A new set of high-performing heuristics to minimize flowtime in permutation flowshops*

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

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

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

May 21, 2015

Abstract

This paper addresses the problem of scheduling jobs in a permutation flowshop with the objective of total completion time minimisation. Since this problem is known to be NP-hard, most research has focused on obtaining procedures –heuristics– able to provide good, but not necessarily optimal, solutions with a reasonable computational effort. Therefore, a full set of heuristics efficiently balancing both aspects (quality of solutions and computational effort) has been developed. 12 out of these 14 efficient procedures are composite heuristics based on the LR heuristic by Liu and Reeves (2001), which is of complexity n^3m. In our paper, we propose a new heuristic of complexity n^2m for the problem, which turns out to produce better results than LR. Furthermore, by replacing the heuristic LR by our proposal in the aforementioned composite heuristics, we obtain a new set of 17 efficient heuristics for the problem, with 15 of them incorporating our proposal. Additionally, we also discuss some issues related to the evaluation of efficient heuristics for the problem, an propose an alternative indicator.

Keywords: Scheduling, Flowshop, Heuristics, Flowtime.

*Preprint submitted to Computers & Operations Research, http://dx.doi.org/10.1016/j.cor.2014.08.004.

†Corresponding author. Tel.: +34-954487214; fax: +34-954487329.

1 Introduction

A flowshop is a common layout in many manufacturing scenarios (see e.g. Vakharia and Wemmerlov, 1990; Krajewski et al., 1987; Storer et al., 1992) where n jobs must be processed on m machines in the same order. The so-called flowshop scheduling problem consists in finding a sequence of jobs for each machine so certain performance measure(s) is(are) minimised. Additionally, it is customary to assume that the job sequences will be the same on every machine (permutation flowshops), along with other hypotheses such as e.g. the simultaneous availability of all jobs and of all machines, deterministic processing times, etc (for a complete list of these assumptions, see e.g. Dudek and Teuton, 1964).

Among the objectives considered in the flowshop scheduling problem, the minimisation of the sum of the completion times of the jobs (or equivalently mean completion time) has been consistently pointed out both as relevant and meaningful for today's dynamic production environment (Liu and Reeves, 2001). Under the assumption of a zero release time for the jobs, the minimization of total (average) completion time is equivalent to total (average) flowtime minimisation, which leads to stable or even use of resources, a rapid turn-around of jobs and the minimisation of in-process inventory (Rajendran and Ziegler, 1997). The flowshop scheduling problem with flowtime objective (denoted as $F|prmu|\sum C_j$, according to the notation by Graham et al., 1979) is known to be NP-hard, therefore most of the research on this topic is devoted to developing approaches yielding good (but not necessarily optimal) solutions in reasonable computation time. Obviously, in such approximate methods –or heuristics–, one may expect a trade-off between quality of solution and computation time so better solutions are obtained by heuristics requiring longer CPU times. Recently, Pan and Ruiz (2013) present an exhaustive evaluation of the different heuristics proposed for the problem in the literature taking into account the quality of the solutions (measured as the average relative percentage deviation over the best known solution) and the CPU time (in seconds). Using these two indicators as in a bicriteria decision problem, they derive a set of non-dominated (i.e. approximation of a Pareto set) heuristics. This 14-heuristics set can thus be used as a benchmark to propose new efficient heuristics for the problem.

A detailed analysis of this Pareto set reveals that 12 out of the 14 heuristics employ a mechanism for constructing the solutions based in the heuristic by Liu and Reeves (2001). In this paper, we propose a new heuristic that improves the results with respect to that by Liu and Reeves both in terms of quality of the solutions and in CPU time. By embedding this new heuristic in several heuristics in the Pareto set, we obtain a completely new efficient Pareto set. Additionally, since the indicators used in the Pareto set by Pan and Ruiz (2013) penalise certain type of heuristics, we propose an alternative way to measure their efficiency.

The rest of the paper is organised as follows: In Section 2, the formal problem statement and the state-of-the-art heuristics are given. Section 3 analyses some issues related with the performance evaluation of the different heuristics for the problem, and propose some alternative indicators. In Section 4, a new set of heuristic is presented for the problem. The computational evaluations are carried out in Section 5. Finally, conclusions are discussed in Section 6.

2 Problem statement and state-of-the-art

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

As mentioned in the previous section, a great number of heuristics have been proposed for the problem. For a detailed presentation and evaluation of all these heuristics, we refer the interested reader to Pan and Ruiz (2013), and we will describe here only a sub-set which are found to be state-of-the-art and consequently are the ones used in this paper for comparison.

According to Framinan et al. (2005), heuristics can include one or several of the following phases: index development, solution construction and solution improvement. A heuristic is deemed *composite* if it employs another heuristic for one or more of the three above-mentioned phases. Otherwise, it is regarded as *simple*.

The heuristics that we will considered in our paper are the following:

- Heuristic $LR(x)$ (Liu and Reeves, 2001). This heuristic constructs a solution for the problem by appending, one by one, the unscheduled jobs (jobs in set U in the following) at the end of a sequence S of already scheduled jobs. To do so, ξ_{jk} an indicator of the suitability for job j ($j \in U$) to be scheduled in last position (position $k+1$ where k indicates the amount of scheduled jobs in each iteration) is calculated according to:

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

where IT_{jk} estimates the weighted idle time induced when scheduling job j in position $k+1$, i.e.:

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

and AT_{jk} is the so-called artificial flowtime and it is defined as the sum of the completion time of job j plus the completion time of job p, an artificial job with processing times equal to the average processing time of the other jobs in U (excluding job j), and can be computed as follows:

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

More specifically, the $LR(x)$ heuristic operates as follows:

1. Sort all jobs in ascending order of indicator ξ_{j0} (Let us U denote such ordered set). Ties are broken in favor of jobs with higher IT_{j0}.
2. Use each of the first x ranked jobs in U as the first job in S, and then constructs a solution by appending the rest of the jobs one by one using indicator ξ_{jk}
3. Out of the x solutions so obtained, select the one with the minimum flowtime.

- Heuristic $LR(x) - FPE(y)$ (Liu and Reeves, 2001). This is a composite heuristic where a local search method (denoted $FPE(y)$) is applied to the solution of $LR(x)$. $FPE(y)$ consists of the following steps: For each job j in a sequence, this job is exchanged with the

next y jobs in the sequence, and the flowtimes of the so-obtained solutions are evaluated. If any of the solutions has improved the flowtime, then the local search procedure is repeated. Otherwise, the local search stops.

- Heuristic NEH (Nawaz et al., 1983). Originally conceived for minimizing the makespan in a permutation flowshop, this well-known algorithm has been used as a reference method for many problems in the literature. Its application to the flowtime minimisation problem was discussed by Framinan et al. (2003), and it was found that the best option is to first sort the jobs in ascending sum of their processing times. Then, a job sequence is constructed in the following manner: Assuming a sequence already determined for the first $k-1$ jobs, k candidate (sub)sequences are obtained by inserting job k in the k possible slots of the current sequence. Out of these k (sub)sequences, the one yielding the minimum flowtime is kept as relative (sub)sequence for these first k jobs given by phase one. Then, job $k+1$ from the first phase is considered analogously, and so on until all n jobs have been sequenced.

- Heuristic Raj (Rajendran, 1993): This heuristic can be seen as a version of the NEH, but here job k is inserted only in slots $\lfloor k/2 \rfloor$ to k, thus reducing the computation time. Additionally, jobs are initially sorted in ascending order of index T_j as defined in equation (1), breaking ties in favor of the job with the lowest sum of total processing times.

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

- Heuristic $LR - NEH(x)$ (Pan and Ruiz, 2013). This is a composite heuristic where the last $n/4$ steps of each x sequences obtained applying the $LR(x)$ procedure are carried out according to the NEH heuristic instead of the normal procedure of the $LR(x)$ algorithm, i.e., the first $3/4n$ jobs of each sequence are scheduled according to the $LR(x)$ procedure and the rest according to the NEH procedure.

- Heuristic RZ (Rajendran and Ziegler, 1997). This heuristic consists of two steps: An initial ordering, and an improvement phase. With respect to the initial ordering, the jobs are sorted in ascending order of the total processing times. The improvement phase (denoted

iRZ in the following) consists of inserting each job in the sequence in the rest of positions updating the sequence when a better solution is found.

- Heuristic $RZ - LW$ (Li and Wu, 2005). This heuristic consists in iteratively performing iRZ until no further improvement is found.

- Heuristic ICi (Li et al., 2009). This is a family of composite heuristics where an initial solution is obtained by using $LR(1)$ and then improved by using different local search methods. If the local search is performed using the iRZ procedure, then the heuristic is denoted $IC1$. Heuristic $IC2$ performs FPE on the solution obtained by $IC1$. Finally, $IC3$ consists of running $IC1$ and then performing a local search denoted as $FPE - R$, which is essentially FPE adding a restart from the first job every time the current solution is improved.

- Heuristic $PRi(x)$ (Pan and Ruiz, 2013). These are several composite heuristics: $PR1(x)$ performs iRZ on each one of the x sequences obtained by heuristic $LR-NEH(x)$. $PR2(x)$ first run the heuristic $LR-NEH(x)$ and then tries to improve this solution using a VNS-like (Variable Neighborhood Search) local search method. This method was introduced by Tasgetiren et al. (2007) and consists in an insertion and interchange variant of the classical VNS where insertion and interchange movements are repeated until no further improvement is found. $PR3(x)$ performs x times a iRZ and two NEH methods after an initial solution obtained by heuristic $LR - NEH(10)$. Finally, $PR4(x)$ replaces the iRZ method of $PR3(x)$ by a VNS local search. In order to bound the computation time of the heuristics, if the CPU time reaches the value of $0.01 \cdot n \cdot m$ seconds, a last loop is performed and the procedure terminates.

The heuristics discussed above constitute the (so-far) set of efficient heuristics for the problem, as found by Pan and Ruiz (2013) in their exhaustive analysis of all existing heuristics for the $Fm|prmu|\sum C_j$ problem with respect to the quality of the solutions and computational requirements. Since there is a clear tradeoff between the solution obtained by one heuristic, and its computation time, the authors were able to depict a Pareto set to place the efficient heuristics for the problem in view of their performance on the well-known Taillard's testbed (see Figure 1). As

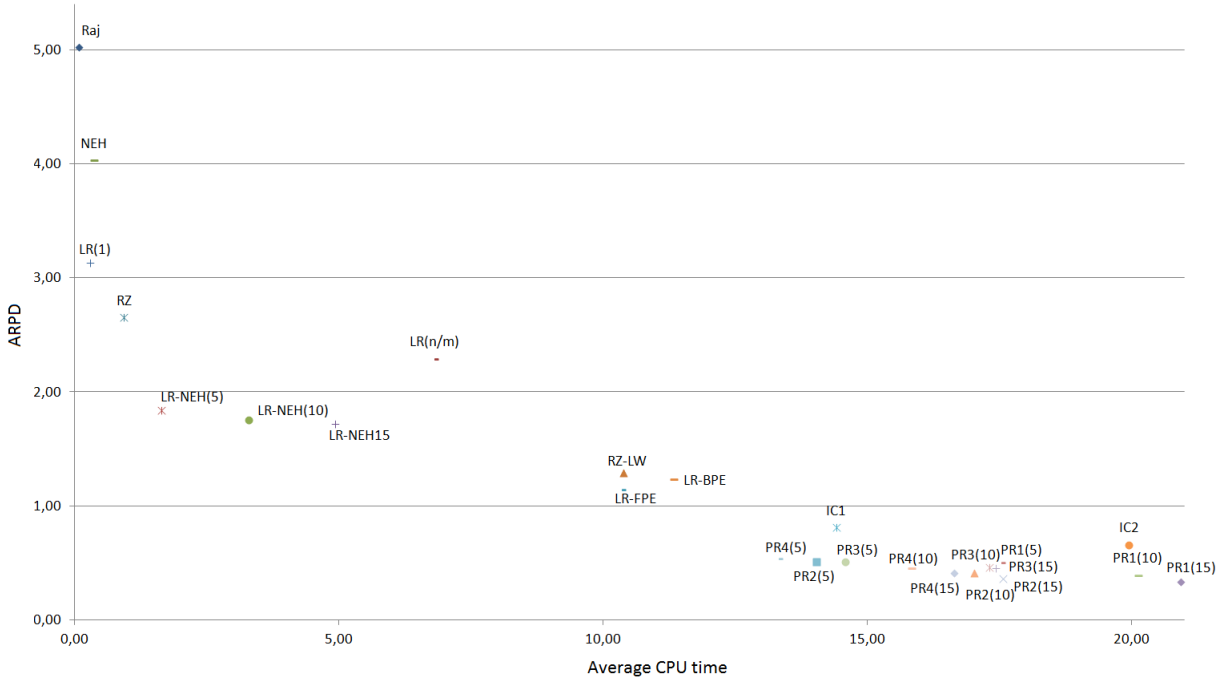

Figure 1: Pareto set using the average computational time (Pan and Ruiz, 2013)

it turns out, this Pareto set is formed by the following heuristics: Raj, $LR(1)$, RZ, $LR{-}NEH(5)$, $LR-NEH(10)$, $LR-NEH(15)$, $LR-FPE$, $PR4(5)$, $PR2(5)$, $PR3(5)$, $PR4(10)$, $PR4(15)$, $PR2(15)$ and $PR1(15)$.

From the analysis of the Pareto set, some conclusions can be derived:

- As it can be seen in Table 1, all the efficient heuristics consist on variation/adaptations of the following five main (or primary) procedures: NEH, $LR(x)$, FPE, iRZ and VNS. More specifically, the $LR(x)$ heuristic is present in 12 of the 14 heuristics in the Pareto set.

- Regarding the complexity of the five primary procedures, NEH is known to be $O(n^3 \cdot m)$, the same complexity as LR. It is easy to check that each iteration of iRZ is $O(n^3 \cdot m)$. Hence, the iRZ has a complexity of $k \cdot n^3 \cdot m$ with k the number of iterations in which there is an improvement in the objective function. The complexity of FPE corresponds to $x \cdot k \cdot n^2 \cdot m$ (for $FPE-R$, the worst case is $O(x \cdot k \cdot n^3 \cdot m)$), with k indicating again the number of iterations with improvement in the objective function. From the complexity of these five procedures, the complexity of the rest of algorithms in the Pareto set can be easily obtained (this information is summarised in Table 1). As it can be seen, each

heuristic in the Pareto set has at least a complexity of $n^3 \cdot m$. Note that the parameter k cannot be nor bounded neither linked to the problem size. However, in the computational experience carried out in Taillard's tested (see Section 5), this value is usually larger than both n and m.

From these conclusions, it can be seen that LR is a key heuristic of complexity $O(n^3 \cdot m)$, playing a role similar to that of the NEH for makespan minimisation. For this latter problem, Taillard (1990) showed that the complexity of the NEH can be reduced from $O(n^3m)$ to $O(n^2m)$ by using an acceleration mechanism, but unfortunately, such mechanism cannot be used to minimize flowtime. The only acceleration proposed is due to Li et al. (2009), who reported savings in the CPU time around 30-50%. Nevertheless, the complexity of the NEH remains the same and thus a way to reduce the complexity of efficient approximate algorithms for flowtime to $O(n^2m)$ has remained elusive.

In this paper (Section 4) we propose a new heuristic to tackle these two issues, as it outperforms $LR(x)$ with a complexity of $O(n^2m)$. Given the high number of efficient heuristics based on LR, we will be able to obtain an improved Pareto set of efficient heuristics for the problem by embedding our proposal into existing heuristics. But first in Section 3 we discuss in detail some shortcomings of the current measures of the efficiency of the heuristics and propose an alternative indicator.

3 An alternative representation of the efficiency of the heuristics

Aside to proposing efficient heuristics for the problem, the paper by Pan and Ruiz (2013) represents an important advance in the evaluation of approximate procedure for combinatorial optimization problems. In most of these problems –including the one under consideration here–, there is a trade-off between the quality of the solutions and the time required by the heuristic to obtain them. Therefore, both aspects should be weighted when selecting one heuristic among the set of heuristics available for the problem. When facing an specific scheduling case, different decision

Algorithm	LR	NEH	iRZ	FPE	VNS	Complexity
Raj		X				$O(n^3 \cdot m)$
RZ			X			$O(n^3 \cdot m)$
LR(x)	X					$O(x \cdot n^3 \cdot m)$
RZ-LW			X			$O(k \cdot n^3 \cdot m)$
LR-NEH(x)	X	X				$O(x \cdot n^3 \cdot m)$
LR(n/m)-FPE(n)	X			X		$O(n^4)$
IC1	X		X			$O(k \cdot n^3 \cdot m)$
IC2	X		X	X		$O(k \cdot n^3 \cdot m)$
IC3	X		X	X (r)		$O(k \cdot n^3 \cdot m)$
PR1(x)	X	X	X			$O(x \cdot k \cdot n^3 \cdot m)$
PR2(x)	X	X			X	$O(x \cdot k \cdot n^3 \cdot m)$
PR3(x)	X	X	X			$O(x \cdot k \cdot n^3 \cdot m)$
PR4(x)	X	X			X	$O(x \cdot k \cdot n^3 \cdot m)$

Table 1: Efficient heuristics (Pan and Ruiz, 2013) as variation/adaptation of primary procedures.

intervals may be required, and different quality of the solution can be accepted. Consequently, in most cases there is no *a priori* knowledge of the precise trade-off required by the Decision Maker. Then, the idea of representing the heuristics along the two important criteria (quality of solutions and computational requirements) and excluding the dominated heuristics allows providing the Decision Maker with the set of Pareto-efficient heuristics so he/she can select the most convenient for his/her specific case.

Note that, for a given heuristic, different measures can be devised both for its quality of the solutions and for its computational requirements. In the paper by Pan and Ruiz (2013), these are measured by the Average Relative Percentage Deviation ($ARPD$) and by the average CPU time in seconds, respectively. The ARPD of heuristic h (out of a total of H heuristics) is obtained by averaging RPD_{ih} the Relative Percentage Deviation of heuristic h in instance i over all instances of Taillard's testbed Taillard (1993):

$$RPD_{ih} = \frac{C_{sum}^{ih} - \min_{1 \leq h \leq H} C_{sum}^{ih}}{\min_{1 \leq h \leq H} C_{sum}^{ih}} \cdot 100 \tag{2}$$

where C_{sum}^{ih} is the flowtime obtained by heuristic h when applied to instance i. Similarly, the average CPU time is obtained by averaging the CPU times required by heuristic h for all instances in Taillard's testbed.

Despite the aforementioned advance in the determination of efficient solutions, using these two measures in the Pareto set presents a number of issues:

- $ARPD$ is an dimensionless indicator that is normalised with respect to the best result obtained for each instance, therefore the influence of the instance (and thus the instance size) is somewhat smoothed. In contrast, CPU times are heavily instance and instance-size dependent. Moreover, given the problem sizes of Taillard's testbed, average CPU times of a heuristic are heavily compromised by the CPU times obtained for the biggest 10 instances (those of size 500×20).

  To illustrate this shortcoming, let us consider the NEH heuristic. This heuristic is known to have a complexity of $O(n^3 \cdot m)$ for the problem under consideration, therefore for the smallest problem size of Taillard's testbed (20×5) its complexity is $O(10^2)$, whereas it is $O(1.6 \cdot 10^8)$ for size 200×20 and $O(2.5 \cdot 10^9)$ for size 500×20. This enormous differences in computation times imply that, using the CPU time data in Pan and Ruiz (2013), the average CPU time for the last 20 instances of Taillard's testbed is 0.36 seconds, while the average for all 120 instances is 0.37 seconds. As a consequence, more than 80% of the testbed (the first 100 instances out of a total of 120) contributes with 0.01 seconds (less than 5%) to the indicator.

- Besides, Taillard's testbed is not orthogonal with respect to the number of machines and the number of jobs. More specifically, n ranges from 20 to 500, and m ranges from 5 to 20. Therefore, the CPU times required for one (hypothetical) heuristic with complexity $O(n^3m)$ would grow in this testbed much faster than that of another (hypothetical) heuristic with complexity $O(n^2m^2)$. This could compromise the average results, thus masking the efficiency –or inefficiency– of some heuristics.

  An extreme case of the above problem can be exemplified with heuristics $PR4(10)$ and $PR4(15)$. According to the data in Table 3 of the paper by Pan and Ruiz (2013), these heuristics are found to be efficient when averaged over all the 120 instances of Taillard's testbed. However, when the results are disaggregated for each problem size, then it turns out that they are not efficient for any size of the testbed. Something similar happens for

heuristics PR2(15) and PR4(5), which are efficient in terms of the aggregate results, but are only efficient for one instance size (instances 100×10 and 500×20 respectively), being this fact explained by the stopping criterion employed in this heuristic ($0.01 \cdot n \cdot m$ seconds at maximum), a criterion only reached for the three biggest sizes (100 x 10, 200 x 20 and 500 x 20) thus reducing heavily the total average computational time of these heuristics.

In order to overcome these shortcomings, we propose an alternative measure to evaluate the efficiency of heuristics. More specifically, we propose replacing the CPU time as indicator of the computational requirements of heuristic h by the Average Relative Percentage computation Time ($ARPT_h$). $ARPT_h$ is defined as follows:

$$ARPT_h = \sum_{i=1}^{I} \frac{RPT_{ih}}{I}$$

where

$$RPT_{ih} = \frac{T_{ih} - ACT_i}{ACT_i}$$

and

$$ACT_i = \sum_{h=1}^{H} T_{ih}/H$$

where RPT_{ih} is the relative percentage computation time obtained by heuristic h for instance i, T_{ih} is the computation time of heuristic h when appplied to instance i, H is the number heuristics considered, I the number of instances of the testbed, and ACT_i is the average (among all heuristics considered) computational times for the instance i.

We can employ the data from Pan and Ruiz (2013) to calculate the corresponding values of $ARPT$ for each heuristic. The results are shown in Table 2, and are represented in two axis in Figure 2. The set of efficient heuristics according to the proposed approach is: Raj, $LR(1)$, RZ, $RZ-LW$, $LR-NEH(5)$, $LR-NEH(10)$, $LR-FPE$, $IC1$, $IC2$, $IC3$, $PR1(5)$, $PR1(10)$ and $PR1(15)$.

It can be checked in Table 2 that the alternative representation of efficiency is more complete in the sense that ten heuristics of the thirteen heuristics considered efficient using $ARPT$ are indeed efficient for six or more problem sizes, whereas only three heuristics with less than six

Algorithm	*ARPD*	*ARPT*	#Efficient Size Average CPU times	#Efficient Size *ARPT*
Raj	5.02	-1.00	**7**	**7**
LIT	8.26	-0.96	0	0
SPD1	17.37	-0.97	0	0
SPD2	16.56	-0.97	1	1
RZ	2.65	-0.97	**3**	**3**
WY	2.83	-0.67	0	0
LR(1)	3.13	-0.99	**7**	**7**
LR(n/m)	2.29	-0.89	2	2
LR(n)	2.09	0.27	0	0
NEH	4.03	-0.99	1	1
FL	1.99	-0.41	0	0
RZ-LW	1.29	-0.82	4	**4**
FL-LS	1.22	0.11	0	0
LR-NEH(5)	1.84	-0.94	**8**	**8**
LR-NEH(10)	1.75	-0.90	**6**	**6**
LR-NEH(15)	1.72	-0.78	**3**	3
LR-FPE	1.14	-0.81	**7**	**7**
LR-BPE	1.23	-0.80	5	5
IH7	1.43	-0.25	0	0
IH7-FL	1.30	-0.22	0	0
C1-FL	1.72	-0.35	0	0
C2-FL	0.95	0.26	1	1
IC1	0.81	-0.75	6	**6**
IC2	0.66	-0.62	8	**8**
IC3	0.62	-0.26	6	**6**
PR1(5)	0.50	-0.15	7	**7**
PR1(10)	0.39	0.79	4	**4**
PR1(15)	0.33	1.43	**6**	**6**
PR2(5)	0.51	0.54	**3**	3
PR2(10)	0.41	1.90	3	3
PR2(15)	0.36	2.93	**1**	1
PR3(5)	0.51	0.04	**4**	4
PR3(10)	0.46	0.91	2	2
PR3(15)	0.45	1.64	1	1
PR4(5)	0.54	0.69	**1**	1
PR4(10)	0.45	2.00	**0**	0
PR4(15)	0.41	2.98	**0**	0

Table 2: Summary of average results of the heuristics implemented in Pan and Ruiz (2013) using *ARPT*. Last two columns show the number of problem sizes where each heuristic is efficient using both average CPU times and *ARPT*. In bold it is indicated when the heuristic is efficient when averaged for the 120-instances using either CPU time or *ARPT*.

efficient sizes are included. Furthermore, the data in Pan and Ruiz (2013) expressed the CPU time with two decimals, therefore for some heuristics the CPU time is 0.00 in some problem sizes, and in this case it is not possible to establish a realistic trade-off between CPU time and $ARPD$. More specifically, heuristics RZ and $RZ - LW$ should have several more efficient sizes due to the fact that their CPU time is 0.00 for the first 3-5 instance sizes (this may also happen with as $IC1$ or $IC2$, among others). It is worth noting that, using CPU time, six heuristics globally efficient are efficient for six or more problem sizes while there are eight heuristics which have less than six sizes for which they are efficient. The average number of efficient sizes for the fourteen efficient heuristics using CPU time is only 4.00, as compared to an average of 6.08 using $ARPT$.

Furthermore, in order to re-assure that no heuristic is excluded by using the proposed indicator, we conduct a series of experiments to extend the comparison between heuristics $PR1$ and $PR2$. Note that, according to the results, $PR1$ seems to outperform $PR2$, but the latter is extremely efficient for the biggest instances (i.e. $100 \times 10, 200 \times 20$, and 500×20). In addition, the improvement phase of VNS used in Pan and Ruiz (2013) (which includes pairwise interchanges and insertion movements in an single position) is used as the first neighborhood. This may affect the performance of the PR2 heuristic, as more exhaustive insertion movements (such as iRZ) could be also considered as the first neighborhood so the performance of this so-obtained heuristic (labelled $PR2A$ in the following) is improved. Note however, that $PR2A$ would be much slower than $PR1$ and $PR2$.

Thus, these three heuristics ($PR1$, $PR2$ and $PR2A$) are further compared using Taillard's testbed. To obtain more points in the Pareto approximation, we extend the initial range of the stopping criteria and that of parameter x for the fastest heuristics (i.e. $PR1$ and $PR2$). More specifically, we test the following stopping criteria: $0.01 \cdot n \cdot m$, $0.05 \cdot n \cdot m$, and $0.1 \cdot n \cdot m$ for all three heuristics, and also $0.2 \cdot n \cdot m$ for $PR1$. Regarding the values of x, $x \in \{5, 10, 15, 20, 25\}$ is used for $PR1(x)$, $x \in \{5, 10, 15, 20\}$ is employed for $PR2(x)$, whereas $x \in \{5, 10, 15\}$ is used for $PR2A(x)$. A clear dominance of $PR1(x)$ over $PR2(x)$ and $PR2A(x)$ is obtained from these results (summarised in Figure 3, where the dotted lines represent quadratic polynomial trend lines for the heuristics). As a result, in the computational experiments in the following sections, $PR2(x)$ and $PR2A(x)$ will be excluded.

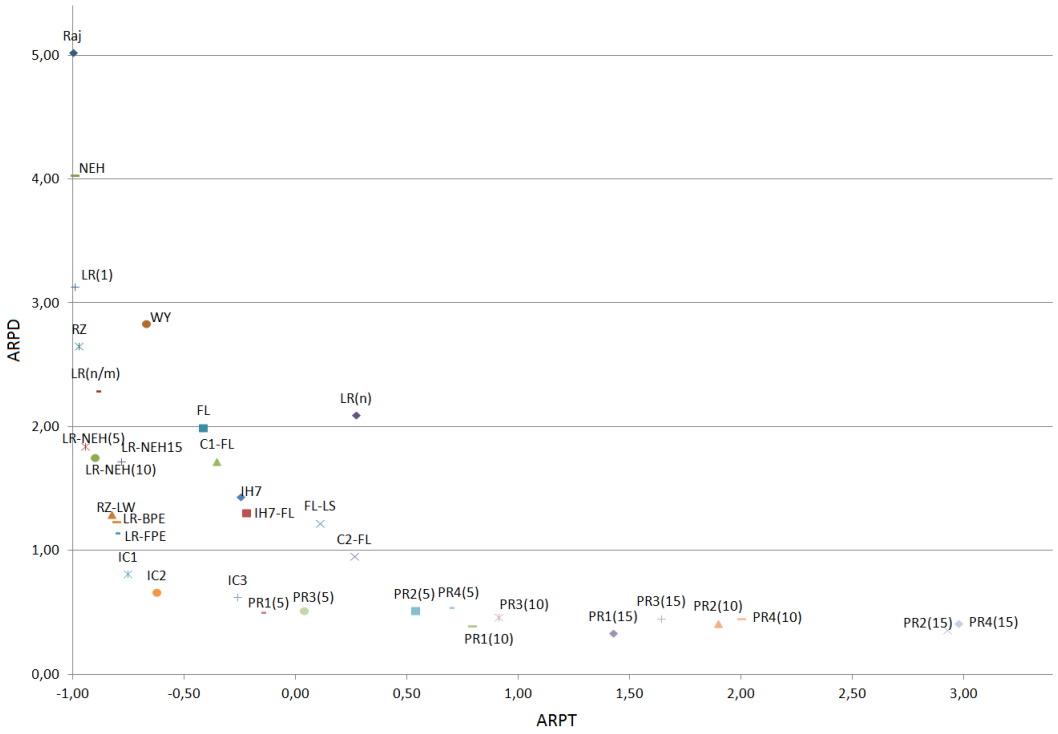

Figure 2: Pareto set using the $ARPT$

In view of the discussion and the results in this section, it seems clear that the evaluation of the performance of heuristics for the problem is not trivial, and that both the quality of the solutions and the computational effort should be taken into account. Building upon the work by Pan and Ruiz (2013), an indicator for measuring the computational effort has been proposed. This indicator, although not perfect, presents more consistency between the disaggregated (i.e. at instance size level) and aggregated (overall) results. Nevertheless, since the state-of-art evaluation of heuristics for flowtime (that of Pan and Ruiz, 2013) was done using CPU time as indicator, we report the subsequent results in this paper using also their scheme.

4 The proposed heuristic

The proposed heuristic –denoted in the following as $FF(x)$– uses the idea present in LR of decreasing number of the evaluations of solutions, beginning with $n-1$ evaluations and finishing with 0. Thereby, the heuristic is composed of $n-1$ step with a maximum of $n-1$ evaluations. However, in contrast to the LR, we focus in the evaluation of each solution trying to reduce the

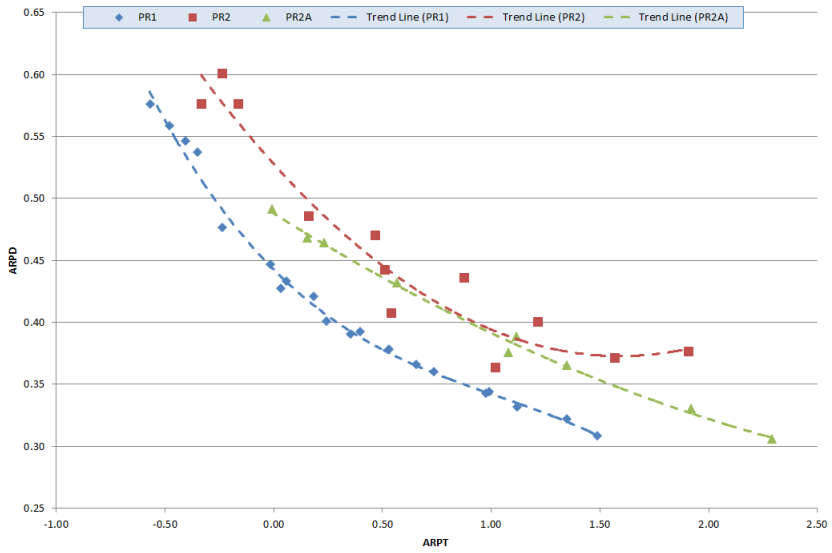

Figure 3: Comparison heuristics $PR1$, $PR2$ and $PR2A$

complexity of the algorithm. When introducing a new job at the end of the sequence, there are three elements to be considered:

- *Idle time induced by the newly inserted job.* This idle time influences the next jobs to be inserted. Clearly, this influence decreases with each step (being 0 in the last step). Its calculation has a complexity $O(m)$ since only the completion time of the preceding job in each machine is required. For a given iteration k, this data is known from the previous iteration (or zero if it is the first job).

- *Completion time in machine m of the newly inserted job.* Its influence in the total flowtime is clear since the completion time of each job in machine m is included in the objective function. This data can be calculated within $O(m)$ using the completion time on each machine of the preceding job.

- *Completion time in machine m of the artificial job.* It seems that it influences the objective function in an indirect manner, as it is an indicator of the completion time in machine m of the yet unscheduled jobs. It is thus convenient to ensure that the unscheduled jobs will not have a very large completion time in machine m. However, the calculation of this completion time has a complexity of $n \cdot m$.

As in the LR heuristic, we intend that, once a job is scheduled in a position, it stays in this position, then choosing the adequate position of a job is critical. The problem thus lies in weighting the influence of the aforementioned elements. To do so, we use two parameters (a and b), to balance the first two elements, i.e. idle time and completion time of the newly inserted job. In contrast, we leave aside the third element (completion time of the artificial job), since its influence in the objective function is not as direct as the other two elements, and its consideration would increase the complexity of the algorithm to $n^3 \cdot m$.

More specifically, the proposed heuristic is as follows:

1. Sort the jobs according to a non descending order of indicator ξ'_{j0} (see equation 3), breaking ties in favor of jobs with lower $IT'_{j,0}$ (see equation 4). Let us denote by I the so-obtained vector

2. Obtain x partial sequences π^i $(i = 1, \ldots, x)$ of length 1, where the first (and only) job of sequence π^i is the job in position i in I. Store in U^i the jobs not scheduled in π^i.

3. For $k = 1$ to $n - 1$:

   (a) For each partial sequence π^i, remove from U^i the job for which the minimum value of $\xi'_{j,k}$ (see equation 3) is found and place it in the last position of π^i.

4. Return the (final) sequence π^i yielding the lowest completion time.

Therefore, the proposed procedure begins with x sequences (π^i with $i \in [1, x]$) with only one job. The first job of each sequence π^i is the job in position i of a vector sorted in non descending order of indicator ξ'_{j0} (equation 3) breaking ties in favor of jobs with higher $IT'_{j,0}$ (equation 4) and each final sequence π^i is obtained adding one by one jobs to the last position of the vector.

Let us denote by k the size of the vector in each step until the vector reaches the n jobs. To insert a new job j ($j \in U^i$) in each sequence π^i, one of the unscheduled jobs of each sequence, U^i, is removed according to an ascending index of a complexity m, $\xi'_{j,k}$. This index is based on $IT'_{j,k}$ the weighted idle time between the job in position k and the new job j to be inserted, and on the makespan of the sequence when inserting job j, $C_{m,j}$. For each job $j \in U^i$, $\xi'_{j,k}$ is calculated as follows:

Notation using LR	Notation using FF
LR(1)	FF(1)
LR-NEH(5)	FF-NEH(5)
LR-NEH(10)	FF-NEH(10)
LR(n/m)-FPE(n)	FF(n/m)-FPE(n)
IC1	FF-IC1
IC2	FF-IC2
IC3	FF-IC3
PR1(5)	FF-PR1(5)
PR1(10)	FF-PR1(10)
PR1(15)	FF-PR1(15)

Table 3: Notation for the heuristics using the proposed heuristic FF

$$\xi'_{j,k} = \frac{(n-k-2)}{a} \cdot IT'_{j,k} + AT'_{j,k} \tag{3}$$

where $AT'_{j,k}$ and $IT'_{j,k}$ are defined as follow:

$$IT'_{j,k} = \sum_{i=2}^{m} \frac{m \cdot max\{C_{i-1,j} - C_{i,[k]}, 0\}}{i - b + k \cdot (m - i + b)/(n-2)} \tag{4}$$

$$AT'_{j,k} = C_{m,j} \tag{5}$$

being a and b the aforementioned parameters to balance the influence of idle times and completion time of the newly inserted job. Note that by avoiding the calculation of the completion time of the artificial job p (C_{mp}), the complexity of the algorithm decreases from $n^3 \cdot m$ to $n^2 \cdot m$, a complexity n times lower than the fastest heuristics in the efficient set by Pan and Ruiz (2013).

As explained in Section 3, 10 of the 13 efficient heuristics using ARPT are based on LR. All of these 10 heuristics can be reimplemented using FF instead of LR. The notation for this set of heuristics is shown in Table 3. Additionally, due to the decrease in complexity, $FF(x)$ can be implemented for larger values of x. Note that $LR(n/m) - FPE(n)$ has a greater complexity than $LR(n/m)$, i.e. $O(n^4)$. Once LR is replaced by FF, $FF(n/m) - FPE(n)$ has a lower complexity and it can be interesting to perform the heuristic $FF(x) - FPE(y)$ for more values of both x and y since e.g. now $FF(1) - FPE(1)$ is also $O(n^2 \cdot m)$.

Source	Sum of Squares	Df	Mean Square	F-Ratio	p-Value
Main Effects					
n	52.169	4	13.042	20.755	0.000
m	26.724	2	13.362	21.264	0.000
a	7.711	3	2.570	4.090	0.007
b	10.763	2	5.381	8.564	0.000
Interaction					
$a * b$	0.114	6	0.019	0.030	1.000
$m * a$	6.087	6	1.015	1.614	0.139
$n * a$	6.063	12	0.505	0.801	0.647
$m * b$	5.548	4	1.387	2.207	0.066
$n * b$	13.679	8	1.710	2.721	0.006
$n * m$	110.808	8	13.851	22.042	0.000
Residual	1095.905	1744			
Total (corrected)	1335.571	1799			

Table 4: ANOVA for the parameters n, m, a, b

Prior to conducting these experiments, the best values for parameters a and b have to be found. To do so we carry out some computational experiments where different values are tried. After a first screening where different ranges of values were discarded, a multi-factor Analysis of Variance (ANOVA) was performed with four factors (n, m, a and b) where $a \in \{1, 2, 3, 4\}$, $b \in \{0, 0.5, 1\}$. Factors n and m are those in the testbed by Taillard (Taillard, 1993), which is employed to perform the analysis. More specifically, $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$. The results in Table 4 show that all parameters n, m, a and b are statistically significant. To determine the best level for each parameter, a Least Significant Difference (LSD) interval for each one is carried out (see Figure 4). Although from this figure it may seem that there is a monotonic trend for both a and b, further tests with $a > 4$ and $b > 1$ did not produce better results. Therefore, $a = 4$ and $b = 1$ were used for the new set of heuristics $FF, FF-FPE, FF-NEH, FF-ICx, FF-PR1(x)$ in the next section.

5 Computational experience

In order to compare the performance of the heuristics proposed, the efficient heuristics described in Section 3 are implemented and their results on the benchmark set of Taillard (1993) with 120 instances are collected. As mentioned before, Li et al. (2009) showed that heuristics for the

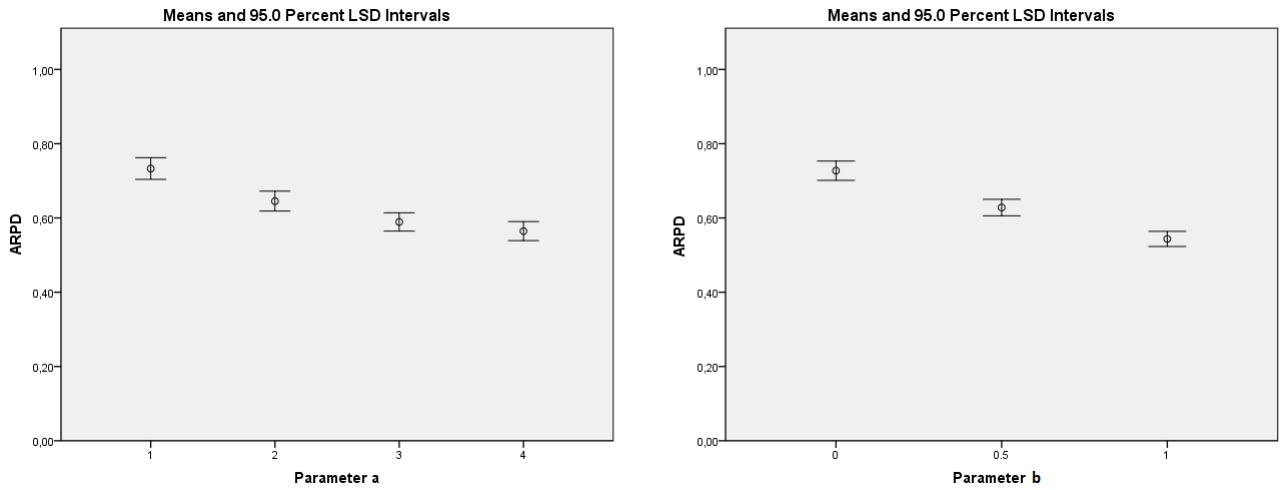

Figure 4: LSD intervals of the RPD for each level of the parameters a and b.

$Fm|prmu|\sum C_j$ with insertion and pair-wise exchanges (i.e. all efficient heuristic but LR) can be implemented reducing around 30-50% computational times. Thus, in order to conduct a fair comparison, this acceleration has been implemented in our codification of each insertion method of all heuristics.

Comparing the CPU time required by each heuristic with those in the paper by Pan and Ruiz (2013), we found that the former were larger by a factor of 3.38 times on average. This can be explained due to the different programming languages used to implement the heuristics, to the different ways of coding the routines and to the different computer employed for the implementation. However, since the stopping criterion of some heuristics in Pan and Ruiz (2013) was set to $0.01 \cdot n \cdot m$ seconds, applying the same criterion in our slower procedures would penalise the performance of these heuristics, as they now require more time per iteration and thus will perform less iterations. Therefore, in order to conduct a fair comparison, we change the stopping criterion to $0.0338 \cdot n \cdot m$ so to have a similar number of iterations to that of Pan and Ruiz (2013).

The overall results of the experiments are summarised in Table 5, where the average results of each heuristic over all 120 instances are shown. The effect of replacing LR by FF is specifically highlighted in Table 6, showing the efficiency of the proposed heuristics. For instance, the average computational time of $FF(1)$ is just $0.02s$ while the average computational time for $LR(1)$ is $0.76s$. Not only the complexity of the algorithm has been reduced from $n^3 \cdot m$ to $n^2 \cdot m$, but the $ARPD$ of $FF(1)$ is also lower as compared to the $ARPD$ of $LR(1)$.

Heuristic	*ARPD*	*ARPT*	Average CPU times
LR(1)	3.01	-0.98	0.76
LR(n/m)-FPE(n)	1.02	-0.47	33.07
IC1	0.64	-0.29	41.93
IC2	0.54	-0.08	55.33
IC3	0.53	1.26	330.92
LR-NEH(5)	1.52	-0.75	6.69
LR-NEH(10)	1.44	-0.52	13.37
Raj	4.86	-0.99	0.29
RZ	2.32	-0.90	2.97
RZ-LW	1.13	-0.42	32.69
PR1(5)	0.37	1.93	58.87
PR1(10)	0.26	4.60	67.38
PR1(15)	0.21	7.06	68.54
FF(1)	2.76	-1.00	0.02
FF(2)	2.34	-0.99	0.05
FF(n/10)	1.95	-0.98	0.99
FF(n/m)	2.05	-0.98	0.54
FF(n)	1.83	-0.69	10.12
FF(1)-FPE(1)	2.36	-0.99	0.15
FF(1)-FPE(n/10)	1.81	-0.94	2.89
FF(1)-FPE(n)	1.23	-0.64	21.39
FF(2)-FPE(1)	1.97	-0.97	0.17
FF(2)-FPE(n/10)	1.55	-0.94	3.05
FF(2)-FPE(n)	1.07	-0.63	19.78
FF(15)-FPE(1)	1.58	-0.88	0.44
FF(15)-FPE(n/10)	1.29	-0.86	2.90
FF(15)-FPE(n)	0.96	-0.62	16.89
FF(n/10)-FPE(1)	1.63	-0.96	1.12
FF(n/10)-FPE(n/10)	1.33	-0.92	3.51
FF(n/10)-FPE(n)	0.94	-0.63	18.79
FF(n/m)-FPE(1)	1.70	-0.96	0.64
FF(n/m)-FPE(n/10)	1.37	-0.92	3.11
FF(n/m)-FPE(n)	1.01	-0.64	18.05
FF(n)-FPE(1)	1.53	-0.65	10.27
FF(n)-FPE(n/10)	1.25	-0.62	12.68
FF(n)-FPE(n)	0.94	-0.35	28.00
FF-IC1	0.62	-0.48	25.33
FF-IC2	0.56	-0.26	36.47
FF-IC3	0.55	1.13	300.93
FF-NEH(5)	1.40	-0.86	3.18
FF-NEH(10)	1.34	-0.72	6.33
FF-PR1(5)	0.34	1.37	48.60
FF-PR1(10)	0.24	3.68	58.48
FF-PR1(15)	0.19	5.45	63.03

Table 5: Summary of results of heuristics

Heuristic	*ARPD*	Avg. Time	*ARPT*		Heuristic	*ARPD*	Avg. Time	*ARPT*
LR(1)	3.01	0.76	-0.98	→	FF(1)	2.76	0.02	-1.00
LR(n/m)-FPE(n)	1.02	33.07	-0.47	→	FF(n/m)-FPE(n)	1.01	18.05	-0.64
IC1	0.64	41.93	-0.29	→	FF-IC1	0.62	25.33	-0.48
IC2	0.54	55.33	-0.08	→	FF-IC2	0.56	36.47	-0.26
IC3	0.53	330.92	1.26	→	FF-IC3	0.55	300.93	1.13
LR-NEH(5)	1.52	6.69	-0.75	→	FF-NEH(5)	1.40	3.18	-0.86
LR-NEH(10)	1.44	13.37	-0.52	→	FF-NEH(10)	1.34	6.33	-0.72
Raj	4.86	0.29	-0.99		—	—	—	—
RZ	2.32	2.97	-0.90		—	—	—	—
RZ-LW	1.13	32.69	-0.42		—	—	—	—
PR1(5)	0.37	58.87	1.93	→	FF-PR1(5)	0.34	48.60	1.37
PR1(10)	0.26	67.38	4.60	→	FF-PR1(10)	0.24	58.48	3.68
PR1(15)	0.21	68.54	7.06	→	FF-PR1(15)	0.19	63.03	5.45

Table 6: Comparisons between composite heuristics which include LR and FF heuristics

Regarding the detailed results, those obtained by the heuristics Raj, $LR(1)$, RZ, $RZ-LW$, $LR-NEH(5)$, $LR-NEH(10)$, $LR-FPE$, $IC1$, $IC2$ and $IC3$ are shown in Table 7 while the CPU times are shown in Table 10.

Instance	LR(1)	LR(n/m)-FPE(n)	IC1	IC2	IC3	LR-NEH(5)	LR-NEH(10)	Raj	RZ	RZ-LW	PR1(5)	PR1(10)	PR1(15)	FF(1)	FF(2)	FF(n/10)	FF(n/m)	FF(n)
20 x 5	2.70	1.47	0.72	0.67	0.66	1.93	1.93	5.57	2.39	1.05	0.30	0.22	0.08	3.13	2.58	2.58	2.32	2.30
20 x 10	3.14	1.42	1.02	0.84	0.84	2.07	2.07	4.67	2.24	0.88	0.56	0.24	0.20	3.33	3.18	3.18	3.18	2.89
20 x 20	3.12	1.69	1.08	1.05	1.08	2.00	1.96	4.60	1.72	0.81	0.45	0.26	0.03	3.31	2.74	2.74	3.31	2.54
50 x 5	2.17	0.79	0.50	0.41	0.41	1.08	1.07	4.47	2.15	1.22	0.26	0.24	0.22	1.81	1.50	1.43	1.42	1.42
50 x 10	5.21	1.06	0.80	0.74	0.84	1.94	1.93	5.64	2.78	1.41	0.28	0.16	0.16	3.02	2.52	2.04	2.04	2.04
50 x 20	3.65	1.65	0.82	0.54	0.54	2.22	2.02	5.66	2.23	1.59	0.40	0.26	0.14	2.86	2.84	2.55	2.84	2.55
100 x 5	1.15	0.31	0.27	0.25	0.25	0.68	0.63	4.21	2.05	1.34	0.28	0.23	0.22	1.03	0.61	0.54	0.54	0.54
100 x 10	2.64	0.65	0.47	0.28	0.25	1.19	1.09	4.77	2.56	1.05	0.32	0.24	0.17	2.31	2.08	1.72	1.72	1.67
100 x 20	4.42	1.42	0.85	0.75	0.63	2.24	1.98	5.48	2.53	0.96	0.48	0.27	0.27	4.66	3.96	2.79	3.14	2.39
200 x 10	2.42	0.39	0.30	0.24	0.23	0.68	0.64	4.26	2.68	1.16	0.17	0.14	0.14	1.89	1.46	1.18	1.18	1.18
200 x 20	3.75	1.02	0.49	0.41	0.31	1.48	1.37	4.69	2.37	1.21	0.39	0.34	0.34	3.87	3.18	1.87	2.06	1.64
500 x 20	1.77	0.43	0.37	0.29	0.26	0.74	0.64	4.24	2.16	0.87	0.55	0.55	0.55	1.90	1.48	0.79	0.81	0.79
ARPD	3.01	1.02	0.64	0.54	0.53	1.52	1.44	4.86	2.32	1.13	0.37	0.26	0.21	2.76	2.34	1.95	2.05	1.83

Table 7: RPD of heuristics I

	FF(x)-FPE(y)																	
	1-1	1-n/10	1-n	2-1	2 - n/10	2 - n	15-1	15-n/10	15-n	n/10-1	n/10-n/10	n/10-n	n/m-1	n/m-n/10	n/m-n	n-1	n-n/10	n-n
20 x 5	2.01	1.80	1.19	1.81	1.76	1.11	1.44	1.40	1.24	1.81	1.76	1.11	1.48	1.44	1.28	1.44	1.40	1.24
20 x 10	2.61	2.22	1.65	2.29	2.18	1.56	1.90	1.75	1.48	2.29	2.18	1.56	2.29	2.18	1.56	1.90	1.75	1.48
20 x 20	2.74	2.48	1.84	2.19	1.97	1.60	2.25	2.30	1.65	2.19	1.97	1.60	2.74	2.48	1.84	2.25	2.30	1.65
50 x 5	1.65	1.20	0.80	1.33	0.92	0.66	1.27	0.96	0.77	1.31	1.03	0.79	1.27	0.96	0.77	1.27	0.96	0.77
50 x 10	2.47	1.99	1.57	2.18	1.74	1.40	1.81	1.57	1.18	1.77	1.56	1.18	1.77	1.56	1.18	1.81	1.57	1.18
50 x 20	2.56	2.31	1.57	2.41	2.17	1.56	2.24	1.88	1.36	2.24	1.88	1.36	2.41	2.17	1.56	2.24	1.88	1.36
100 x 5	0.96	0.86	0.47	0.57	0.41	0.21	0.50	0.41	0.25	0.50	0.41	0.25	0.50	0.41	0.25	0.50	0.41	0.25
100 x 10	2.05	1.27	0.98	1.81	1.15	0.86	1.55	0.95	0.77	1.55	0.98	0.80	1.55	0.98	0.80	1.52	0.97	0.76
100 x 20	4.18	3.23	1.82	3.45	2.76	1.45	2.40	1.87	1.15	2.50	2.02	1.15	2.72	2.03	1.28	2.15	1.72	1.20
200 x 10	1.70	0.88	0.62	1.34	0.65	0.43	1.10	0.61	0.48	1.08	0.59	0.49	1.08	0.59	0.49	1.08	0.59	0.49
200 x 20	3.55	2.41	1.40	2.84	2.07	1.38	1.64	1.18	0.80	1.64	1.18	0.80	1.83	1.26	0.84	1.47	0.98	0.67
500 x 20	1.80	1.10	0.79	1.40	0.88	0.58	0.91	0.58	0.34	0.72	0.43	0.23	0.74	0.42	0.23	0.72	0.43	0.23
ARPD	2.36	1.81	1.23	1.97	1.55	1.07	1.58	1.29	0.96	1.63	1.33	0.94	1.70	1.37	1.01	1.53	1.25	0.94

Table 8: RPD of heuristics II

	FF-ICH1	FF-ICH2	FF-ICH3	FF-NEH(5)	FF-NEH(10)	FF-PR1(5)	FF-PR1(10)	FF-PR1(15)
20 x 5	1.16	1.07	1.07	1.54	1.54	0.33	0.20	0.13
20 x 10	0.87	0.97	0.97	2.22	2.22	0.47	0.34	0.10
20 x 20	1.08	1.08	1.07	2.03	1.95	0.30	0.18	0.12
50 x 5	0.46	0.32	0.32	1.11	1.11	0.37	0.29	0.29
50 x 10	0.52	0.62	0.60	1.53	1.53	0.27	0.21	0.14
50 x 20	0.91	0.86	0.86	1.80	1.80	0.27	0.22	0.12
100 x 5	0.26	0.12	0.11	0.63	0.61	0.20	0.17	0.14
100 x 10	0.60	0.36	0.40	1.21	1.07	0.26	0.16	0.14
100 x 20	0.60	0.56	0.54	1.95	1.72	0.43	0.24	0.17
200 x 10	0.39	0.26	0.23	0.92	0.84	0.30	0.18	0.18
200 x 20	0.46	0.39	0.34	1.25	1.11	0.28	0.19	0.19
500 x 20	0.14	0.08	0.04	0.66	0.61	0.55	0.55	0.51
ARPD	0.62	0.56	0.55	1.40	1.34	0.34	0.24	0.19

Table 9: RPD of heuristics III

Instance	LR(1)	LR(n/m)-FPE(n)	IC1	IC2	IC3	LR-NEH(5)	LR-NEH(10)	Raj	RZ	RZ-LW	PR1(5)	PR1(10)	PR1(15)	FF(1)	FF(2)	FF(n/10)	FF(n/m)	FF(n)
20 x 5	0.00	0.00	0.01	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.02	0.04	0.06	0.00	0.00	0.00	0.00	0.00
20 x 10	0.00	0.00	0.00	0.01	0.01	0.00	0.01	0.00	0.00	0.01	0.03	0.07	0.12	0.00	0.00	0.00	0.00	0.00
20 x 20	0.00	0.01	0.01	0.02	0.02	0.01	0.02	0.00	0.00	0.01	0.06	0.13	0.20	0.00	0.00	0.00	0.00	0.01
50 x 5	0.00	0.05	0.07	0.09	0.11	0.03	0.05	0.00	0.01	0.06	0.29	0.57	0.89	0.00	0.00	0.00	0.01	0.03
50 x 10	0.01	0.08	0.10	0.15	0.19	0.05	0.09	0.00	0.02	0.09	0.61	1.24	1.82	0.00	0.00	0.01	0.01	0.08
50 x 20	0.01	0.12	0.19	0.28	0.37	0.09	0.18	0.00	0.04	0.16	1.09	2.25	3.37	0.00	0.00	0.01	0.00	0.12
100 x 5	0.02	0.56	0.70	0.82	1.13	0.17	0.34	0.01	0.08	0.44	2.19	4.48	7.00	0.00	0.00	0.03	0.05	0.24
100 x 10	0.04	0.86	1.10	1.55	2.75	0.32	0.64	0.02	0.14	0.96	5.32	11.08	17.01	0.01	0.01	0.06	0.05	0.55
100 x 20	0.08	1.48	2.22	2.89	6.01	0.65	1.28	0.03	0.28	1.65	10.99	23.11	35.28	0.01	0.02	0.13	0.06	1.16
200 x 10	0.27	10.10	11.65	14.73	35.94	2.47	4.94	0.12	1.11	8.89	44.48	72.83	71.87	0.02	0.03	0.34	0.34	3.32
200 x 20	0.58	14.77	19.27	25.48	101.91	5.04	10.01	0.23	2.19	15.89	102.84	148.47	147.15	0.03	0.07	0.68	0.35	6.51
500 x 20	8.11	368.83	467.78	617.92	3822.54	71.43	142.83	3.07	31.76	364.16	538.48	544.33	537.68	0.22	0.44	10.69	5.62	109.42
Average	0.76	33.07	41.93	55.33	330.92	6.69	13.37	0.29	2.97	32.69	58.87	67.38	68.54	0.02	0.05	0.99	0.54	10.12
ARPT	-0.98	-0.47	-0.29	-0.08	1.26	-0.75	-0.52	-0.99	-0.90	-0.42	1.93	4.60	7.06	-1.00	-0.99	-0.98	-0.98	-0.69

Table 10: Computational Times of heuristics I

	FF(x)-FPE(y)																	
	1-1	1-n/10	1-n	2-1	2 - n/10	2 - n	15-1	15-n/10	15-n	n/10-1	n/10-n/10	n/10-n	n/m-1	n/m-n/10	n/m-n	n-1	n-n/10	n-n
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	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	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01
20 x 20	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.01	0.01
50 x 5	0.00	0.01	0.03	0.00	0.01	0.03	0.01	0.01	0.03	0.00	0.01	0.03	0.01	0.01	0.03	0.04	0.04	0.05
50 x 10	0.00	0.01	0.06	0.01	0.01	0.06	0.02	0.02	0.06	0.01	0.02	0.06	0.01	0.02	0.06	0.07	0.07	0.12
50 x 20	0.01	0.02	0.14	0.01	0.02	0.12	0.04	0.05	0.14	0.02	0.03	0.13	0.01	0.02	0.12	0.14	0.15	0.24
100 x 5	0.01	0.04	0.22	0.01	0.04	0.26	0.04	0.06	0.25	0.03	0.06	0.30	0.06	0.09	0.32	0.27	0.29	0.51
100 x 10	0.02	0.10	0.58	0.03	0.12	0.67	0.08	0.14	0.52	0.08	0.15	0.74	0.07	0.15	0.69	0.54	0.62	1.16
100 x 20	0.04	0.23	1.26	0.06	0.24	1.35	0.15	0.27	1.07	0.13	0.28	1.47	0.08	0.27	1.41	1.11	1.22	2.00
200 x 10	0.10	1.01	5.32	0.12	0.99	6.21	0.32	1.02	4.39	0.48	1.37	5.84	0.50	1.42	6.03	4.59	5.23	9.77
200 x 20	0.20	2.16	14.15	0.29	2.21	13.39	0.63	2.00	10.29	0.94	2.57	13.33	0.56	2.32	12.60	8.16	9.79	19.23
500 x 20	1.41	31.07	234.85	1.53	32.91	215.32	3.95	31.27	185.93	11.74	37.67	203.61	6.37	33.07	195.33	108.26	134.71	302.87
Average	0.15	2.89	21.39	0.17	3.05	19.78	0.44	2.90	16.89	1.12	3.51	18.79	0.64	3.11	18.05	10.27	12.68	28.00
ARPT	-0.99	-0.94	-0.64	-0.97	-0.94	-0.63	-0.88	-0.86	-0.62	-0.96	-0.92	-0.63	-0.96	-0.92	-0.64	-0.65	-0.62	-0.35

Table 11: Computational Times of heuristics II

	FF-ICH1	FF-ICH2	FF-ICH3	FF-NEH(5)	FF-NEH(10)	FF-PR1(5)	FF-PR1(10)	FF-PR1(15)
20 x 5	0.00	0.00	0.01	0.00	0.00	0.02	0.05	0.06
20 x 10	0.01	0.01	0.01	0.00	0.01	0.03	0.06	0.10
20 x 20	0.01	0.01	0.01	0.00	0.01	0.05	0.10	0.16
50 x 5	0.05	0.06	0.08	0.01	0.03	0.23	0.44	0.66
50 x 10	0.09	0.12	0.16	0.03	0.05	0.44	0.89	1.32
50 x 20	0.18	0.22	0.27	0.05	0.10	0.87	1.71	2.64
100 x 5	0.32	0.52	0.89	0.09	0.18	1.84	3.74	5.36
100 x 10	0.77	1.30	2.71	0.18	0.35	3.75	7.66	11.75
100 x 20	1.95	2.71	5.18	0.32	0.65	8.29	16.28	24.29
200 x 10	7.93	11.00	49.87	1.21	2.40	36.42	67.75	72.55
200 x 20	18.85	24.30	100.12	2.34	4.58	72.73	144.41	146.06
500 x 20	273.86	397.42	3451.85	33.92	67.65	458.55	458.72	491.43
Average	25.33	36.47	300.93	3.18	6.33	48.60	58.48	63.03
ARPT	-0.48	-0.26	1.13	-0.86	-0.72	1.37	3.68	5.45

Table 12: Computational Times of heuristics III

Different values of parameters x and y of the heuristic $FF(x) - FPE(y)$ have been analysed according to the literature (see e.g. Liu and Reeves, 2001; Pan and Ruiz, 2013). More specifically, $x \in \{1, 2, 15, n/10, n/m, n\}$ and $y \in \{1, n/10, n\}$ have been employed. Regarding the parameters of the heuristics $FF - NEH(x)$ and $PR1(a)$, the same values than in Pan and Ruiz (2013) (i.e. $x \in \{5, 10\}$ and $a \in \{5, 10, 15\}$) are chosen since the analysis of these parameters was already performed by these authors. Detailed $ARPD$ results are shown in Tables 7, 8 and 9, while computational results are shown in Tables 10, 11 and 12.

Graphically, the new efficient set of heuristics using the average computational time is shown in Figure 5 while new efficient heuristics using $ARPT$ as time reference are shown in Figure 6. For the former, the Pareto set is formed by the following heuristics: $FF(1)$, $FF(2)$, $FF(2) - FPE(1)$, $FF(15) - FPE(1)$, $FF(15) - FPE(n/10)$, $FF(15) - FPE(n)$, $FF(n/10) - FPE(n)$, $FF(n) - FPE(n)$, $FF - IC1$, $FF - IC2$, $FF - PR1(5)$, $FF - PR1(10)$ and $FF - PR1(15)$. For the latter, the efficient frontier is: $FF(1)$, $FF(2)$, $FF(n/10)$, $FF(n/m)$, $FF(2) - FPE(n/10)$, $FF(15) - FPE(n/10)$, $FF(n/10) - FPE(1)$, $FF(n/10) - FPE(n/10)$, $FF(n/10) - FPE(n)$, $FF(n/m) - FPE(n)$, $FF - IC1$, $FF - IC2$, $IC2$, $IC3$, $FF - PR1(5)$, $FF - PR1(10)$ and $FF - PR1(15)$. It represent a total of 17 heuristics in the new Pareto set. 15 of these heuristics correspond to the new set of heuristics presented in this paper.

Seven paired samples t-test were carried out in order to compare the new set of efficient heuristic to the old one. Comparisons were always performed between algorithms with higher ARPT, i.e.: Raj vs $FF(1)$, $LR(1)$ vs $FF(2)$; $LR(n/m) - FPE(n)$ vs $FF - IC1$; $LR - NEH(5)$ vs $FF(15) - FPE(n/10)$; RZ vs $FF(n/10) - FPE(n/10)$; $RZ - LW$ vs $FF - IC1$ and $LR - NEH(10)$ vs $FF(n/10) - FPE(n)$. The results of the analysis are shown in Table 13. Statistically significant differences were found for each comparison being 0.000 the maximum p-value found in the analysis. The Least Significant Difference (LSD) intervals for each heuristics are shown in Figure 7.

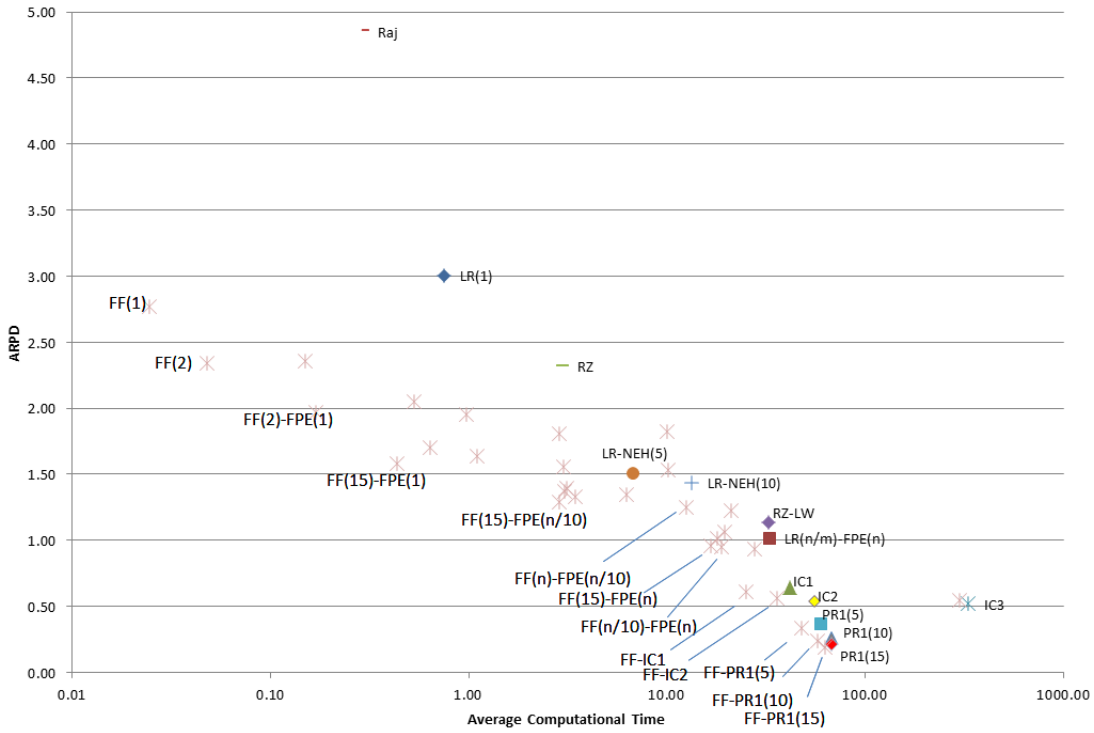

Figure 5: *ARPD* versus average computational times. X-axis (Average computational time) is shown in logarithmic scale. Noted that only the efficient heuristics of the new set of heuristics are named.

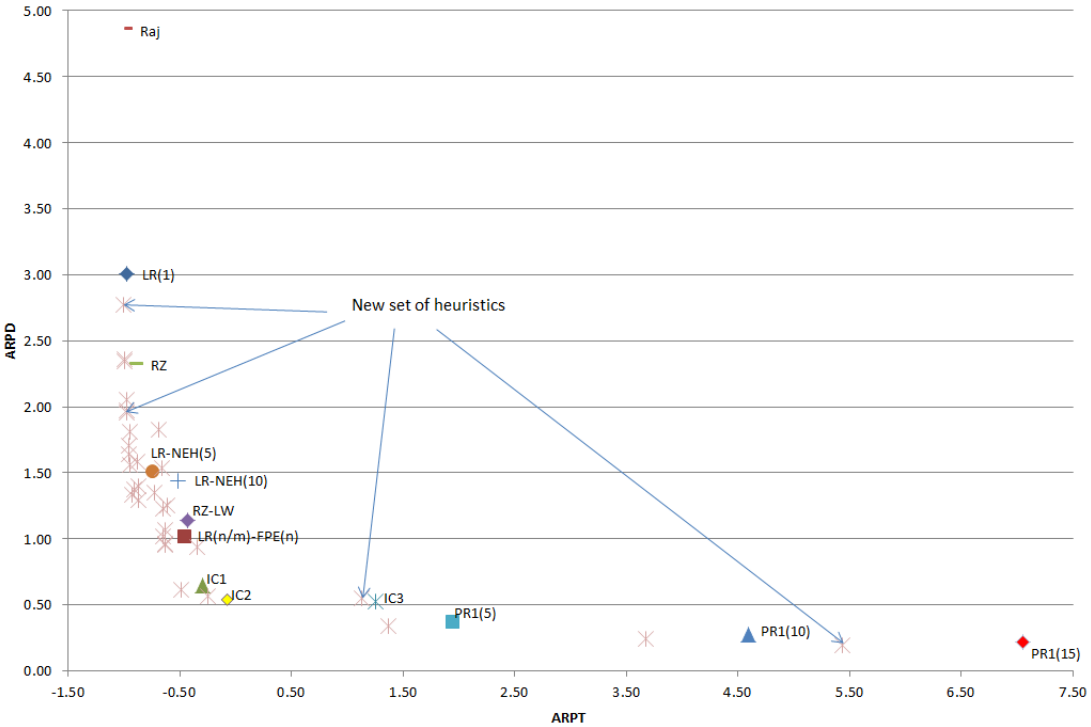

Figure 6: *ARPD* versus *ARPT*.

Heuristics	Mean	SEM	IC - Lower	IC - Upper	t	Significance
Raj vs FF(1)	2.096	1.798	1.770	2.421	12.767	0.000
LR(1) vs FF(2)	0.668	1.507	0.396	0.941	4.857	0.000
LR(n/m)-FPE(n) vs FF-IC1	0.404	0.667	0.283	0.524	6.628	0.000
LR-NEH(5) vs FF(15)-FPE(n/10)	0.233	0.709	0.105	0.361	3.598	0.000
RZ vs FF(n/10)-FPE(n/10)	0.989	1.156	0.780	1.198	9.374	0.000
RZ-LW vs FF-IC1	0.510	0.830	0.360	0.660	6.738	0.000
LR-NEH(10) vs FF(n/10)-FPE(n)	0.500	0.711	0.371	0.628	7.699	0.000

Table 13: Paired samples t-test using Taillard's benchmark.

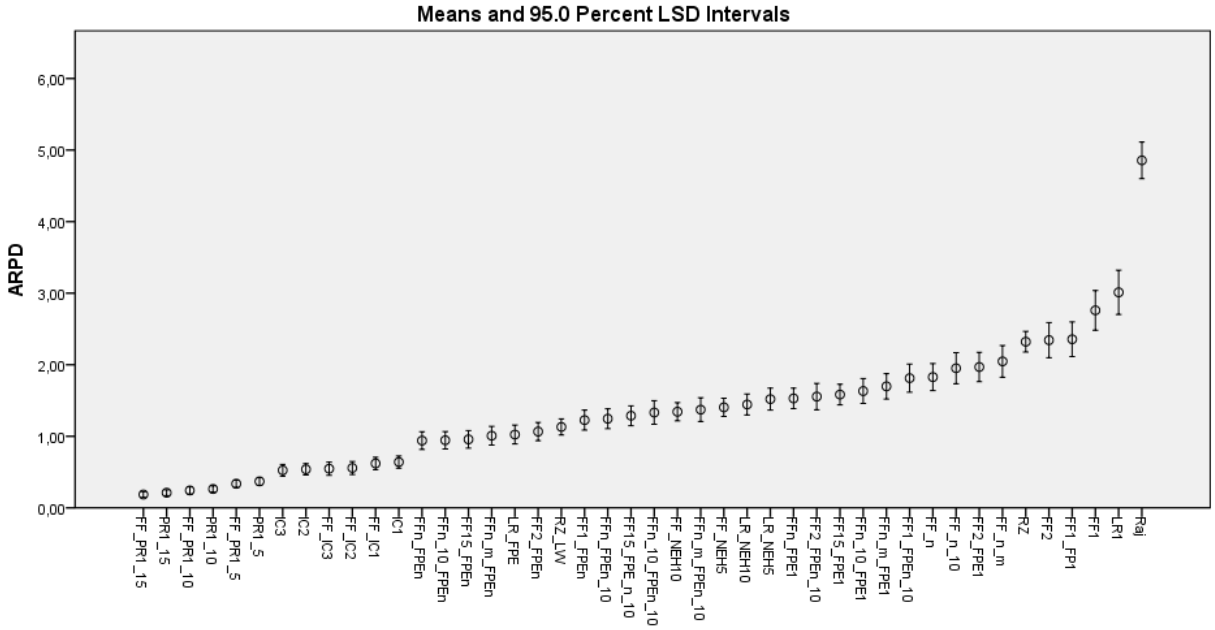

Figure 7: LSD intervals of the RPD for each analysed heuristics.

6 Conclusions

In this paper, we have presented a new constructive heuristic denoted by $FF(x)$ for the permutation flowshop scheduling problem to minimise flowtime. This heuristic constructs the final sequence adding jobs, one by one, at the end of the sequence based in the machine idle times and in the makespan of the inserted job. The complexity of the proposed algorithm is $x \cdot n^2 \cdot m$ being lower than the complexity of the heuristics in the actual Pareto set.Since most efficient heuristics use the algorithm LR in some of their phases, the latter can be replaced by the new algorithm FF in each of these heuristics, so a new set of efficient heuristics is obtained as a benchmark for new efficient heuristics for the problem.

Additionally, certain issues have been identified in the evaluation of efficient heuristics in the literature. When analysing the trade-off between the quality of the solutions and the time required by the heuristic to obtain them, an dimensionless (and relative) variable ($ARPD$) was used to represent the former while a dimensional and absolute variable (average computational time of the heuristic) was used to represent the latter. As discussed earlier in this paper, some heuristics are deemed as efficient whereas they are not efficient for many problem sizes.

The intended contribution of the paper can be summarised as follow:

- FF, a new heuristic of complexity $O(n^2 \cdot m)$ has been presented. This heuristic achieves better results in terms of both CPU time and $ARPD$ than those obtained by the fastest efficient heuristic (with complexity $O(n^3 \cdot m)$) so far.

- A new set of 17 efficient heuristics for the problem has been identified. 15 out of these 17 incorporate FF.

- An alternative representation of the efficiency of heuristics is proposed by introducing the indicator $ARPT$ for evaluating the computational time of the heuristics resulting in a more robust Pareto set.

Acknowledgements

The authors are sincerely grateful to the anonymous referees, who provide very valuable comments on an earlier version of the paper. This research has been funded by the Spanish Ministry of Science and Innovation, under the project "SCORE" with reference DPI2010-15573/DPI.

References

Dudek, R. and Teuton, O. (1964). Development of m stage decision rule for scheduling n jobs through m machines. *Operations Research*, 12:471.

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.

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.

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.

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

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.

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.

Nawaz, M., Enscore, Jr, E. 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 and Operations Research*, 40(1):117–128.

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.

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.

Tasgetiren, M., Liang, Y.-C., Sevkli, M., and Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, 177(3):1930–1947.

Vakharia, A. and Wemmerlov, U. (1990). Designing a cellular manufacturing system: a materials flow approach based on operation sequences. *IIE Transactions*, 22:84–97.