} RPD2_{h,s}}{S} \tag{16}$$

$$ACPU_h = \frac{\sum_{\forall s} T_{h,s}}{S} \tag{17}$$

where S is the total number of instances (with $s = 1..S$), $T_{h,s}$ is the CPU time of heuristic h in instance s, and $RPD2_{h,s}$ is defined by (18) being $Best_s$ the minimum total flowtime time among the implemented heuristics (available as on-line materials).

$$RPD2_{h,s} = \frac{OF_{h,s} - Best_s}{Best_s} \cdot 100 \tag{18}$$

5.5 Computational Evaluation of Heuristics

Each implemented heuristic is tested on benchmark $\mathcal{B}_2$. Computational results are shown in Table 1 in terms of $ARPD$ (second and fifth columns) and $ACPU$ (third and sixth columns). The best ARPDs are found by the proposed heuristic BS(x) ($\forall x \in \{5, 15, n/10, n\}$) being 1.239, 0.687, 1.029 and 0.333 respectively. Note the distance among the best heuristic BS(n) and the best non-proposed heuristics which is 1.682 found by PF-NEH$_{\text{LS}}$(5) heuristic. Furthermore, the BS(n) needs in average 42.73% less CPU time than PF-NEH$_{\text{LS}}$(5), i.e. $ACPU_{\text{BS}(n)}$ is 13.148 seconds while $ACPU_{\text{PF-NEH}_{\text{LS}}(5)}$ is 22.959 seconds. Graphically, the heuristics are shown in Figure 3. More detailed results of $ARPD$ and $ACPU$ for each size for the problem are shown in Tables 2 and 3 respectively. The proposed heuristic BS(x) ($\forall x \in \{2, 5, 15, n\}$) is efficient as there is no other heuristic with lower $ACPU$ and $ARPD$. The excellent performance of the proposed heuristic is also highlighted by the 33 the new upper bounds found for the problem. Their sequences and the objectives functions are available as on-line material. Note that its good performance presumably lies in the high number and the quality of the sequences evaluated using lower complexity order. On the one hand, let us compare the sequences evaluated and the complexity between BS(x) and the well-known NEH heuristic: the number of sequences evaluated by NEH is $\frac{n(n+1)}{2} - 1$, and its complexity order is $O(n^3m)$, whereas the number of sequences evaluated by BS(x) is $x\frac{n(n+1)}{2} - 1$, and its complexity order is $O(\max\{xn^2m, x^2n^2\})$. Thereby, e.g. BS(1) evaluates the same number of sequences than NEH, but with complexity $O(n^2m)$ instead of $O(n^3m)$. For BS($n/10$), the number of sequences evaluated is $\frac{n}{10}\frac{n(n+1)}{2} - 1$ and its complexity is still lower than $O(n^3m)$, since both $xn^2m = n^3m/10 < n^3m$ and $x^2n^2 = n^4/100 < n^3m$ for each instance in benchmark $\mathcal{B}_2$. On the other hand, each sequence evaluated in BS(x) comes from one of the best sequences in the previous iteration, which ensures the high quality of the sequences to be evaluated.

Table 4 summarises the $ARPD$ values of the most promising heuristics for different values of n and m. Each element of the table indicates the average $RPD2$ for each one of the heuristics in the first row, for the parameter value indicated in the first column, e.g. the value 1.11 of the third row and column is $ARPD_{BS(5)} = \frac{\sum_{\forall s \ | n=50} RPD2_{BS(5),s}}{\sum_{\forall s \ | n=50} 1}$. Note that the number of instances for each parameter is not the same in benchmark $\mathcal{B}_2$. Results show the good performance of

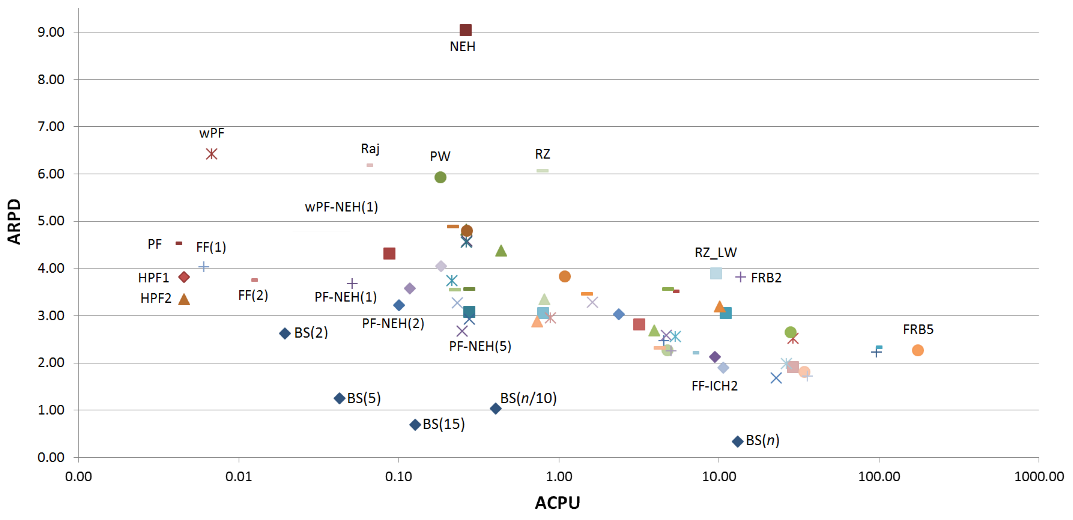

Figure 3: *ARPD* against *ACPU*. *X*-axis (*ACPU*) is shown in logarithmic scale. For sake of clarity, only the name of the most representative heuristics is depicted. The full results are shown in Tables 1.

the proposed heuristics BS($n/10$) and BS(n) with the increase in n, which probably lies in the proportional relationship between n and the beam width in both heuristics. Regarding m, the *ARPD* of the proposed heuristic decreases with m, whereas the performance of the improvement heuristics PF-NEH$_{\text{LS}}$(2), FF-FPE(n/10,n), FF-ICH1 and FF-ICH2 decrease as m increases.

Regarding heuristics adapted from related decision problems, some of them yield an excellent performance as compared to heuristics specifically implemented for the problem under study. Thereby, e.g. the heuristics PF-NEH(2), FF-FPE(n/10,1) and PF-NEH(5) (with an *ARPD* of 3.22, 3.27 and 2.67 respectively) clearly outperform NEH_WPT, MME_A, MME, MME_B and NPF (*ARPDs* of 4.82, 4.55, 4.58, 4.80 and 3.56 respectively) using less *ACPU*. PF-NEH(5) even slightly outperforms NHPF1 and NHPF2 with 3.08, and 2.92 of *ARPD* respectively. In fact, the best *ARPD* among the non-proposed heuristics is found by PF-NEH$_{\text{LS}}$(5), which was originally proposed for the $Fm|block|C_{max}$ problem.

In order to statistically justify the efficiency of the proposed heuristic, we compare it with the best heuristics requiring higher *ACPU*. We use a Holm's procedure (Holm, 1979) to contrast the following hypotheses:

- H_1: BS(5) = PF-NEH(2)

Table 1: *ARPDs* and *ACPUs* of the implemented heuristics (ordered by increasing *ACPU*). In bold it is indicated the proposed set of heuristics.

Heuristic	*ARPD*	*ACPU*	Heuristic	*ARPD*	*ACPU*
PF	4.529	0.004	FF-NEH(5)	3.340	0.813
HPF2	3.349	0.005	FF-FPE(n/10,n/10)	2.945	0.890
HPF1	3.813	0.005	PW-NEH(5)	3.828	1.091
FF(1)	4.028	0.006	FRB4_2	3.452	1.509
wPF	6.423	0.007	FF-NEH(10)	3.286	1.623
FF(2)	3.750	0.012	FRB4_4	3.025	2.386
BS(2)	**2.614**	**0.019**	FRB4_8	2.807	3.205
BS(5)	**1.239**	**0.043**	FRB4_8	2.684	3.946
wPF-NEH(1)	4.915	0.044	FF-ICH1	2.313	4.271
PF-NEH(1)	3.670	0.051	PF-NEH$_{\text{LS}}$(1)	2.462	4.548
Raj	6.184	0.063	FRB4_{10}	2.584	4.723
wPF-NEH(2)	4.304	0.087	FF-FPE(n/10,n)	2.258	4.776
PF-NEH(2)	3.221	0.101	PW-NEH$_{\text{LS}}$(1)	3.560	4.847
FF(n/m)	3.573	0.117	FF-FPE(n/m,n)	2.250	5.058
BS(15)	**0.687**	**0.127**	wPF-NEH$_{\text{LS}}$(1)	3.508	5.225
PW	5.926	0.182	FRB4_{12}	2.558	5.372
LR(1)	4.039	0.184	FF-FPE(n,n)	2.209	6.943
wPF-NEH(5)	3.732	0.216	PF-NEH$_{\text{LS}}$(2)	2.120	9.504
PW-NEH(1)	4.885	0.219	RZ_LW	3.891	9.651
FF(n/10)	3.548	0.224	PW-NEH$_{\text{LS}}$(2)	3.197	10.164
FF-FPE(n/10,1)	3.266	0.234	FF-ICH2	1.896	10.745
PF-NEH(5)	2.669	0.250	wPF-NEH$_{\text{LS}}$(2)	3.056	11.037
NEH	9.043	0.262	**BS(n)**	**0.333**	**13.148**
NEH_WPT	4.816	0.264	FRB2	3.814	13.749
MME_A	4.553	0.266	PF-NEH$_{\text{LS}}$(5)	1.682	22.959
MME	4.576	0.267	FF-PR1(5)	1.978	26.477
MME_B	4.797	0.268	PW-NEH$_{\text{LS}}$(5)	2.647	28.197
NHPF1	3.080	0.277	wPF-NEH$_{\text{LS}}$(5)	2.516	29.102
NHPF2	2.921	0.277	FF-ICH3	1.902	29.157
NPF	3.563	0.277	FF-PR1(10)	1.801	34.279
BS(n/10)	**1.029**	**0.403**	FF-PR1(15)	1.720	35.965
PW-NEH(2)	4.375	0.439	FRB3	2.321	96.546
FF-FPE(15,n/10)	2.879	0.732	NEH-MK	2.229	96.771
RZ	6.066	0.792	FRB5	2.269	176.403
FF-FPE(2,n/10)	3.057	0.801			

Table 2: Detailed values of *ARPD* for each size of the problem. The proposed set of heuristics is indicated in bold.

Heuristic	Size of the problem (n x m)												$ARPD$
	20x5	20x10	20x20	50x5	50x10	50x20	100x5	100x10	100x20	200x10	200x20	500x20	
NEH_WPT	2.92	2.84	3.39	5.03	4.31	3.56	6.70	5.31	4.39	6.97	5.36	7.00	4.816
MME	2.88	2.74	2.17	5.16	3.81	3.31	5.94	5.27	3.81	7.08	5.67	7.06	4.576
MME_A	2.76	2.69	2.20	5.02	3.88	3.21	5.90	5.26	4.30	7.03	5.35	7.05	4.553
MME_B	3.39	3.26	3.34	4.89	4.15	3.35	6.40	5.41	4.84	6.69	5.17	6.69	4.797
NEH-MK	1.13	0.92	0.69	2.42	1.51	0.93	3.50	2.42	1.84	4.22	2.82	4.35	2.229
PF	4.78	4.63	4.23	5.26	3.90	4.81	7.80	4.47	3.96	4.16	3.39	2.94	4.529
HPF1	4.04	4.74	3.87	3.12	4.12	4.64	2.92	3.59	4.48	3.26	3.86	3.11	3.813
HPF2	3.71	3.02	3.80	2.99	2.50	3.96	2.73	2.94	4.72	2.94	3.64	3.25	3.349
NPF	2.62	2.49	2.65	4.15	2.94	3.34	6.41	4.18	3.49	4.16	3.39	2.94	3.563
NHPF1	2.60	2.58	2.67	2.89	2.99	3.26	2.86	3.42	3.69	3.15	3.74	3.11	3.080
NHPF2	2.73	2.21	2.58	2.71	2.50	3.14	2.74	2.94	3.80	2.83	3.63	3.25	2.921
BS(2)	1.64	1.90	2.34	2.65	2.25	2.99	2.65	2.51	3.04	2.73	3.50	3.18	2.614
BS(5)	0.50	1.04	1.72	1.17	0.71	1.44	1.25	1.09	1.60	1.33	1.46	1.56	1.239
BS(15)	0.17	1.13	1.64	0.33	0.41	0.94	0.68	0.49	0.41	0.67	0.54	0.84	0.687
BS(n/10)	1.64	1.90	2.34	1.17	0.71	1.44	0.71	0.48	0.74	0.44	0.39	0.39	1.029
BS(n)	0.21	1.09	1.56	0.11	0.12	0.58	0.12	0.03	0.07	0.00	0.09	0.00	0.333
wPF	6.11	4.77	3.58	8.64	7.08	4.06	10.47	7.10	4.18	9.29	5.73	6.07	6.423
PW	7.22	2.73	2.62	9.58	6.11	2.46	10.57	6.60	3.63	8.67	5.14	5.77	5.926
PF-NEH(1)	2.87	2.49	2.73	4.35	2.90	3.87	7.10	4.02	3.75	4.17	2.98	2.81	3.670
PF-NEH(2)	2.45	2.08	2.13	4.29	2.15	2.88	6.55	3.45	3.03	4.02	2.89	2.72	3.221
PF-NEH(5)	1.96	1.45	1.63	3.52	1.97	2.26	5.57	2.43	2.67	3.51	2.62	2.43	2.669
wPF-NEH(1)	2.87	2.22	2.40	6.16	4.98	3.04	8.75	5.40	3.60	8.73	5.18	5.64	4.915
wPF-NEH(2)	2.61	1.92	1.88	5.50	3.58	2.44	8.00	5.26	3.11	7.47	4.53	5.35	4.304
wPF-NEH(5)	1.81	1.74	1.47	4.68	3.06	1.73	7.25	4.86	2.51	6.69	3.86	5.14	3.732
PW-NEH(1)	2.84	2.58	2.86	6.78	4.44	1.89	9.52	5.85	3.29	8.15	4.82	5.61	4.885
PW-NEH(2)	2.62	2.39	1.63	5.61	3.88	1.77	9.25	5.32	2.80	7.68	4.23	5.34	4.375
PW-NEH(5)	1.96	1.73	1.44	5.26	3.09	1.50	7.54	4.97	2.60	6.74	3.90	5.21	3.828
PF-NEH$_{\mathrm{LS}}$(1)	1.52	1.08	1.04	3.30	2.04	1.21	5.41	3.17	2.30	3.70	2.37	2.39	2.462
PF-NEH$_{\mathrm{LS}}$(2)	1.03	0.89	0.71	3.10	1.30	1.01	4.92	2.58	1.85	3.54	2.22	2.27	2.120
PF-NEH$_{\mathrm{LS}}$(5)	0.65	0.47	0.33	2.48	0.97	0.71	4.03	1.88	1.48	3.12	1.95	2.10	1.682
wPF-NEH$_{\mathrm{LS}}$(1)	1.78	1.21	0.98	4.59	3.13	1.47	5.52	4.32	2.87	6.53	4.45	5.25	3.508
wPF-NEH$_{\mathrm{LS}}$(2)	1.09	1.01	0.66	3.88	2.38	1.22	4.87	4.13	2.38	6.15	4.00	4.93	3.056
wPF-NEH$_{\mathrm{LS}}$(5)	0.62	0.49	0.30	3.16	1.97	0.65	4.22	3.64	1.92	5.23	3.31	4.69	2.516
PW-NEH$_{\mathrm{LS}}$(1)	1.91	0.81	0.77	5.12	3.29	1.31	6.20	4.34	2.74	6.67	4.24	5.33	3.560
PW-NEH$_{\mathrm{LS}}$(2)	1.37	0.68	0.63	4.40	2.63	1.20	6.16	3.98	2.33	6.06	3.82	5.10	3.197
PW-NEH$_{\mathrm{LS}}$(5)	0.94	0.48	0.37	3.60	1.93	0.78	4.72	3.40	1.97	5.25	3.49	4.82	2.647
LR(1)	4.30	2.99	2.25	6.00	3.50	2.25	7.13	5.11	2.60	5.59	2.94	3.81	4.039
FF(1)	3.99	2.74	2.34	6.21	3.36	2.38	6.74	5.68	2.63	5.28	3.21	3.77	4.028
FF(2)	3.64	2.63	1.96	5.65	3.30	2.28	6.33	5.23	2.35	4.92	3.10	3.61	3.750
FF(n/10)	3.64	2.63	1.96	5.21	3.14	2.21	6.17	4.43	2.10	4.74	2.95	3.38	3.548
FF(n/m)	3.47	2.63	2.34	5.15	3.14	2.28	6.10	4.43	2.18	4.74	2.97	3.43	3.573
FF-FPE(2,n/10)	2.54	2.04	1.53	4.82	2.91	1.71	5.19	3.83	2.03	4.13	2.78	3.18	3.057
FF-FPE(15,n/10)	2.46	2.04	1.51	4.30	2.72	1.55	4.91	3.46	1.83	3.99	2.70	3.07	2.879
FF-FPE(n/10,1)	3.07	2.19	1.54	4.92	2.98	1.95	5.88	4.05	1.97	4.52	2.86	3.26	3.266
FF-FPE(n/10,n/10)	2.54	2.04	1.53	4.57	2.95	1.56	5.10	3.60	1.83	3.99	2.71	2.93	2.945
FF-FPE(n/10,n)	1.53	1.43	1.10	3.22	1.99	1.28	3.30	2.93	1.65	3.41	2.53	2.73	2.258
FF-FPE(n/m,n)	1.57	1.43	1.12	3.07	1.99	1.27	3.17	2.93	1.70	3.41	2.50	2.84	2.250
FF-FPE(n,n)	1.57	1.43	1.11	3.07	1.91	1.14	3.17	2.88	1.65	3.36	2.51	2.72	2.209
FF-ICH1	1.74	0.96	0.78	3.19	2.03	1.12	4.28	2.85	1.64	3.72	2.51	2.93	2.313
FF-ICH2	1.46	0.76	0.72	2.38	1.64	0.92	2.94	2.39	1.51	2.99	2.27	2.78	1.896
FF-ICH3	1.55	0.76	0.77	2.41	1.74	0.84	2.93	2.21	1.48	3.08	2.29	2.76	1.902
FF-NEH(5)	2.60	2.01	1.60	5.09	2.89	1.93	5.98	4.53	2.16	4.75	2.96	3.57	3.340
FF-NEH(10)	2.60	2.01	1.59	5.02	2.89	1.92	5.93	4.23	2.06	4.69	2.96	3.54	3.286
Raj	4.42	3.20	3.54	6.22	4.91	4.96	9.43	6.62	6.21	9.28	6.52	8.89	6.184
RZ	3.68	2.29	2.01	6.52	5.11	3.46	9.07	7.48	5.40	10.33	7.62	9.83	6.066
RZ_LW	1.48	1.81	0.92	4.12	3.39	2.51	5.67	4.77	3.65	6.35	5.06	6.96	3.891
FF-PR1(5)	0.59	0.65	0.29	2.37	1.35	0.60	4.02	2.98	1.55	3.78	2.36	3.19	1.978
FF-PR1(10)	0.43	0.39	0.22	2.05	1.31	0.42	3.85	2.60	1.38	3.53	2.26	3.17	1.801
FF-PR1(15)	0.37	0.36	0.21	2.05	1.09	0.41	3.57	2.38	1.31	3.51	2.22	3.17	1.720
NEH	6.26	5.27	3.69	11.27	8.53	6.51	12.93	10.74	8.09	13.83	9.87	11.55	9.043
FRB2	1.48	1.75	0.62	4.07	3.07	1.52	6.24	5.06	3.05	7.19	4.92	6.81	3.814
FRB3	1.30	0.88	0.71	2.73	1.58	1.15	3.82	2.57	1.87	4.05	2.88	4.31	2.321
FRB4$_2$	1.95	1.67	1.65	3.40	2.92	1.71	5.27	3.58	3.02	6.13	4.17	5.95	3.452
FRB4$_4$	1.62	1.16	1.13	3.10	2.43	1.73	4.65	3.62	2.47	5.24	3.69	5.46	3.025
FRB4$_8$	1.49	1.23	0.75	2.96	2.37	1.52	4.23	3.07	2.52	4.87	3.46	5.21	2.807
FRB4$_8$	1.31	1.02	0.86	2.90	1.84	1.41	4.32	3.01	2.28	4.84	3.49	4.93	2.684
FRB4$_{10}$	1.30	0.96	0.77	2.93	1.75	1.40	4.22	2.81	2.08	4.49	3.28	5.01	2.584
FRB4$_{12}$	1.30	0.96	0.81	2.74	1.69	1.40	4.13	2.72	2.36	4.44	3.21	4.93	2.558
FRB5	1.25	0.94	0.69	2.58	1.63	0.79	3.87	2.56	1.76	4.13	2.86	4.16	2.269

Table 3: Detailed values of CPU times for each size of the problem. The proposed set of heuristics is indicated in bold.

Heuristic	Size of the problem (n x m)												$ACPU$
	20x5	20x10	20x20	50x5	50x10	50x20	100x5	100x10	100x20	200x10	200x20	500x20	
NEH_WPT	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.07	0.16	2.89	0.264
MME	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.07	0.16	2.91	0.267
MME_A	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.07	0.16	2.90	0.266
MME_B	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.07	0.16	2.92	0.268
NEH-MK	0.00	0.00	0.01	0.03	0.06	0.11	0.39	0.72	1.53	10.39	24.24	1123.77	96.771
PF	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.03	0.004
HPF1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.04	0.005
HPF2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.04	0.005
NPF	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.08	0.17	3.02	0.277
NHPF1	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.08	0.17	3.02	0.277
NHPF2	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.08	0.17	3.02	0.277
BS(2)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.04	0.14	0.019
BS(5)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.05	0.07	0.33	0.043
BS(15)	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.04	0.06	0.12	0.18	1.04	0.127
BS(n/10)	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.03	0.04	0.16	0.25	4.33	0.403
BS(n)	0.00	0.00	0.00	0.04	0.05	0.06	0.28	0.33	0.45	4.10	5.03	147.44	13.148
wPF	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.05	0.007
PW	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.07	0.14	1.92	0.182
PF-NEH(1)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.03	0.07	0.48	0.051
PF-NEH(2)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.05	0.13	0.96	0.101
PF-NEH(5)	0.00	0.00	0.00	0.01	0.01	0.02	0.02	0.03	0.07	0.13	0.32	2.39	0.250
wPF-NEH(1)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.03	0.06	0.41	0.044
wPF-NEH(2)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.05	0.11	0.82	0.087
wPF-NEH(5)	0.00	0.00	0.00	0.01	0.01	0.02	0.02	0.03	0.06	0.12	0.28	2.05	0.216
PW-NEH(1)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.09	0.19	2.27	0.219
PW-NEH(2)	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.06	0.18	0.37	4.58	0.439
PW-NEH(5)	0.00	0.00	0.00	0.01	0.02	0.03	0.04	0.07	0.15	0.44	0.91	11.41	1.091
PF-NEH$_{LS}$(1)	0.00	0.00	0.00	0.01	0.02	0.04	0.06	0.11	0.23	1.00	2.37	50.74	4.548
PF-NEH$_{LS}$(2)	0.00	0.00	0.01	0.02	0.03	0.07	0.11	0.23	0.48	1.84	5.07	106.19	9.504
PF-NEH$_{LS}$(5)	0.00	0.01	0.01	0.03	0.07	0.17	0.28	0.50	1.30	4.50	12.51	256.12	22.959
wPF-NEH$_{LS}$(1)	0.00	0.00	0.00	0.01	0.02	0.04	0.07	0.11	0.25	1.25	2.25	58.69	5.225
wPF-NEH$_{LS}$(2)	0.00	0.00	0.01	0.02	0.03	0.07	0.13	0.21	0.47	2.27	5.03	124.21	11.037
wPF-NEH$_{LS}$(5)	0.00	0.01	0.01	0.04	0.08	0.16	0.34	0.55	1.28	5.75	13.88	327.12	29.102
PW-NEH$_{LS}$(1)	0.00	0.00	0.00	0.01	0.02	0.03	0.08	0.13	0.29	1.25	3.12	53.22	4.847
PW-NEH$_{LS}$(2)	0.00	0.00	0.01	0.02	0.04	0.07	0.15	0.24	0.58	2.72	6.28	111.86	10.164
PW-NEH$_{LS}$(5)	0.00	0.01	0.01	0.04	0.08	0.16	0.37	0.59	1.43	6.35	14.57	314.74	28.197
LR(1)	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.07	0.14	1.95	0.184
FF(1)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.05	0.006
FF(2)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.10	0.012
FF(n/10)	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.08	0.16	2.40	0.224
FF(n/m)	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.08	0.08	1.20	0.117
FF-FPE(2,n/10)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.04	0.14	0.38	9.01	0.801
FF-FPE(15,n/10)	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.06	0.20	0.45	7.99	0.732
FF-FPE(n/10,1)	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.09	0.17	2.49	0.234
FF-FPE(n/10,n/10)	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.05	0.22	0.49	9.86	0.890
FF-FPE(n/10,n)	0.00	0.00	0.00	0.01	0.01	0.03	0.07	0.11	0.23	1.04	2.41	53.40	4.776
FF-FPE(n/m,n)	0.00	0.00	0.00	0.01	0.01	0.03	0.08	0.11	0.22	1.04	2.33	56.86	5.058
FF-FPE(n,n)	0.00	0.00	0.00	0.02	0.03	0.05	0.11	0.20	0.40	1.73	3.76	77.00	6.943
FF-ICH1	0.00	0.00	0.00	0.01	0.02	0.03	0.07	0.11	0.25	0.90	1.97	47.88	4.271
FF-ICH2	0.00	0.00	0.01	0.02	0.03	0.06	0.14	0.23	0.46	2.42	5.90	119.68	10.745
FF-ICH3	0.00	0.00	0.01	0.03	0.04	0.08	0.34	0.50	0.82	6.42	10.97	330.67	29.157
FF-NEH(5)	0.00	0.00	0.00	0.01	0.01	0.01	0.02	0.03	0.07	0.22	0.50	8.88	0.813
FF-NEH(10)	0.00	0.00	0.00	0.01	0.01	0.02	0.04	0.06	0.13	0.43	1.00	17.76	1.623
Raj	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.04	0.67	0.063
RZ	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.06	0.20	0.47	8.71	0.792
RZ_LW	0.00	0.00	0.00	0.01	0.02	0.04	0.08	0.14	0.41	1.60	4.42	109.08	9.651
FF-PR1(5)	0.00	0.01	0.01	0.04	0.07	0.15	0.26	0.55	1.20	4.97	12.14	298.32	26.477
FF-PR1(10)	0.01	0.01	0.02	0.08	0.14	0.31	0.54	1.13	2.44	10.20	23.93	372.53	34.279
FF-PR1(15)	0.01	0.02	0.03	0.11	0.21	0.47	0.83	1.68	3.81	15.22	36.66	372.54	35.965
NEH	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.02	0.07	0.16	2.87	0.262
FRB2	0.00	0.00	0.01	0.01	0.03	0.14	0.06	0.23	0.97	1.67	8.19	153.68	13.749
FRB3	0.00	0.00	0.01	0.03	0.06	0.11	0.38	0.70	1.49	10.25	23.88	1121.64	96.546
FRB4$_2$	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.05	0.11	0.38	0.89	16.60	1.509
FRB4$_4$	0.00	0.00	0.00	0.01	0.01	0.03	0.04	0.08	0.17	0.59	1.41	26.29	2.386
FRB4$_8$	0.00	0.00	0.00	0.01	0.02	0.03	0.06	0.11	0.23	0.77	1.86	35.38	3.205
FRB4$_8$	0.00	0.00	0.00	0.01	0.02	0.04	0.07	0.13	0.28	0.95	2.29	43.56	3.946
FRB4$_{10}$	0.00	0.00	0.01	0.01	0.02	0.04	0.08	0.15	0.32	1.12	2.69	52.23	4.723
FRB4$_{12}$	0.00	0.00	0.01	0.02	0.03	0.05	0.09	0.17	0.37	1.28	3.06	59.39	5.372
FRB5	0.00	0.00	0.01	0.04	0.08	0.17	0.62	1.11	2.61	17.50	41.14	2053.54	176.403

Parameter	BS(2)	BS(5)	BS(15)	BS(n/10)	BS(n)	HPF2	PF-NEH(5)	PF-NEH$_{\text{LS}}$(2)	FF-FPE(n/10,n)	FF-ICH1	FF-ICH2
n=20	1.96	1.09	0.98	1.96	0.95	3.51	1.68	0.49	1.35	1.16	0.98
n=50	2.63	1.11	0.56	1.11	0.27	3.15	2.58	1.39	2.16	2.11	1.65
n=100	2.73	1.31	0.52	0.64	0.08	3.46	3.56	2.46	2.63	2.92	2.28
n=200	3.11	1.39	0.61	0.41	0.05	3.29	3.06	2.53	2.97	3.12	2.63
n=500	3.18	1.56	0.84	0.39	0.00	3.25	2.43	2.10	2.73	2.93	2.78
m=5	2.31	0.97	0.39	1.18	0.15	3.14	3.68	2.39	2.68	3.07	2.26
m=10	2.35	1.04	0.68	0.88	0.31	2.85	2.34	1.61	2.44	2.39	1.95
m=20	3.01	1.56	0.87	1.06	0.46	3.87	2.32	1.31	1.86	1.80	1.64

Table 4: *ARPDs* of the most promising heuristics depending on the parameters n and m

i	H_i	p-value	Mann-Whitney	$\alpha/(k-i+1)$	Holm's Procedure
1	BS(5) = PF-NEH(2)	0.000	R	0.0167	R
2	BS(15) = FF-ICH2	0.000	R	0.0250	R
3	BS(n) = PF-NEH$_{\text{LS}}$(5)	0.000	R	0.0500	R

Table 5: Holm's procedure.

- H_2: BS(15) = FF-ICH2
- H_3: BS(n) = PF-NEH$_{\text{LS}}$(5)

We use a non-parametric Mann-Whitney test assuming a 0.95 confidence level (i.e. $\alpha = 0.05$) to establish the p-value of each hypothesis (see e.g. Pan et al., 2008a and Fernandez-Viagas and Framinan, 2015b for similar statistical approach). In Holm's procedure, a hypothesis i among a total of k (ordered in ascending order of p-values) is rejected if its p-value is lower than $\alpha/(k-i+1)$. The results of the procedure are shown in Table 5. Each p-value is 0.000 and therefore, each hypothesis can be rejected.

Regarding the fastest heuristics, i.e. HPF1, HPF2, PF, wPF, FF(1) and FF(2), the best *ARPD* is found by HPF2. We perform again a Mann-Whitney test to establish the efficiency of HPF1 using the same confidence. We compare it with both HPF2 and FF(2). Results are shown in Table 6. There is no statistical significant difference between HPF2 and FF(2).

Hypothesis	p-value	Mann-Whitney
HPF2 = HFP1	0.043	R
HPF2 = FF(2)	0.129	

Table 6: Comparison of HPF2 against HPF1 and FF(2) using a Mann-Whitney non-parametric test.

6 Conclusions

In this paper, an efficient beam-search-based constructive heuristic is proposed. The heuristic constructs a pool of partial sequences in each iteration appending jobs at the end of the most promising sequences. An index based on the idle, blocking and departure times of the jobs is proposed to determine the jobs selected in each iteration. Thereby, the heuristic adopts a beam-search-based strategy which successfully combines the diversification of population-based algorithms and the speed of constructive heuristics.

The proposed heuristic is compared with the best known constructive and improvement heuristics both for the problem under consideration and for related scheduling problems. A total of 36 heuristics are tested in an exhaustive computational evaluation using the set of instances of Taillard (1993), where each heuristic has been reimplemented in C# to perform a fair comparison. Additionally, a speed up procedure has been proposed to accelerate the insertion phases of each heuristic. This procedure has been included in each insertion phase if applicable.

Among the implemented heuristics, the best $ARPD$ are found by the proposed heuristic BS(x) ($\forall x \in \{5, 15, n/10, n\}$). 33 new upper bounds for the well-known Taillard benchmark are found by these heuristics (which means that new best-so-far solutions have been found for more than 27% of these instances). The computational experience also highlights the good performance of several heuristics adapted from related scheduling problems, particularly from the $Fm|block|C_{max}$ problem. This fact may speak for certain correlation between both problems and opens some avenues for further research.

Acknowledgements

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

References

Bao, Y., Zheng, L., and Jiang, H. (2012). An improved hs algorithms for the blocking flow shop scheduling problems. *Proceedings - 2012 International Conference on Computer Science and Information Processing, CSIP 2012*, pages 1289–1291.

Deng, G., Xu, Z., and 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.

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

Fernandez-Viagas, V. and Framinan, J. (2015a). A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, 53(4):1111–1123.

Fernandez-Viagas, V. and Framinan, J. (2015b). Efficient non-population-based algorithms for the permutation flowshop scheduling problem with makespan minimisation subject to a maximum tardiness. *Computers & Operations Research*, 64(0):86 – 96.

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

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

Ferrer, A., Guimarans, D., Ramalhinho, H., and Juan, A. (2016). A brils metaheuristic for non-smooth flow-shop problems with failure-risk costs. *Expert Systems with Applications*, 44:177–186.

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., 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. (2005). Comparison of heuristics for flowtime minimisation in permutation flowshops. *Computers & Operations Research*, 32(5):1237–1254.

Framinan, J. M., Leisten, R., and Rajendran, C. (2003). Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *International Journal of Production Research*, 41(1):121–148.

Gong, H., Tang, L., and Duin, C. (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.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics*, 5:287–326.

Hall, N. and Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3):510–525.

Han, Y.-Y., Duan, J.-H., Yang, Y.-J., Zhang, M., and Yun, B. (2011). Minimizing the total flowtime flowshop with blocking using a discrete artificial bee colony. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6839 LNAI:91–97.

Han, Y.-Y., Liang, J., Pan, Q.-K., Li, J.-Q., Sang, H.-Y., and Cao, N. (2013). Effective hybrid discrete artificial bee colony algorithms for the total flowtime minimization in the blocking flowshop problem. *International Journal of Advanced Manufacturing Technology*, 67(1-4):397–414.

Han, Y.-Y., Pan, Q.-K., Li, J.-Q., and Sang, H.-Y. (2012). An improved artificial bee colony algorithm for the blocking flowshop scheduling problem. *International Journal of Advanced Manufacturing Technology*, 60(9-12):1149–1159.

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

Kalczynski, P. J. and Kamburowski, J. (2007). On the NEH heuristic for minimizing the makespan in permutation flow shops. *OMEGA, The International Journal of Management Science*, 35(1):53–60.

Kalczynski, P. J. and Kamburowski, J. (2008). An improved NEH heuristic to minimize makespan in permutation flow shops. *Computers & Operations Research*, 35(9):3001–3008.

Kalczynski, P. J. and Kamburowski, J. (2009). An empirical analysis of the optimality rate of flow shop heuristics. *European Journal of Operational Research*, 198(1):93 – 101.

Khorasanian, D. and Moslehi, G. (2012). An iterated greedy algorithm for solving the blocking flow shop scheduling problem with total flow time criteria. *International Journal of Industrial Engineering & Production Research*, 23(4):301–308.

Krajewski, L., King, B., Ritzman, L., and Wong, D. (1987). Kanban, MRP, and shaping the manufacturing environment. *Management Science*, 33:39–57.

Leisten, R. (1990). Flowshop sequencing problems with limited buffer storage. *International Journal of Production Research*, 28(11):2085–2100.

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

Li, X. and Wu, C. (2005). An efficient constructive heuristic for permutation flow shops to minimize total flowtime. *Chinese Journal of Electronics*, 14(2):203–208.

Lin, S.-W., Huang, C.-Y., Lu, C.-C., and Ying, K.-C. (2012). Minimizing total flow time in permutation flowshop environment. *International Journal of Innovative Computing, Information and Control*, 8(10 A):6599–6612.

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.

McCormick, S., Pinedo, M. L., Shenker, S., and Wolf, B. (1989). Sequencing in an assembly line with blocking to minimize cycle time. *Operations Research*, 37(6):925–935.

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

Naderi, B. and Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, 37(4):754–768.

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. (2013). A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers & Operations Research*, 40(1):117–128.

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

Pan, Q.-K. and Wang, L. (2012). Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. *Omega*, 40(2):218–229.

Pan, Q.-K., Wang, L., and Zhao, B.-H. (2008b). An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. *International Journal of Advanced Manufacturing Technology*, 38(7-8):778–786.

Rad, S. F., Ruiz, R., and Boroojerdian, N. (2009). New high performing heuristics for minimizing makespan in permutation flowshops. *OMEGA, The International Journal of Management Science*, 37(2):331–345.

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.

Reklaitis, G. (1982). Review of scheduling of process operations. *AIChE Symposium Series*, 78(214):119–133.

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

Ribas, I. and Companys, R. (2015). Efficient heuristic algorithms for the blocking flow shop scheduling problem with total flow time minimization. *Computers and Industrial Engineering*, 87:30–39.

Ribas, I., Companys, R., and Tort-Martorell, X. (2010). Comparing three-step heuristics for the permutation flow shop problem. *Computers & Operations Research*, 37(12):2062–2070.

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

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

Ribas, I., Companys, R., and Tort-Martorell, X. (2015). An efficient discrete artificial bee colony algorithm for the blocking flow shop problem with total flowtime minimization. *Expert Systems with Applications*, 42(15-16).

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

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.

Sethi, S., Sriskandarajah, C., Sorger, G., Blazewicz, J., and Kubiak, W. (1992). Sequencing of parts and robot moves in a robotic cell. *International Journal of Flexible Manufacturing Systems*, 4(3-4):331–358.

Storer, R., Wu, S., and Vaccari, R. (1992). New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38:1495–1509.

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

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

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

Vasiljevic, D. and Danilovic, M. (2015). Handling ties in heuristics for the permutation flow shop scheduling problem. *Journal of Manufacturing Systems*, 35:1–9.

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

Ying, K.-C. and Lin, S.-W. (2013). A high-performing constructive heuristic for minimizing makespan in permutation flowshops. *Journal of Industrial and Production Engineering*, 30(6):355–362.

Zanakis, S., Evans, J., and Vazacopoulos, A. (1989). Heuristic methods and applications: A categorized survey. *European Journal of Operational Research*, 43(1):88–110.

Zäpfel, G., Braune, R., and Bögl, M. (2010). *Metaheuristic Search Concepts*. Springer.