Computers & Operations Research 40 (2013) 117–128

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Invited Review

A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime

Quan-Ke Pan [a,b], Rubén Ruiz [c,*]

[a] State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang 110819, PR China
[b] College of Computer Science, Liaocheng University, Liaocheng 252059, PR China
[c] Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain

ARTICLE INFO

Available online 19 June 2012

Keywords:
Scheduling
Flowshop
Flowtime
Heuristics

ABSTRACT

In recent years, a large number of heuristics have been proposed for the minimization of the total or mean flowtime/completion time of the well-known permutation flowshop scheduling problem. Although some literature reviews and comparisons have been made, they do not include the latest available heuristics and results are hard to compare as no common benchmarks and computing platforms have been employed. Furthermore, existing partial comparisons lack the application of powerful statistical tools. The result is that it is not clear which heuristics, especially among the recent ones, are the best. This paper presents a comprehensive review and computational evaluation as well as a statistical assessment of 22 existing heuristics. From the knowledge obtained after such a detailed comparison, five new heuristics are presented. Careful designs of experiments and analyses of variance (ANOVA) techniques are applied to guarantee sound conclusions. The comparison results identify the best existing methods and show that the five newly presented heuristics are competitive or better than the best performing ones in the literature for the permutation flowshop problem with the total completion time criterion.

Contents

* Corresponding author. Tel.: +34 96 387 70 07x74946; fax: +34 96 387 74 99.
E-mail addresses: panquanke@gmail.com (Q.-K. Pan), rruiz@eio.upv.es (R. Ruiz).

0305-0548/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2012.05.018

1. Introduction

A flowshop is a common layout in production shops where m continuously available machines are disposed in series. Each machine is a production stage and products must visit all machines in order. Scheduling in a flowshop entails the production of n known jobs from a set $J=\{1, 2,..., n\}$. All the n jobs follow the same order of visitation to the machines. This order is, without loss of generality, machine 1, machine 2 and so un until machine m. Each job requires a given known, deterministic and non-negative processing time at each machine, denoted as $p_{i,j}$, $j\in J$, $i=1, 2,\ldots, m$. The flowshop scheduling problem or FSP in short is a theoretical version of reality and several simplifying assumptions apply: all jobs are independent and available for processing at time 0; machines are continuously available; each job is either waiting for processing or being processed by a machine at any given time; machines can only process one job at a time, etc. A complete list of these assumptions is detailed, for example, in Baker [3]. A solution for the FSP is a production sequence or schedule for all jobs which aims at optimizing a given criterion. Most optimization criteria in scheduling are based on the completion times of the jobs or C_j. The time at which a given job finishes processing at a given machine is denoted as $C_{i,j}$ and therefore, $C_{m,j}=C_j$. The most common and widely studied optimization criterion in the flowshop problem is the makespan or $C_{\max}$ minimization. Minimizing makespan is important in situations where a batch of jobs is received and it is required to be completed as soon as possible. For example, a multi-item order submitted by a single customer which needs to be delivered at the earliest possible time. The makespan criterion also increases the utilization of machines. The paper of Johnson [20] is recognized as the pioneering work for the FSP where the specific cases of two and three machines were studied with the objective of makespan minimization. Since then, the FSP has attracted considerable attention from researchers and hundreds of papers have been published in scheduling and related journals. The vast majority of research on flowshop scheduling deals with makespan minimization and several survey papers have been published like those of Framinan et al. [7], Ruiz and Maroto [39], Hejazi and Saghafian [16] and Gupta and Stafford [15].

As of late, there has been an increasing interest in other objective functions. Sometimes each job is needed as soon as it is completed. Similarly, the need to reduce Work In Process (WIP) or in-process inventory has fostered the study of the total flowtime, also referred to as total completion time. When all jobs are available for processing at time 0 (i.e., no release times) the flowtime of a job is equal to its completion time and hence, the total flowtime is equal to $\sum_{j=1}^{n} C_j$. Flowtime minimization leads to a more stable utilization of machines. The FSP with a total flowtime minimization objective was initially classified as $n/m/F/\sum C_j$ following the four parameter notation A/B/C/D of Conway et al. [5]. Later, it has been denoted as $F//\sum C_j$ using the three field notation $\alpha/\beta/\gamma$ of Graham et al. [13]. In the most general setting, the FSP has a search space of $(n!)^m$ sequences. However, the majority of the published research deals with a more restricted version, the so called permutation flowshop scheduling problem of PFSP in which job passing is not allowed and all machines follow the same sequence of jobs. In this case, the search space reduces to $n!$ sequences. The PFSP is classified as $n/m/P/\sum C_j$ or $F/prmu/\sum C_j$ according to Pinedo [34]. We will refer to this last problem with flowtime objective as PFSP-TFT in short. The PFSP-TFT was demonstrated to be NP-Hard in the strong sense for two or more machines by Gonzalez and Sahni [12].

Initial efforts focused on the development of exact implicit enumeration techniques and on approximate approaches to obtain good (but not necessarily optimal) solutions. These solution techniques can be broadly classified into two groups referred to as heuristics and metaheuristics, respectively. Some initial heuristics for the PFSP were introduced by Campbell et al. [4], Gupta [14] and Miyazaki et al. [29], to name just a few. Metaheuristics include many different approaches, like genetic algorithms [42], simulated annealing [44], differential evolution [33] and many others. A metaheuristic method usually obtains better solutions than heuristic algorithms but normally at the cost of significantly added CPU time. Heuristics typically need no more than a few seconds whereas metaheuristics might take several minutes. This is problematic, especially if there are real time requirements or large scale problems [25]. Furthermore, effective and efficient heuristics are still needed in metaheuristic methods for the initial seed sequence. As a result, heuristics are still essential in the scheduling community.

This paper focuses on heuristics for the PFSP-TFT. The flowshop literature already contains some reviews such as Framinan et al. [11]. However, there is room for improvement: comparisons have been performed among no more than a few heuristics; the latest heuristics have not been compared; no common data sets have been used and available results cannot be easily generalized or are not even reproducible; existing comparisons have not carried out comprehensive statistical testing. For all these reasons, we provide an up to date comprehensive review and evaluation of the existing heuristics. From the knowledge obtained after such evaluation we also present five heuristics for the problem under consideration. In total we compare 27 heuristics, which are put through comprehensive computational and statistical testing. The benchmark of choice is given by Taillard [41]. Our results attest to the fact that the five presented heuristics outperform all heuristics proposed up to date.

The rest of the paper is organized as follows: in Section 2, the most well-known heuristics for the PFSP-TFT are reviewed. Section 3 presents the five new heuristics in detail. A comprehensive comparison of the various heuristics is given in Section 4. Finally, we conclude the paper in Section 5.

2. Heuristics for the flowshop scheduling problem

Framinan et al. [11] divided the existing heuristics into two groups: simple and composite methods. A heuristic commonly consists of one or more of three typical phases, namely index development, solution construction, and solution improvement. According to Framinan et al. [11], the method is regarded as composite if it employs a simple heuristic for one or more of the three above-mentioned phases [11]. Conversely, it is regarded as a simple method if no phase contains a heuristic. This distinction is sometimes not easy to apply for some methods but it represents a simple framework. Our literature review is therefore divided between simple and composite heuristics.

2.1. Simple heuristics

The CDS heuristic introduced by Campbell et al. [4] is a simple heuristic for the PFSP. It is basically an extension of the algorithm of Johnson [20]. The CDS creates m^{-1} problems with of two "virtual"

machines, each of them containing some of the original m machines. Johnson's algorithm is then applied to the m^{-1} problems with two virtual machines and m^{-1} sequences are obtained. The schedule with the minimum flowtime is selected. The CDS heuristic has a computational complexity of $O(m^2n+mn\log n)$ and researchers have typically used CDS as benchmark for comparisons. Gupta [14] introduced three simple heuristics, named minimum idle time (MINIT), minimum completion time (MICOT) and MINIMAX algorithms, and compared the results against the CDS heuristic providing better results with less computational time. However, it has to be noted that the maximum instance size tested at that time was really small with just 7 jobs and 20 machines maximum (7×20). Krone and Steiglitz [21] presented an early heuristic in which in the first phase, permutation sequences were improved by insertion movements. In the second phase, job passing was allowed. Miyazaki et al. [29] also presented a heuristic but in this case based on the improvement of the sequence by the interchange of adjacent jobs. Later, Miyazaki and Nishiyama [28] provided a similar extension but for the additional consideration of job weights. Ho and Chang [18] proposed a heuristic that works by minimizing the idle times between jobs in the m machine case. The heuristic was evaluated against other existing methods but mainly those proposed for makespan minimization. Rajendran and Chaudhuri [37] introduced three simple heuristics and compared them with those of Gupta [14], Miyazaki et al. [29] and the aforementioned heuristic of Ho and Chang [18]. The results favored the introduced methods for the studied instances. In a related work, Rajendran and Chaudhuri [36], the same authors presented another heuristic that uses a lower bound in the construction phase of the sequence. The proposed heuristic is applied also to the no-wait problem. No comparisons against the three heuristics of Rajendran and Chaudhuri [37] are shown.

The NEH heuristic of Nawaz et al. [32] is regarded as the best heuristic for the PFSP with makespan criterion [40,39]. It is based on the idea that jobs with larger total processing times should be scheduled as early as possible. Consequently, the heuristic first generates an initial order of jobs with respect to descending sums of their total processing times. Then a job sequence is constructed by evaluating all the sequences obtained by inserting a job from this initial order into all the possible positions of the current partial sequence. The NEH heuristic evaluates $[n(n+1)/2-1]$ sequences and has a complexity of $O(n^3m)$ for the TFT criterion. Due to its effectiveness, the NEH heuristic has been inspiring research on the total completion time criterion since its publication. Rajendran [35] proposed an insertion heuristic, denoted as Raj, having many similarities with the NEH heuristic. The heuristic arranges the jobs according to the weighted total processing times and inserts a job into a restricted subset of all possible positions of the current partial sequence. According to the author's results, the proposed heuristic is more efficient than the methods of Gupta [14], Miyazaki et al. [29] and Ho and Chang [18]. Another heuristic was proposed by Woo and Yim [46] (denoted as WY in short). Unlike the Raj heuristic, WY does not require an initial starting job sequence. However, it also has an insertion phase where a schedule is constructed by inserting all non-scheduled jobs in all possible positions of the partial sequence. This heuristic is also based on the aforementioned NEH heuristic but has a higher complexity of $O(n^4m)$. The authors concluded that their algorithm outperforms the adaptation for flowtime minimization of the NEH, CDS and Raj heuristics.

Framinan et al. [10] investigated the phases of the NEH heuristic and their contribution to its excellent performance regarding makespan minimization. They proposed to modify the NEH heuristic in order to accomplish total flowtime criterion, and proved that the NEH heuristic starting with an initial sequence of jobs sorted by an increasing (instead of decreasing) sum of processing times performs better than the adaptation of the original NEH heuristic. It almost equals the WY heuristic in terms of the quality of the solutions but with smaller computational times. Later, Framinan et al. [9] further delved into the NEH initialization and studied 177 different initial orders for the NEH, including some specially geared towards TFT minimization. Among the proposed methods, a heuristic called B5FT, consisting of the best of five-tuples among the 177 approaches, is shown to outperform the RZ heuristic of Rajendran and Ziegler [38] (to be discussed later) and the WY method which were regarded as the best constructive heuristics for the problem prior to the year 2000 according to Framinan et al. [11]. Framinan and Leisten [8] presented another NEH-based heuristic, referred to as FL, with the same complexity as the WY method. After the insertion process in the basic NEH heuristic, the obtained partial sequence is improved by performing a pairwise interchange improvement procedure. If a better result is obtained, the new partial solution is retained as the current partial sequence. Computational results indicated that this approach outperformed RZ and WY heuristics. More recently, Laha and Sarin [22] have presented a modification of the FL heuristic, denoted as FL-LS. It implements the iteration of the insertion step of the NEH heuristic by performing job insertions rather than the pairwise interchanges. The authors proved by numerical experiments that the modification significantly improves the performance of the FL heuristic while not affecting its computational complexity.

Ho [17] presented a sorting-based heuristic that includes an iterated improvement scheme based on job insertions and pairwise interchanges. The author compared the method with the heuristics of Rajendran and Chaudhuri [37] and Raj of Rajendran [35]. In this case, larger instances of up to 50×20 were tested and the proposed heuristic was shown to be superior. However, this heuristic seems closer to local search techniques such as simulated annealing or tabu search rather than to constructive heuristics as its computational effort does not make it suitable for large problem sizes and/or in those environments where sequencing decisions are required in a short time [11].

Other heuristics assign a weight or index to every job and then arrange the sequence by sorting the jobs according to the assigned index. This idea was exploited by Wang et al. [45]. The authors presented two heuristic approaches by choosing jobs according to a given weight or index function and appending them to a current partial sequence. The first one, named less idle time rule (LIT), focuses on reducing machine idle times, while the second one, named smallest process distance rule (SPD), focuses on reducing both machine idle times and job waiting times. The second approach also consists of two heuristics; one is based on the Euclidean distance measure, while the other is based on the linear distance. The authors did not compare their heuristics with previous ones. Instead, they compared them against the lower bound provided by Ahmadi and Bagchi [1]. The heuristics proposed by Wang et al. [45] have a computational complexity of $O(n^2m)$. The already mentioned RZ heuristic of Rajendran and Ziegler [38] consists of two phases. The first phase involves the generation of a seed sequence according to a priority rule similar to the shortest weighted processing time, whereas the second phase improves the solution by carrying out a local search based on the sequential insertion of each job in the seed sequence at each possible different position of the incumbent partial sequence. The RZ heuristic has a complexity of $O(n^3m)$. Comparisons between the RZ and WY heuristics have been performed by several researchers [9,26]. It was found that the RZ heuristic performs better than the WY heuristic for small-sized problem instances but the relative performance of the WY heuristic improves with increasing number of jobs and finally it surpasses

the RZ heuristic. In addition, the effectiveness of the improvement scheme of the RZ heuristic was also demonstrated by Rajendran and Ziegler [38], and it has been used as an improvement procedure in several composite heuristics [11,25] and Allahverdi and Aldowaisan [2]). Li and Wu [26] have developed an improved RZ heuristic, denoted RZ-LW, where the authors generate an initial sequence by sorting the jobs in ascending order of the sum of processing times, and then perform the RZ local search to the solution until no improvement is found. The performance of RZ-LW is shown to be comparable to that of the Framinan and Leisten [8] but needs far less computational time.

Liu and Reeves [27] proposed a constructive heuristic, referred to as LR that initially sorts jobs according to some indexes that consider both the machine idle times and the effects on the completion times of later jobs. The LR heuristic does not fix the number of sequences to be generated and it is therefore flexible in its computational effort. It can be adjusted according to the requirements of the problem. The benchmark of Taillard [41] has been used to compare the proposed heuristic against the previous ones including Wang et al. [45], Ho [17], Rajendran and Ziegler [38] and Woo and Yim [46]. The computational results demonstrated that the LR heuristic is the best performer, especially in large sized problems.

2.2. Composite heuristics

Liu and Reeves [27] proposed an improvement scheme based on job pairwise exchanges. Starting from an initial sequence, the procedure tries to exchange every job with a certain number of jobs following it in the sequence. If the best sequence obtained by these exchanges is better than the current sequence, it is replaced. After all the jobs are tested, the procedure starts over again from the first job in the sequence. The above procedure is repeated until no improvement can be found for a round of trials (i.e., a form of local search up to local optimality). This procedure is known as the forward pairwise exchange (FPE). The reversed version which checks the exchanges of jobs from right to left in the sequence is called backward pairwise exchange (BPE). The authors studied the effectiveness of different combinations of their heuristics with local search, referred to as LR(x)-FPE and LR(x)-BPE, respectively. The result is that composite methods are more effective than the simple ones at the expense of additional computation time.

Allahverdi and Aldowaisan [2] proposed a total of seven composite heuristics by combining the NEH, WY and RZ methods with local search procedures including FPE with restart (FPE-R in short) and the local search of the RZ heuristic. The authors compared their methods (named IH1∼IH7) against many of the earlier heuristics like those of Ho [17], Wang et al. [45], Rajendran and Ziegler [38] and Woo and Yim [46]. The experimental results indicated that the performance of the heuristic by Ho [17] is good but computationally demanding. They also reported that the heuristics by Wang et al. [45] do not perform well when compared with the others except the CDS method. The proposed heuristics outperform all others in terms of solution quality, and IH7 is the best performer. Framinan and Leisten [8] proposed an improvement to the IH7 heuristic, called IH7-FL, by employing the FL heuristic as an initial solution instead of the WY heuristic as in the original IH7. Later, Framinan et al. [11] presented a comprehensive comparison of recent heuristics for the problem. A total of eight heuristics were compared and a number of composite methods were also presented. One of these new composite heuristics, named C2-FL, is observed to produce better solutions than those of the best method from the earlier study [8].

More recently, Li et al. [25] presented three composite heuristics, denoted as IC1, IC2, and IC3, respectively, by integrating FPE, FPE-R and RZ local search with an effective iterative method where the procedure is repeated until no better solution is found or a given stopping criterion is reached. Computational results show that the three proposed algorithms outperform the existing best composite ones including the C1-FL and C2-FL of Framinan et al. [11] and IH7-FL of Framinan and Leisten [8]. Among the presented heuristics, IC3 performs best in terms of solution quality but needs much more CPU time than both IC1 and IC2. In a related work [24], the authors presented two composite heuristics, named ECH1 and ECH2, which were similar to the heuristics IC1, IC2 and IC3.

A summary of the different heuristics reviewed in chronological order is reported in Table 1.

3. Proposed heuristics

The previous evaluation has prompted us to test some new composite heuristics. We present five new high performing methods. The first one is a simple procedure which combines the LR heuristic of Liu and Reeves [27] and the NEH algorithm. The others are composite heuristics based on this first one and local search methods. More specifically, the RZ local search of Rajendran and Ziegler [38] and a Variable Neighborhood Search scheme (VNS) based on the work of Mladenovic and Hansen [30].

3.1. The presented LR-NEH(x) heuristic

3.1.1. The LR(x) heuristic in detail

The LR(x) heuristic developed by Liu and Reeves [27] constructs x different sequences by appending jobs one by one using an index function. The sequence with the minimum flowtime is selected as the final solution. The index function consists of two terms: the weighted total machine idle time and the artificial total flowtime. Let π be a partial sequence formed by k already scheduled jobs, and U be the set of unscheduled jobs, i.e., those not yet in π. A job $j \in U$ is selected and appended to π according to an index function $\xi_{j,k}$. The weighted total machine idle time between the processing of the job occupying the kth position of the sequence and job j is computed as follows:

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

where $C_{i,[k]}$ is the completion time of the job in the kth position of π at machine i.

The other jobs in U are considered as a single artificial job λ. Its processing time is the average of the processing times of these jobs. Job λ is appended to job j and its completion time $C_{i,\lambda}$ is calculated. Then the total flowtime of jobs j and λ, $AT_{j,k}$, is given below:

$$AT_{j,k} = C_{m,j} + C_{m,\lambda} \quad (2)$$

And the index function $\xi_{j,k}$ is finally defined as follows:

$$\xi_{j,k} = (n-k-2)IT_{j,k} + AT_{j,k} \quad (3)$$

The index function $\xi_{j,k}$ is calculated for all jobs in U. The job with the minimum value of this index function is selected, and ties are broken by selecting the one with the minimum weighted total machine idle time $IT_{j,k}$.

Finally, the procedure of LR(x) is outlined in Fig. 1.

LR(x) does not fix the number of sequences to be generated, and it can be adjusted to the requirements of the problem.

3.1.2. The NEH heuristic

The NEH heuristic of Nawaz et al. [32] was originally designed for the FPSP with the objective of minimizing the makespan. The first step consists of ordering jobs according to descending total

Table 1
Summary of heuristics for flowshop scheduling with total flowtime criterion.

Year	Authors	Acronym	Heuristic type	Comments
1970	Campbell et al.	CDS	Simple	Based on Johnson's rule
1972	Gupta	MINIT	Simple	Based on job pair exchange
		MICOT		Based on job pair exchange
		MINIMAX		Based on Johnson's rule
1974	Krone and Steiglitz		Simple	Based on insertion improvement and job passing
1978	Miyazaki et al.		Simple	Based on interchange of adjacent jobs
1980	Miyazaki and Nishiyama		Simple	Based on interchange of adjacent jobs and job weights
1991	Ho and Chang		Simple	Minimizing the idle time between jobs
1991	Rajendran and Chaudhuri		Simple	Based on lower bound
1992	Rajendran and Chaudhuri		Simple	Considering a job's impact to its immediate successor
1993	Rajendran	Raj	Simple	Based on NEH
1995	Ho		Simple	Based on sorting
1997	Wang et al.	LIT	Simple	Assigning a weight to every job
		SPD1	Simple	Assigning a weight to every job
		SPD2	Simple	Assigning a weight to every job
1997	Rajendran and Ziegler	RZ	Simple	Assigning a weight to every job and performing RZ local search
1998	Woo and Yim	WY	Simple	Based on NEH
2001	Liu and Reeves	LR(x)	Simple	Assigning a weight to every job
		LR(x)-FBE	Composite	Based on LR(x) and FPE
		LR(x)-BPE	composite	Based on LR(x) and BPE
2002	Framinan et al.	NEH-flowtime	Simple	Based on NEH
2002	Allahverdi and Aldowaisan	IH1	Composite	Base on NEH and FPE-R
		IH2	Composite	Based on NEH
		IH3	Composite	Consisting of IH2 and FPE-R
		IH4	Composite	Consisting of WY and FPE-R
		IH5	Composite	Consisting of RZ and FPE-R
		IH6	Composite	Consisting of WY and RZ local search
		IH7	Composite	Consisting of IH6 and FPE-R
2003	Framinan et al.	B5FT	Simple	Based on NEH
2003	Framinan and Leisten	FL	Simple	Based on NEH and interchange
		IH7-FL	Composite	consisting of FL, RZ and FPE-R
2005	Framinan et al.	C1-FL	Composite	Based on LR and FL
		C2-FL	Composite	Based on C1, RZ and FIE-R
2005	Li and Wu	RZ-LW	Simple	Based on iterated RZ local search
2006	Li and Wang	ECH1	Composite	Similar to IC3
		ECH2	Composite	Similar to IC2
2009	Li et al.	IC1	Composite	Consisting of LR and iterated RZ local search.
		IC2	Composite	Consisting of LR, iterated RZ and FPE
		IC3	Composite	Consisting of LR, iterated RZ and FPE-R
2009	Laha and Sarin	FL-LS	Simple	Based on NEH and Insertion

Procedure LR(x)

Generate a job sequence $\alpha=\{\alpha_1,\alpha_2,...,\alpha_n\}$ by ascending $\xi_{j,0}$ value (break ties according to ascending $IT_{j,0}$ value).

for $l:=1$ **to** x **do** %(generate x sequences)

$\pi^l:=\{\alpha_l\}$, $U:=J-\{\alpha_l\}$.

for $k:=2$ **to** n **do** %(construct a complete sequence)

Take the job j with minimum $\xi_{j,k}$ value (break ties according to minimum $IT_{j,k}$ value) from U and place it at the end of π^l. Remove job j from U.

endfor

endfor

return the sequence $\pi\in\{\pi^1,\pi^2,...,\pi^x\}$ with the minimum total flowtime.

Fig. 1. LR(x) heuristic.

processing times. The job with the maximum total processing time is placed first. All other jobs are inserted in all possible positions of the incumbent sequence and finally placed in the position with the lowest partial objective value. The procedure of NEH is described in Fig. 2.

Framinan et al. [10] adapted the NEH heuristic for total flowtime criterion, and found that ranking jobs according to their ascending total processing times performs much better than descending total processing times. As a result, we also employ this improved version. As we can see, the main loop of the NEH can be regarded as an insertion local search around the seed sequence β. We denote this local search as NEH(β) for our other composite heuristics.

3.1.3. The proposed LR-NEH(x) heuristic

The first presented heuristic is denoted as LR-NEH(x). It uses LR(x) and NEH to generate sequences. More specifically, we first generate a partial sequence with d jobs using the LR(x), and then the remaining n–d jobs are inserted into the partial sequence using the NEH heuristic. The relative positions of jobs generated

by the LR(x) are not changed as the algorithm progresses. The procedure of the proposed LR-NEH(x) is outlined in Fig. 3.

LR-NEH(x) has a single parameter d, which is basically the number of jobs after which NEH kicks in. Initial experiments showed that the best value was $d=3n/4$. However, we found that for reasonable values, LR-NEH(x) is robust as regards this parameter. We leave for a further study a deeper examination of the effect of this parameter.

3.2. Composite heuristic PR1(x)

3.2.1. Iterated RZ local search

The improvement procedure presented by Rajendran and Ziegler [38] is a typical local search based on an insertion neighborhood, which sequentially inserts each job in the seed sequence in all possible positions in the incumbent sequence and is, as mentioned, similar to the NEH heuristic. Let $\pi^s=\pi_1^s$, $\pi_2^s,\ldots,\pi_n^s$ be a seed sequence, and π be the incumbent sequence. The procedure of the RZ local search is given in Fig. 4.

The above RZ procedure is a one-pass local search process. The process can be iterated while improvements are found and local optimality is reached. This iterated process can find better results but at the expense of more computational effort. Therefore, a trade-off between effectiveness and efficiency arises. We denote the iterated RZ procedure as iRZ in short.

3.2.2. The proposed composite heuristic PR1(x)

Based on the LR-NEH(x) heuristic and the iRZ local search, we propose a composite heuristic PR1(x). PR1(x) improves each of the solutions generated by LR-NEH(x) using iRZ. To save computational effort, we terminate the iteration if the CPU time is longer than 0.01 mn seconds. The procedure of PR1(x) is outlined in Fig. 5.

Procedure NEH
Generate a job sequence $\beta=\{\beta_1,\beta_2,\ldots,\beta_n\}$ by descending order of total processing times.
$\pi:=\{\beta_1\}$
for $k:=2$ **to** n **do** %(construct a complete sequence)
Take job β_k from β and insert it in all the k possible positions of π.
Place job β_k in π at the tested position resulting in the lowest objective value.
endfor
return π

Fig. 2. NEH heuristic.

Procedure $RZ(\pi)$
$\pi^s:=\pi$
for $i:=1$ **to** n **do**
$\pi':=\pi$
Remove job π_i^s from π'.
Take job π_i^s and insert it in all possible positions of π' except for its original position.
Place job π_i^s in π' at the position resulting in the lowest total flowtime.
if $TFT(\pi')<TFT(\pi)$ **then** $\pi:=\pi'$ %($TFT(\pi)$ denotes the total flowtime of π)
endfor
return π

Fig. 4. RZ local search.

Procedure PR1(x)
Generate a job sequence $\alpha=\{\alpha_1,\alpha_2,\ldots,\alpha_n\}$ by ascending $\xi_{j,0}$ value (break ties according to ascending $IT_{j,0}$ value).
$l:=1$
repeat
$\pi^l:=\{\alpha_l\}$, $U:=J-\{\alpha_l\}$.
for $k:=2$ **to** d **do** %(construct a partial sequence with d jobs)
Take the job j with minimum $\xi_{j,k}$ value (break ties according to minimum $IT_{j,k}$ value) from U and place it at the end of π^l. Remove job j from U.
endfor
Generate a partial sequence $\beta=\{\beta_1,\beta_2,\ldots,\beta_{n-d}\}$ ($\beta_j\in U$, $j=1,2,\ldots,n-d$) by ascending order of total processing times.
for $k:=1$ **to** $n-d$ **do** %(construct a complete sequence)
Take job β_k from β and insert it in all the $k+d$ possible positions of π^l.
Place job β_k in π^l at the tested position resulting in the lowest total flowtime.
endfor
$\pi^l:=iRZ(\pi^l)$ %(perform iRZ local search to π^l)
$l:=l+1$
until $l>x$ **or** $CPUTime>0.01mn$ seconds
return the sequence $\pi\in\{\pi^1,\pi^2,\ldots,\pi^l\}$ with the minimum total flowtime.

Fig. 5. PR1(x) heuristic.

Procedure LR-NEH(x)
Generate a job sequence $\alpha=\{\alpha_1,\alpha_2,\ldots,\alpha_n\}$ by ascending $\xi_{j,0}$ value (break ties according to ascending $IT_{j,0}$ value).
for $l:=1$ **to** x **do** %(generate x sequences)
$\pi^l:=\{\alpha_l\}$, $U:=J-\{\alpha_l\}$.
for $k:=2$ **to** d **do** %(construct a partial sequence with d jobs)
Take the job j with minimum $\xi_{j,k}$ value (break ties according to minimum $IT_{j,k}$ value) from U and place it at the end of π^l. Remove job j from U.
endfor
% (NEH heuristic)
Generate a partial sequence $\beta=\{\beta_1,\beta_2,\ldots,\beta_{n-d}\}$ ($\beta_j\in U$, $j=1,2,\ldots,n-d$) by ascending order of total processing times.
for $k:=1$ **to** $n-d$ **do** %(construct a complete sequence)
Take job β_k from β and insert it in all the $k+d$ possible positions of π^l.
Place job β_k in π^l at the tested position resulting in the lowest total flowtime.
endfor
endfor
return the sequence $\pi\in\{\pi^1,\pi^2,\ldots,\pi^l\}$ with the minimum total flowtime.

Fig. 3. Proposed LR-NEH(x) heuristic.

3.3. Composite heuristic PR2(x)

The variable neighborhood search (VNS) is an effective metaheuristic presented by Mladenovic and Hansen [30]. Tasgetiren et al. [43] proposed a local search based on the *insertion+interchange* variant of the VNS method and embedded it in a particle swarm optimization algorithm to solve the permutation flowshop with both makespan and total flowtime criterion. As a result, it seems promising to employ VNS in the heuristics. Let π be an incumbent job permutation to improve, and $l_{\max}$ be the maximum number of iterations. The VNS is detailed in Fig. 6.

In the above VNS procedure, the pairwise interchange movement randomly selects two jobs in the sequence ϕ and exchanges their positions. The insertion movement removes a random job from its original position and inserts it in another randomly selected position. In order to have a sufficient exploration of both interchange and insert neighborhoods, we set $l_{\max}$ to $2n^2$. We simply change the iRZ of PR1(x) by VNS, resulting in the PR2(x) heuristic.

3.4. Composite heuristics PR3(x) and PR4(x)

The composite heuristic PR3(x) first generates an initial solution using LR-NEH(x), and then improves the solution using a different improvement procedure. The procedure of PR3(x) is given in Fig. 7.

In the above procedure, the parameter y for the LR-NEH(y) initialization is fixed at 10. The final proposed heuristic PR4(x) uses the VNS local search as an improvement procedure instead of the iRZ local search of PR3(x).

Procedure $VNS(\pi)$
  for $l := 1$ **to** $l_{\max}$ **do**
    improved $:= true$
    repeat
      $\phi := \pi$
      if improved $= true$ **then** Perform a pairwise interchange movement in ϕ.
      else Perform an insertion movement in ϕ.
      if $TFT(\phi) \le TFT(\pi)$ **then** $\pi := \phi$, improved $:= true$
      else improved $:= false$
    until improved $= false$
  endfor
return π

Fig. 6. VNS local search.

Procedure PR3(x)
  $\pi := \mathrm{LR-NEH}(y)$ %(generate an initial solution)
  $\pi^b := \pi$, $l := 1$
  repeat %(improvement procedure)
    $\pi' := iRZ(\pi)$ % (iRZ local search)
    if $TFT(\pi') < TFT(\pi^b)$ **then** $\pi^b := \pi'$.
    $\pi'' := NEH(\pi')$ % (NEH local search)
    if $TFT(\pi'') < TFT(\pi^b)$ **then** $\pi^b := \pi''$
    $\pi := NEH(\pi'')$ % (NEH local search)
    if $TFT(\pi) < TFT(\pi^b)$ **then** $\pi^b := \pi$
    $l := l + 1$
  until $l > x$ **or** $CPUTime > 0.01mn$ seconds
return π^b

Fig. 7. PR3(x) heuristic.

4. Computational and statistical experiments

In this section we conduct a comprehensive computational and statistical evaluation of most existing high-performing heuristics as well as of the presented methods. The tested heuristics comprise 14 simple and 13 composite heuristics as follows:

4.1. Simple heuristics

1. Raj heuristic of Rajendran [35],
2–4. LIT, SPD1 and SPD2 heuristics by Wang et al. [45],
5. RZ heuristic of Rajendran and Ziegler [38],
6. WY heuristic by Woo and Yim [46],
7–9. LR(1), LR(n/m) and LR(n) of Liu and Reeves [27],
10. NEH heuristic modified by Framinan et al. [10],
11. FL heuristic of Framinan and Leisten [8],
12. RZ-LW heuristic of Li and Wu [26],
13. FL-LS heuristic by Laha and Sarin [22],
14. Proposed LR-NEH(x) heuristic.

4.2. Composite heuristics

15–16. LR-FPE and LR-BPE of Liu and Reeves [27],
17. IH7 heuristic of Allahverdi and Aldowaisan [2],
18. IH7-FL heuristic of Framinan and Leisten [8],
19–20. Composite heuristics C1-FL and C2-FL of Framinan et al. [11],
21–23. IC1, IC2, IC3 heuristics of Li et al. [25],
24–27. The presented composite heuristics PR1(x), PR2(x), PR3(x) and PR4(x).

All other reviewed heuristics from Section 2 were clearly outperformed by the above heuristics in previous research and are not tested in this work. In addition, the general flowtime computing method presented by Li et al. [19] is employed to save computation time in all heuristics that allow it.

The test bed presented by Taillard [41] is a well-known set for the PFSP with makespan criterion, which consists of a total of 120 instances of various sizes, having 20, 50, 100, 200, and 500 jobs and 5, 10, or 20 machines. These instances are divided into 12 subsets, each of which consists of 10 instances with the same size. These subsets are denoted according to their sizes: 20×5, 20×10, 20×20, 50×5, 50×10, 50×20, 100×5, 100×10, 100×20, 200×10, 200×20 and 500×20. Recently, an increasing number of researchers have used this test bed to evaluate their algorithms dealing with total or mean flowtime criterion [27,23,24,43,19,6,25] and Zhang et al. [47], possibly among many others). Thus, we evaluate the above mentioned heuristics based on this test bed, and the performance measure is the relative percentage increase (*RPI*) as follows:

$$RPI(c_i) = (c_i - c^*)/c^* \times 100 \quad (4)$$

where C_i is the solution obtained by the i^{th} heuristic, and c^* is the best solution found by any of the heuristics.

All methods have been coded in Visual C++ 6.0 and run on a cluster of 30 blade severs each one with two Intel XEON 5254 processors running at 2.5 GHz with 16 GB of RAM memory. Each processor has four cores and the experiments are carried out in virtualized Windows XP machines, each one with one virtualized processor and 2 GB of RAM memory. In order to better estimate the performance and elapsed CPU time of the compared algorithms, a total of 5 replications for each instance are carried out. Results are then averaged across the 5 replications for each instance. In our five proposed heuristics LR-NEH(x), PR1(x), PR2(x), PR3(x) and PR4(x), x is tested at three values: 5, 10 and

15. This gives a total of 37 heuristics which are run 5 times each for the 120 instances of Taillard for a grand total of 22,200 results. The average *RPI* values, grouped for each subset (600 results averaged at each cell) are given in Tables 2 and 3 for simple and composite heuristics, respectively. Both types of heuristics are summarized in Table 4, ordered by *RPI*. All CPU times are given in seconds.

It can be seen that simple heuristics clearly perform worse than the composite ones except FL-LS and RZ-LW, which produce slightly smaller *RPI* values than IH7, IH7-FL and C1-FL. Among the simple heuristics, the worst performing algorithms are the three heuristics presented by Wang et al. [45], with SPD1 and SPD2 being more than 15% over the best solution found by any of the compared methods. The best simple heuristic is FL-LS, which produces the smallest mean *RPI* value of 1.22%. However, this is a very costly method, which needs, on average, 120.24 s. As a matter of fact, the column "PARETO" in Table 4 indicates the number of heuristics that Pareto-dominate a given one as regards average *RPI* and average CPU time. For FL-LS we see that there are 16 methods that dominate it: from the 17 heuristics with lower *RPI* values than FL-LS, all of them, except C2-FL, need less CPU time and therefore, dominate, in a Pareto sense, FL-LS.

The presented LR-NEH(x) heuristic yields a much smaller overall *RPI* value than both the NEH and LR heuristics for $x=5$, 10 or 15. Additionally, the proposed LR-NEH(x) is not dominated, i.e., it represents the best trade-off between CPU time and *RPI* among the simple heuristics. Raj is the fastest method, needing barely a tenth of a second, on average.

Among existing composite heuristics from the literature, IC3 is obviously the best performer producing 0.62% *RPI* value within 77.25 s. However, all presented composite heuristics at all tested x values result in lower *RPI* values at a lower computational cost.

Fig. 8 shows a scatter plot of average *RPI* versus average CPU time for the best performing methods. Pareto dominating heuristics are depicted in red (boldface in Table 4).

Previous tables and plots contain average results. We conduct comprehensive statistical analyses to ascertain if the observed differences in *RPI* values are indeed statistically significant. Design of experiments (DOE) and analyses of variance (ANOVA) [31] are conducted for all results.

We consider all 27 tested heuristics. Recall that the proposed methods are tested with three values of x, namely 5, 10 and 15. As a result, we have 37 methods, all of them present in Table 4. Five replicates and 120 instances are tested which recall results in 22,200 treatments. In Taillard's benchmark, not all combinations of n and m are present and therefore, n and m are not orthogonal. In order to study these two factors, we define a factor called type of instance "Type" which has 12 levels, 1 for 20×5, 2 for 20×10 and so on until 12 for 500×20. The replicate (note that all methods are run five independent times) is a witness factor that is shown to be statistically not significant with a p-value close to 1.0. This factor is then removed after validating the experiment. As a result, the initial ANOVA has two factors, Algorithm, with 37 levels, and type of instance, at 12 levels. The response variable is the *RPI*. ANOVA is a parametric statistical tool and there are three main assumptions: normality, homogeneity of variance (homoscedasticity) and independence of the residuals. We carefully checked all three assumptions and the results showed that no major departures were found. The only minor problem is a slight small departure from normality. However, as is well known, ANOVA is robust with respect to the normality assumption. The result of the ANOVA is that the two factors as well as the interaction between the two are statistically significant with p-values very close to 0. The most significant factor is the

Table 2
Results of the simple heuristics.

Instance	Raj		LIT		SPD1		SPD2		RZ		WY		LR(1)		LR(n/m)	
	RPI	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time
20×5	5.80	0.00	9.40	0.00	18.85	0.00	19.35	0.00	2.90	0.00	3.03	0.00	2.74	0.00	2.55	0.00
20×10	4.61	0.00	11.23	0.00	15.56	0.00	14.37	0.00	1.90	0.00	2.80	0.00	3.77	0.00	3.42	0.00
20×20	4.55	0.00	6.95	0.00	10.23	0.00	10.07	0.00	1.97	0.00	2.96	0.00	3.21	0.00	3.21	0.00
50×5	4.51	0.00	7.69	0.00	21.01	0.00	21.56	0.00	2.70	0.00	3.70	0.01	2.20	0.00	1.56	0.01
50×10	6.28	0.00	9.45	0.01	15.96	0.01	14.24	0.00	3.22	0.01	3.39	0.02	5.26	0.00	2.97	0.01
50×20	5.88	0.00	10.19	0.02	11.57	0.01	10.74	0.00	2.94	0.01	3.22	0.04	3.85	0.01	3.40	0.01
100×5	4.09	0.00	5.42	0.03	23.05	0.03	23.31	0.00	2.57	0.02	2.32	0.17	1.25	0.01	0.63	0.17
100×10	4.73	0.00	8.04	0.06	20.98	0.05	19.55	0.00	3.26	0.05	2.78	0.35	2.89	0.01	2.03	0.14
100×20	6.03	0.01	8.19	0.14	12.41	0.11	10.22	0.00	2.72	0.09	3.03	0.68	4.28	0.03	3.36	0.14
200×10	4.64	0.03	6.39	0.54	22.25	0.46	21.50	0.10	2.77	0.33	2.56	6.03	2.47	0.10	1.39	1.99
200×20	4.98	0.06	9.14	1.01	17.80	0.90	14.95	5.82	2.51	0.70	2.35	11.72	3.81	0.20	2.00	2.00
500×20	4.21	0.81	7.03	15.28	18.76	12.61	18.87	11.29	2.34	10.01	1.79	483.79	1.79	3.10	0.94	77.23
Average	5.02	0.08	8.26	1.42	17.37	1.18	16.56	1.43	2.65	0.94	2.83	41.90	3.13	0.29	2.29	6.81
Instance	LR(n)		NEH		FL		RZ-LW		FL-LS		LR-NEH(5)		LR-NEH(10)		LR-NEH(15)	
	RPI	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time
20×5	2.44	0.00	5.05	0.00	2.49	0.00	1.54	0.00	1.74	0.00	2.26	0.00	2.26	0.00	2.26	0.00
20×10	3.35	0.00	4.14	0.00	2.40	0.00	1.43	0.00	1.25	0.00	2.51	0.00	2.51	0.00	2.51	0.00
20×20	2.42	0.01	3.88	0.00	2.44	0.00	1.07	0.00	1.04	0.01	2.01	0.00	1.89	0.00	1.89	0.01
50×5	1.56	0.05	4.25	0.00	2.02	0.03	1.63	0.01	1.37	0.04	1.33	0.01	1.33	0.01	1.33	0.02
50×10	2.81	0.10	5.08	0.00	2.54	0.05	1.44	0.04	1.61	0.08	2.48	0.01	2.43	0.02	2.43	0.04
50×20	3.07	0.23	4.39	0.01	2.01	0.10	1.63	0.05	1.28	0.17	2.67	0.02	2.43	0.05	2.43	0.07
100×5	0.63	0.83	3.18	0.01	1.20	0.31	1.69	0.10	1.12	0.52	1.35	0.04	1.25	0.07	1.25	0.11
100×10	2.00	1.37	4.54	0.02	2.41	0.73	1.09	0.28	1.12	1.24	1.67	0.08	1.54	0.16	1.54	0.24
100×20	2.67	2.76	4.48	0.04	2.04	1.47	0.87	0.57	0.91	2.59	2.44	0.17	2.16	0.34	1.98	0.52
200×10	1.39	19.85	3.16	0.14	1.39	10.16	1.40	2.42	1.11	18.38	1.03	0.59	0.96	1.16	0.95	1.74
200×20	1.94	39.98	3.88	0.29	1.62	20.91	0.80	6.26	1.32	38.61	1.57	1.27	1.52	2.57	1.48	3.74
500×20	0.82	1544.32	2.32	4.00	1.29	701.12	0.88	114.84	0.73	1381.23	0.80	17.66	0.68	35.24	0.65	52.85
Average	2.09	134.12	4.03	0.37	1.99	61.24	1.29	10.38	1.22	120.24	1.84	1.65	1.75	3.30	1.72	4.94

Table 3
Results of the composite heuristics.

Instance	LR-FPE		LR-BPE		IH7		IH7-FL		C1-FL		C2-FL		IC1		IC2		IC3		PR1(5)		PR1(10)	
	RPI	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time
20×5	1.30	0.00	1.40	0.00	1.31	0.00	1.66	0.00	1.68	0.00	1.22	0.00	1.00	0.00	0.66	0.00	0.66	0.00	0.57	0.00	0.53	0.01
20×10	1.91	0.00	1.91	0.00	1.45	0.00	1.45	0.00	1.74	0.00	0.90	0.00	1.30	0.00	1.07	0.00	1.06	0.00	0.84	0.01	0.63	0.01
20×20	1.60	0.00	2.07	0.00	1.24	0.00	1.34	0.00	2.32	0.00	1.12	0.01	1.29	0.00	1.35	0.00	1.32	0.00	0.56	0.01	0.40	0.03
50×5	0.72	0.02	0.83	0.02	1.61	0.03	1.35	0.03	1.67	0.03	1.04	0.05	0.84	0.02	0.61	0.03	0.62	0.03	0.51	0.05	0.46	0.11
50×10	1.37	0.03	1.51	0.02	1.84	0.04	1.63	0.07	2.27	0.06	1.46	0.09	1.10	0.04	0.87	0.06	0.82	0.07	0.50	0.14	0.30	0.27
50×20	1.70	0.04	2.12	0.05	1.78	0.08	1.33	0.13	2.63	0.11	1.36	0.17	1.05	0.07	0.77	0.12	0.68	0.15	0.77	0.27	0.44	0.54
100×5	0.34	0.22	0.33	0.21	1.01	0.55	0.71	0.47	0.94	0.43	0.67	0.54	0.30	0.24	0.14	0.30	0.18	0.38	0.56	0.43	0.47	0.90
100×10	0.87	0.29	0.87	0.29	1.52	0.81	1.71	0.91	1.80	0.85	0.74	1.33	0.69	0.36	0.52	0.57	0.48	1.02	0.53	1.06	0.46	2.24
100×20	1.86	0.40	2.04	0.51	1.49	1.40	1.44	1.77	1.98	1.57	1.01	2.90	1.01	0.61	0.92	1.13	0.77	1.91	0.37	2.81	0.26	5.53
200×10	0.59	3.08	0.53	3.10	1.41	15.77	0.91	14.28	1.19	11.48	0.60	20.66	0.52	3.80	0.33	6.29	0.29	14.29	0.22	10.10	0.21	19.35
200×20	0.97	4.63	0.80	5.07	1.32	24.18	1.19	25.56	1.60	22.22	0.82	49.46	0.48	6.50	0.47	11.08	0.43	21.86	0.18	26.17	0.18	43.16
500×20	0.39	115.48	0.38	126.84	1.13	1006.51	0.89	989.50	0.82	730.96	0.40	1583.46	0.20	161.33	0.17	219.79	0.14	887.24	0.34	169.30	0.34	169.30
Average	1.14	10.35	1.23	11.34	1.43	87.45	1.30	86.06	1.72	63.98	0.95	138.22	0.81	14.41	0.66	19.95	0.62	77.25	0.50	17.53	0.39	20.12

Instance	PR1(15)		PR2(5)		PR2(10)		PR2(15)		PR3(5)		PR3(10)		PR3(15)		PR4(5)		PR4(10)		PR4(15)	
	RPI	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time	*RPI*	Time
20×5	0.37	0.01	0.56	0.01	0.45	0.02	0.37	0.02	0.88	0.00	0.85	0.01	0.84	0.01	0.64	0.01	0.46	0.02	0.40	0.02
20×10	0.48	0.02	0.86	0.02	0.62	0.03	0.52	0.05	0.68	0.01	0.68	0.01	0.68	0.02	0.75	0.02	0.57	0.03	0.49	0.04
20×20	0.31	0.04	0.56	0.03	0.45	0.06	0.31	0.08	0.65	0.02	0.57	0.02	0.57	0.04	0.57	0.03	0.45	0.05	0.40	0.07
50×5	0.45	0.16	0.33	0.10	0.30	0.20	0.30	0.30	0.61	0.07	0.57	0.14	0.57	0.20	0.45	0.12	0.39	0.22	0.34	0.33
50×10	0.24	0.39	0.68	0.23	0.55	0.45	0.45	0.67	0.59	0.17	0.43	0.30	0.42	0.44	0.87	0.27	0.69	0.51	0.58	0.75
50×20	0.38	0.83	0.86	0.44	0.63	0.90	0.54	1.34	0.50	0.32	0.39	0.62	0.32	0.92	0.77	0.51	0.64	0.98	0.56	1.44
100×5	0.43	1.37	0.27	0.64	0.22	1.28	0.21	1.92	0.56	0.58	0.50	1.14	0.49	1.66	0.33	0.78	0.30	1.49	0.27	2.20
100×10	0.41	3.41	0.39	1.58	0.36	3.16	0.33	4.73	0.43	1.49	0.40	2.86	0.36	4.24	0.48	1.88	0.46	3.61	0.44	5.35
100×20	0.20	8.22	0.65	3.38	0.48	6.76	0.40	10.14	0.26	3.18	0.25	6.11	0.22	9.01	0.72	3.93	0.66	7.56	0.60	11.17
200×10	0.19	21.18	0.16	11.11	0.14	21.18	0.14	21.25	0.28	13.11	0.26	21.22	0.26	21.18	0.21	13.34	0.21	20.96	0.21	20.96
200×20	0.18	43.14	0.49	24.19	0.39	43.44	0.39	43.44	0.42	25.78	0.42	43.12	0.42	43.14	0.44	28.37	0.43	43.98	0.43	44.00
500×20	0.34	172.36	0.33	126.74	0.33	126.74	0.33	126.78	0.21	130.25	0.21	132.19	0.21	128.32	0.19	110.61	0.19	110.61	0.19	113.38
Average	0.33	20.93	0.51	14.04	0.41	17.02	0.36	17.56	0.51	14.58	0.46	17.31	0.45	17.43	0.54	13.32	0.45	15.84	0.41	16.64

Table 4
Summary of all heuristics, ordered by *RPI*.

#	Algorithm	*RPI*	Time	Type	PARETO	#	Algorithm	*RPI*	Time	Type	PARETO
1	**PR1(15)**	0.33	20.93	Composite	0	20	RZ-LW	1.29	10.38	Simple	1
2	**PR2(15)**	0.36	17.56	Composite	0	21	IH7-FL	1.30	86.06	Composite	18
3	PR1(10)	0.39	20.12	Composite	1	22	IH7	1.43	87.45	Composite	19
4	PR2(10)	0.41	17.02	Composite	1	23	C1-FL	1.72	63.98	Composite	17
5	**PR4(15)**	0.41	16.64	Composite	0	24	**LR-NEH(15)**	1.72	4.94	Simple	0
6	PR3(15)	0.45	17.43	Composite	2	25	**LR-NEH(10)**	1.75	3.30	Simple	0
7	**PR4(10)**	0.45	15.84	Composite	0	26	**LR-NEH(5)**	1.84	1.65	Simple	0
8	PR3(10)	0.46	17.31	Composite	3	27	FL	1.99	61.24	Simple	20
9	PR1(5)	0.50	17.53	Composite	5	28	LR(n)	2.09	134.12	Simple	26
10	**PR3(5)**	0.51	14.58	Composite	0	29	LR(n/m)	2.29	6.81	Simple	3
11	**PR2(5)**	0.51	14.04	Composite	0	30	**RZ**	2.65	0.94	Simple	0
12	**PR4(5)**	0.53	13.32	Composite	0	31	WY	2.83	41.90	Simple	22
13	IC3	0.62	77.25	Composite	12	32	**LR(1)**	3.13	0.29	Simple	0
14	IC2	0.66	19.95	Composite	10	33	NEH	4.03	0.37	Simple	1
15	IC1	0.81	14.41	Composite	2	34	**Raj**	5.02	0.08	Simple	0
16	C2-FL	0.95	138.22	Composite	15	35	LIT	8.26	1.42	Simple	4
17	**LR-FPE**	1.14	10.35	Composite	0	36	SPD2	16.56	1.43	Simple	5
18	FL-LS	1.22	120.24	Simple	16	37	SPD1	17.37	1.18	Simple	4
19	LR-BPE	1.23	11.34	Composite	1						

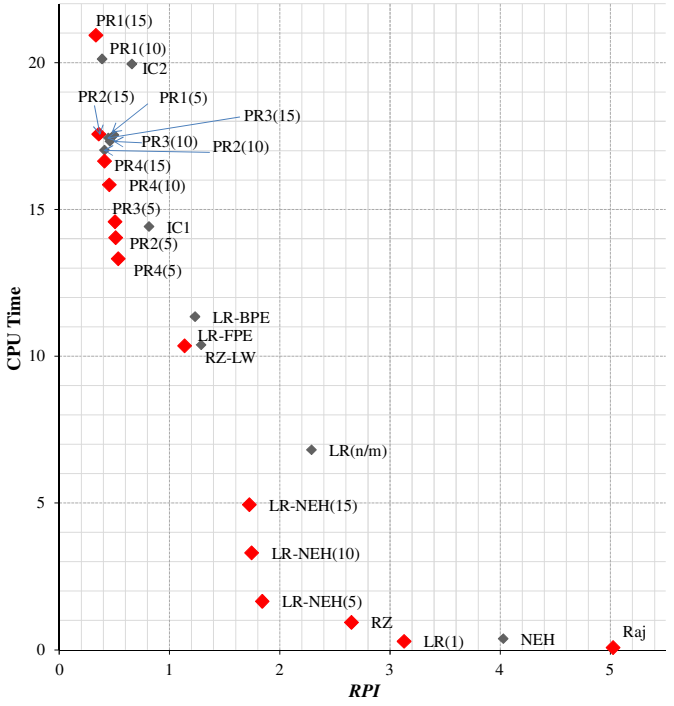

Fig. 8. Average *RPI* versus average CPU time for the heuristics. Pareto dominant methods shown in red. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article).

Algorithm with an *F*-Ratio close to 5000. An initial means plot with 99% confidence level intervals (not shown due to space limitations) clearly shows that the following heuristics are statistically different (from worst to best): SPD1, SPD2, LIT, Raj, NEH and LR(1). Note that all these methods have average RPI over 3%. They are statistically worse than all other heuristics and are therefore disregarded from the ANOVA and from following plots (this in turn also helps with the normality assumption). We employ the most restrictive technique for calculating the confidence intervals around the means: the Tukey's Honest Significant Difference (HSD). If the intervals around two plotted means overlap, it means that there are no statistically significant differences between the means at the given confidence level. The means plot with the remaining 31 heuristics is given in Fig. 9.

It can be observed from Fig. 9 that heuristics can be divided into 14 homogenous groups where no significant differences can be found within each group. The last three heuristics are each in a group, i.e., from worst to best, WY, RZ and LR(n/m). Group 11 is formed by methods FL and LR(n). In Fig. 9 we see how the confidence intervals for the average *RPI* of these two methods overlap. All groups are formed in a similar way. Fig. 9 also shows, in red, the intervals from those Pareto non-dominated heuristics.

It is shown that our proposed methods, from PR1 to PR4, in all three values of x, are better than all other heuristics. Statistically speaking, from PR1(15) to PR3(10) we have significant differences with IC3, the best competing method from the literature. It has to be pointed out that IC3 needs, on average, more than 77 s of CPU time while PR1(15) needs less than 21 s on average.

Of course, all previous statistical analyses depict the overall picture of the heuristics across all instances. The results vary slightly from one instance size to another. The interaction between the type of instance and the heuristics is relatively weak (still statistically significant but with a rather small *F*-Ratio). An example of this type of interaction is given in Fig. 10. Only four heuristics are shown for clarity.

We see that the confidence intervals are very wide (there are only 10 instances at each group) and there is not enough data to draw strong conclusions. We see, for example, how PR1(15) is statistically equivalent to IC3 for several instance groups, while being better for the others. Only for 100×5, IC3 results in a lower *RPI*, albeit this difference is not statistically significant.

There are other cases where some differences appear. However, we believe that the main interest lies with the average performance depicted in the previous Fig. 9.

In a nutshell, we put forward the following statements based on the above comparison and analysis. (1) All the simple heuristics are surpassed by the composite ones except FL-LS and RZ-LW regarding to the quality of solutions. On the other hand, most simple heuristics run much faster than their composite counterparts. (2) The best four simple heuristics are FL-LS, RZ-LW, the proposed LR-NEH(x) and FL in terms of effectiveness with the computational time of the proposed LR-NEH(x) being much less than that of the other three heuristics. (3) Both IC2 and IC3 are significantly better than the other existing composite heuristics in terms of overall *RPI* value. However, both are outperformed by the presented heuristics PR1(x)–PR4(x) both in terms of *RPI* as well as CPU time.

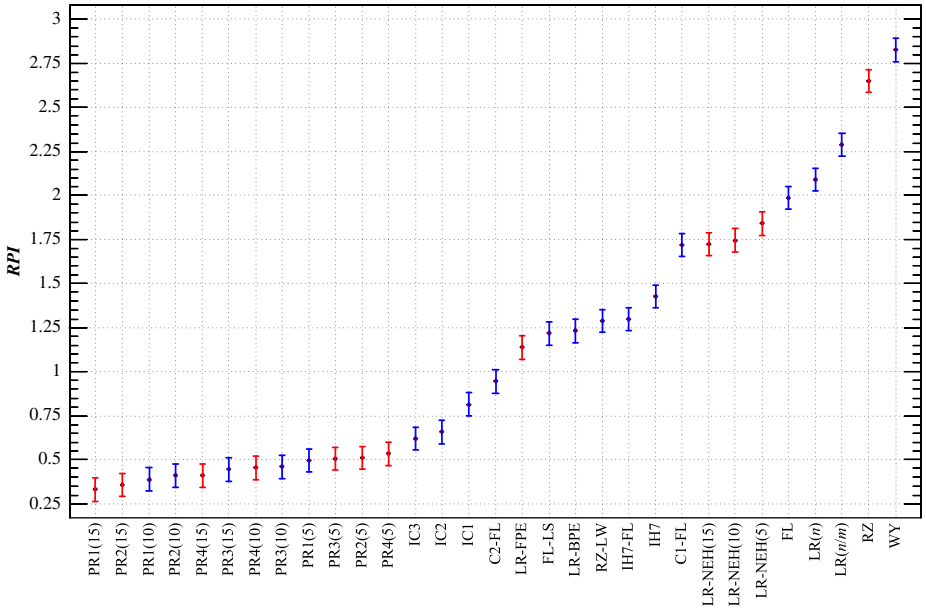

Fig. 9. Means plot for the *RPI* with Tukey's Honest Significant Difference (HSD) 99% confidence intervals for the 31 best performing heuristics.

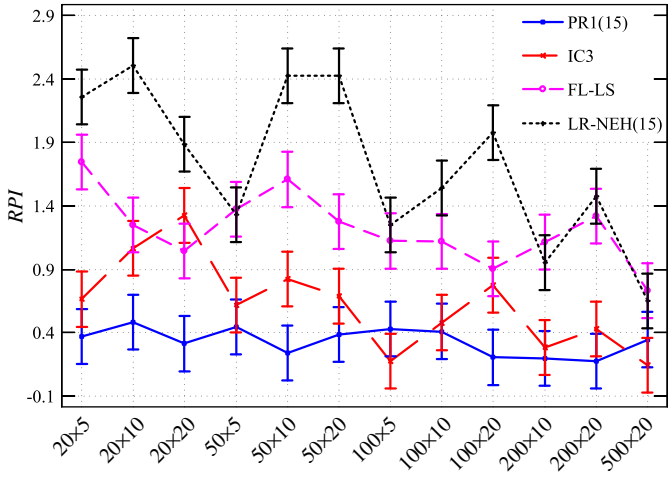

Fig. 10. Means plot for the *RPI* with Tukey's Honest Significant Difference (HSD) 99% confidence intervals for the interaction between the instance type and some chosen heuristics.

5. Conclusions

In this paper we have carried out an extensive review and comparison of the heuristics for the permutation flowshop scheduling problem with total or mean flowtime minimization criterion. A total of 22 existing heuristics have been coded and tested. With the knowledge obtained, five new methods have been presented. The well known Taillard [41] benchmark has been employed. All heuristics have been coded in the same language and have been tested on the same computing platform therefore the results are fully comparable. Furthermore, extensive use of the design of experiments (DOE) approach and analysis of variance (ANOVA) statistical technique results in sound conclusions.

Among simple heuristics, the FL-LS of Laha and Sarin [22] is the best performer. However, it is computationally costly; needing about one order of magnitude more CPU time than other composite heuristics that are, in turn, better performers. Our simple proposed LR-NEH(x) method represents a good trade-off between CPU time and quality, dominating most other existing simple heuristics from a Pareto perspective.

It is demonstrated that the heuristics IC2 and IC3 presented by Li et al. [25] were the best two performers from the literature as regards the quality of solutions and composite heuristics. However, our four presented composite heuristics PR1(x)–PR4(x) result both in lower average relative percentage deviations and lower average CPU time. For example, PR1(x) results in an average deviation of just 0.33% and average CPU time of 20.93 s whereas IC3 has almost double the deviation (0.62%) and more than three times more CPU time (77.25 s). In conclusion, our presented methods can now be considered state-of-the-art heuristics for the permutation flowshop scheduling problem with total flowtime minimization criterion.

Acknowledgments

This research is partially supported by National Science Foundation of China (60874075, 61174187), and Science Foundation of Shandong Province, China (BS2010DX005), and Postdoctoral Science Foundation of China (20100480897). Rubén Ruiz is partially funded by the Spanish Ministry of Science and Innovation, under the project "SMPA—Advanced Parallel Multiobjective Sequencing: Practical and Theorerical Advances" with reference DPI2008-03511/DPI and by the Small and Medium Industry of the Generalitat Valenciana (IMPIVA) and by the European Union through the European Regional Development Fund (FEDER) inside the R+D program "Ayudas dirigidas a Institutos Tecnológicos de la Red IMPIVA" during the year 2011, with project number IMDEEA/2011/142.

References

[1] Ahmadi RH, Bagchi U. Improved lower bounds for minimizing the sum of completion times of n jobs over m machines in a flow shop. European Journal of Operational Research 1990;44(3):331–6.
[2] Allahverdi A, Aldowaisan T. New heuristics to minimize total completion time in m-machine flowshops. International Journal of Production Economics 2002;77(1):71–83.
[3] Baker KR. Introduction to sequencing and scheduling. New York: Wiley; 1974.

[4] Campbell HG, Dudek RA, Smith ML. Heuristic algorithm for n job, m machine sequencing problem. Management Science Series B-Application 1970;16(10): B630–7.
[5] Conway RW, Maxwell WI, Miller LW. Theory of scheduling. Reading, Mass: Addison-Wesley; 1967.
[6] Dong XY, Huang HK, Chen P. An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. Computers & Operations Research 2009;36(5):1664–9.
[7] Framinan JM, Gupta JND, Leisten R. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. Journal of the Operational Research Society 2004;55(12):1243–55.
[8] Framinan JM, Leisten R. An efficient constructive heuristic for flowtime minimisation in permutation flow shops. Omega—International Journal of Management Science 2003;31(4):311–7.
[9] Framinan JM, Leisten R, Rajendran C. 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 2003;41(1):121–48.
[10] Framinan JM, Leisten R, Ruiz-Usano R. Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation. European Journal of Operational Research 2002;141(3):559–69.
[11] Framinan JM, Leisten R, Ruiz-Usano R. Comparison of heuristics for flowtime minimisation in permutation flowshops. Computers & Operations Research 2005;32(5):1237–54.
[12] Gonzalez T, Sahni S. Flowshop and jobshop schedules: complexity and approximation. Operations Research 1978;26(1):36–52.
[13] Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 1979;5(2):287–326.
[14] Gupta JND. Heuristic algorithms for multistage flowshop scheduling problem. AIIE Transactions 1972;4(1):11–8.
[15] Gupta JND, Stafford EF. Flowshop scheduling research after five decades. European Journal of Operational Research 2006;169(3):699–711.
[16] Hejazi SR, Saghafian S. Flowshop-scheduling problems with makespan criterion: a review. International Journal of Production Research 2005;43(14):2895–929.
[17] Ho JC. Flowshop sequencing with mean flowtime objective. European Journal of Operational Research 1995;81(3):571–8.
[18] Ho JC, Chang Y-L. A new heuristic for the n-job, M-machine flow-shop problem. European Journal of Operational Research 1991;52(2):194–202.
[19] Jarboui B, Eddaly M, Siarry P. An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems. Computers & Operations Research 2009;36(9):2638–46.
[20] Johnson SM. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1954;1(1):61–8.
[21] Krone MJ, Steiglitz K. Heuristic-programming solution of a flowshop-scheduling problem. Operations Research 1974;22(3):629–38.
[22] Laha D, Sarin SC. A heuristic to minimize total flow time in permutation flow shop. Omega—International Journal of Management Science 2009;37(3): 734–9.
[23] Li X, Liu L, Wu C. A fast method for heuristics in large-scale flow shop scheduling. Tsinghua Science & Technology 2006;11(1):12–8.
[24] Li X, Wang Q. Iterative heuristics for permutation flows hops with total flowtime minimization. In: Shen W, editor. Information technology for balanced manufacturing systems. IFIP TC5, Proceedings of the WG 5.5 seventh international conference on information technology for balanced automation systems in manufacturing and services. New York: Springer; 2006. p. 349–56.
[25] Li XP, Wang Q, Wu C. Efficient composite heuristics for total flowtime minimization in permutation flow shops. Omega—International Journal of Management Science 2009;37(1):155–64.
[26] Li XP, Wu C. An efficient constructive heuristic for permutation flow shops to minimize total flowtime. Chinese Journal of Electronics 2005;14(2):203–8.
[27] Liu JY, Reeves CR. Constructive and composite heuristic solutions to the $P//\Sigma C_i$ scheduling problem. European Journal of Operational Research 2001;132(2):439–52.
[28] Miyazaki S, Nishiyama N. Analysis for minimizing weighted mean flow-time in flow-shop scheduling. Journal of the Operations Research Society of Japan 1980;23(2):118–32.
[29] Miyazaki S, Nishiyama N, Hashimoto F. An adjacent pairwise approach to the mean flow-time scheduling problem. Journal of the Operations Research Society of Japan 1978;21(2):287–99.
[30] Mladenovic N, Hansen P. Variable neighborhood search. Computers & Operations Research 1997;24(11):1097–100.
[31] Montgomery DC. Design and analysis of experiments. Hoboken, NJ: Wiley; 2008.
[32] Nawaz M, Enscore Jr EE, Ham I. A heuristic algorithm for the m machine, n job flowshop sequencing problem. Omega—International Journal of Management Science 1983;11(1):91–5.
[33] Pan QK, Tasgetiren MF, Liang YC. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Computers & Industrial Engineering 2008;55(4):795–816.
[34] Pinedo M. Scheduling: theory, algorithms, and systems. New York: Springer; 2008.
[35] Rajendran C. Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. International Journal of Production Economics 1993;29(1): 65–73.
[36] Rajendran C, Chaudhuri D. A flowshop scheduling algorithm to minimize total flowtime. Journal of the Operations Research Society of Japan 1991;34(1): 28–46.
[37] Rajendran C, Chaudhuri D. An efficient heuristic approach to the scheduling of jobs in a flowshop. European Journal of Operational Research 1992;61(3): 318–25.
[38] Rajendran C, Ziegler H. An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. European Journal of Operational Research 1997;103(1):129–38.
[39] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 2005;165(2): 479–94.
[40] Taillard E. Some efficient heuristic methods for the flow-shop sequencing problem. European Journal of Operational Research 1990;47(1):65–74.
[41] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64(2):278–85.
[42] Tang LX, Liu JY. A modified genetic algorithm for the flow shop sequencing problem to minimize mean flow time. Journal of Intelligent Manufacturing 2002;13(1):61–7.
[43] Tasgetiren MF, Liang YC, Sevkli M, Gencyilmaz G. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. European Journal of Operational Research 2007;177(3):1930–47.
[44] Varadharajan TK, Rajendran C. A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. European Journal of Operational Research 2005;167(3): 772–95.
[45] Wang CG, Chu CB, Proth JM. Heuristic approaches for $n/m/F/\Sigma C_i$ scheduling problems. European Journal of Operational Research 1997;96(3):636–44.
[46] Woo HS, Yim DS. A heuristic algorithm for mean flowtime objective in flowshop scheduling. Computers & Operations Research 1998;25(3):175–82.
[47] Zhang Y, Li XP, Wang Q. Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. European Journal of Operational Research 2009;196(3):869–76.