Omega 40 (2012) 218–229

Contents lists available at ScienceDirect

Omega

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

Effective heuristics for the blocking flowshop scheduling problem with makespan minimization

Quan-Ke Pan [a,b,*], Ling Wang [c]

[a] State Key Laboratory of Synthetical Automation for Process Industries (Northeastern University), Shenyang 110819, China
[b] College of Computer Science, Liaocheng University, Liaocheng 252059, China
[c] Tsinghua National Laboratory for Information Science and Technology (TNList), Department of Automation, Tsinghua University, Beijing 100084, China

ARTICLE INFO

Article history:
Received 16 December 2010
Accepted 19 June 2011
Processed by Jozefowska
Available online 30 June 2011

Keywords:
Flowshop
Blocking
Makespan
Constructive heuristics
Composite heuristics

ABSTRACT

The blocking flowshop scheduling problem with makespan criterion has important applications in a variety of industrial systems. Heuristics that explore specific characteristics of the problem are essential for many practical systems to find good solutions with limited computational effort. This paper first presents two simple constructive heuristics, namely weighted profile fitting (wPF) and PW, based on the profile fitting (PF) approach of McCormick et al. [Sequencing in an assembly line with blocking to minimize cycle time. Operations Research 1989;37:925–36] and the characteristics of the problem. Then, three improved constructive heuristics, called PF-NEH, wPF-NEH, and PW-NEH, are proposed by combining the PF, wPF, and PW with the enumeration procedure of the Nawaz–Enscore–Ham (NEH) heuristic [A heuristic algorithm for the *m*-machine, *n*-job flow shop sequencing problem. OMEGA-International Journal of Management Science 1983;11:91–5] in an effective way. Thirdly, three composite heuristics i.e., PF-NEH$_{LS}$, wPF-NEH$_{LS}$, and PW-NEH$_{LS}$, are developed by using the insertion-based local search method to improve the solutions generated by the constructive heuristics. Computational simulations and comparisons are carried out based on the well-known flowshop benchmarks of Taillard [Benchmarks for basic scheduling problems. European Journal of Operation Research 1993;64:278–85] that are considered as blocking flowshop instances. The results show that the presented constructive heuristics perform significantly better than the existing ones, and the proposed composite heuristics further improve the presented constructive heuristics by a considerable margin. In addition, 17 new best-known solutions for Taillard benchmarks with large scale are found by the presented heuristics.

1. Introduction

Flowshop problem has important applications in manufacturing systems, assembly lines, and information service facilities [1–4]. As an important branch of flowshop scheduling problems, the blocking flowshop scheduling has attracted much attention in recent years. The classic flowshop scheduling considers the processing of *n* jobs on *m* machines in the same order, and it is assumed that the intermediate buffers between two consecutive machines have infinite capacity so that jobs can be stored in the buffers to wait for their next operations. Nevertheless, in practice, there exist production processes where no intermediate buffer exists between machines due to technical requirements or the process characteristics. Therefore, a job having completed processing on a machine has to remain on this machine and to block itself until next machine is available for processing. The flowshop scheduling problem without intermediate buffers, or the blocking flowshop scheduling problem, has important applications in a variety of industrial systems. Grabowski and Pempera [5] described an example coming from the manufacturing of concrete blocks, which did not allow stock in some stages of the manufacturing process. Ronconi [6] described another example in the chemical industry where partially processed jobs (physical and chemical changes in the material) were sometimes kept in the machines because of the lack of intermediary storage. Gong et al. [7] considered a scheduling problem with blocking constraint found in the iron and steel industry. A comprehensive review on flowshop scheduling with blocking and no wait constraint in process can be found in Hall and Sirskandarajah [8].

This paper considers minimizing the makespan or maximum completion time for the *n*-job *m*-machine flowshop scheduling problem with blocking constraint, where there are no buffers between machines and hence intermediate queues of jobs waiting in the production system for their next operations are not

* Corresponding author at: State Key Laboratory of Synthetical Automation for Process Industries (Northeastern University), Shenyang 110819, China.
Tel.: +86 635 8258240.
E-mail address: panquanke@gmail.com (Q.-K. Pan).

0305-0483/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.omega.2011.06.002

allowed. This problem is denoted as *Fm/block/Cmax* by using the well-known $\alpha/\beta/\gamma$ notation presented by Graham et al. [9]. For the complexity of the problem, although it can be solved by using Gilmore and Gomory's algorithm when $m=2$ [10], it is proved by Hall and Sriskandarajah [8] to be NP-hard in the strong sense when $m>2$. Therefore, it is unlikely that efficient algorithms will be found that can optimally solve the general blocking flowshop problem.

For decades, heuristics and meta-heuristics have been mainly developed for solving the blocking flowshop scheduling problem. With regards to the heuristics, McCormick et al. [11] developed a profile fitting (PF) approach for solving sequencing problems in an assembly line with blocking to minimize cycle time, where the PF tried to initially sequence jobs leading to the minimization of the idle and blocking times on machines. Leisten [12] presented a more comprehensive approach for dealing with permutation and non-permutation flowshops with finite and unlimited buffers to maximize the use of buffers and to minimize the machine blocking. However, according to the numerical experiments, the author concluded that their heuristic did not produce better solutions than the Nawaz–Enscore–Ham (NEH) heuristic [13] initially proposed for the classic flowshop problem. Based on the makespan properties proposed by Ronconi and Armentano [14], Ronconi [6] proposed a MinMax (MM) heuristic for the blocking flowshop problem with makespan criterion. Then the author obtained two heuristics, i.e., PFE and MME, by combining the MM and PF heuristics with the enumeration procedure of the NEH heuristic. It was demonstrated by the authors that the MME and PFE heuristics outperformed the NEH algorithm in problems with up to 500 jobs and 20 machines. According to the connection between no-wait flowshop scheduling problem and blocking flowshop scheduling problem, Abadi et al. [15] proposed a heuristic for minimizing cycle time in blocking flowshop scheduling. Recently, Ronconi and Henriques [16] tried to minimize the total tardiness in a flowshop with blocking and presented some constructive heuristics providing promising results for the problems considered.

For the meta-heuristics, Caraffa et al. [17] developed a genetic algorithm for the large size restricted slowdown flowshop problem in which the blocking flowshop problem was a special case. Grabowski and Pempera [18] developed a tabu search (TS) approach to minimize makespan for the blocking flowshop scheduling problem. According to the experimental results, the authors claimed that the TS approach outperformed both the genetic algorithm [17] and Ronconi's branch-and-bound method [19], which used the new lower bounds exploited the blocking nature and were better than those presented in their earlier paper [14], and they also reported the obtained upper bounds for 120 Taillard benchmarks [20]. Recently, Wang et al. [21] proposed a hybrid discrete differential evolution (HDDE) algorithm by combing the presented discrete version of the differential evolution and insertion neighborhood based local search procedure. It was shown that the HDDE performed much better than the TS approach and improved 112 out of 120 best-known solutions for Taillard benchmarks provide by Grabowski and Pempera [18]. More recently, Ribas et al. [22] proposed an iterated greedy algorithm, and the author claimed that it outperformed the HDDE algorithm and provided new best-known solutions for most of Taillard benchmarks.

Generally, a meta-heuristic method can obtain better solution than a heuristic but it needs much more computational time, which cannot meet the real time requirements in practice, especially for large-scale cases [23]. Therefore, heuristics that explore the specific characteristics of the problem are essential for many practical manufacturing systems to find good solutions with limited computational effort. For this purpose, we present five constructive heuristics and three composite heuristics in this paper. Computational results based on the benchmark instances demonstrated the effectiveness and efficiency of the presented heuristics.

The remaining contents of this paper are organized as follows. In Section 2, the blocking flowshop scheduling problem is stated and formulated. In Section 3, the presented constructive heuristics are described in detail. Section 4 presents three composite heuristics by combining the constructive heuristic with a local search procedure. The computational results and comparisons are provided in Section 5. Finally, we end the paper with some concluding remarks in Section 6.

2. The blocking flowshop scheduling problem

The blocking flowshop scheduling problem can be described as follows. There are n jobs from the set $J=\{1,2,\ldots,n\}$ and m machines from the set $M=\{1,2,\ldots,m\}$. Each job $j\in J$ will be sequentially processed on machine 1, machine 2, and so on until last machine m. Operation $o_{j,k}$ corresponds to the processing of job $j\in J$ on machine $k\in M$ during an non-negative, known and deterministic processing time $p_{j,k}$. Since the flowshop has no intermediate buffers, a job cannot leave a machine until its next machine downstream is free. In other words, the job has to be blocked on its machine if its next machine is not free. Without loss of generality, we assume that all jobs are independent and available for processing at time 0; machines are continuously available; each machine can process at most one job and each job can be processed on at most one machine at any time. Given that the sequence in which the jobs are to be processed is the same for each machine. The aim is then to find a sequence for processing all jobs on all machines so that its maximum completion time (i.e. makespan) is minimized.

Each schedule of jobs can be represented as a permutation $\pi=(\pi_1,\pi_2,\ldots,\pi_n)$, where $\pi_i\in J$. Let Π denote the set of all such permutations. The aim of the considered problem is to find a permutation $\pi^*\in\Pi$ such that

$$C_{\max}(\pi^*)\le C_{\max}(\pi)^* \quad \forall\pi\in\Pi, \tag{1}$$

where $C_{\max}(\pi)$ represents the makespan value of permutation π.

Let $d_{[i],k}$ be the departure time of operation $o_{[i],k}$, where $[i]$ represents the ith job, i.e., π_i, of permutation π. According to the literature [18,24], $d_{[i],k}$ can be calculated as follows:

$$d_{[1],0}=0, \tag{2}$$

$$d_{[1],k}=d_{[1],k-1}+p_{[1],k}, \quad k=1,\ldots,m-1, \tag{3}$$

$$d_{[i],0}=d_{[i-1],1}, \quad j=2,\ldots,n, \tag{4}$$

$$d_{[i],k}=\max\{d_{[i],k-1}+p_{[i],k},\ d_{[i-1],k+1}\}, \quad i=2,\ldots,n,\ k=1,\ldots,m-1, \tag{5}$$

$$d_{[i],m}=d_{[i],m-1}+p_{[i],m}, \quad i=1,\ldots,n, \tag{6}$$

where $d_{[i],0}$, $i=1,\ldots,n$, denotes the starting time of job π_i on the first machine, and $d_{[i],m}$ represents the completion time of job π_i in the shop. In the above recursion, the departure times of the first job on every machine are calculated first, then the second job, and so on until the last job. Then the makespan or the maximum completion time of the schedule $\pi=(\pi_1,\pi_2,\ldots,\pi_n)$ is defined by $C_{\max}(\pi)=\max_{i=1,2,\ldots,n}d_{[j],\,m}=d_{[n],\,m}$. Calculating the $C_{\max}(\pi)$ has a complexity of $O(mn)$.

3. The presented constructive heuristics

3.1. The presented simple constructive heuristics

This section presents two simple constructive heuristics based on the well-known profile fitting (PF) approach of McCormick et al. [11], which tries to sequence jobs for resulting in a minimum sum of the idle and blocking time on machines. In the presented heuristics, we consider the idle and blocking time on different machines with different weights, and the effect of the selection of the current job on the starting and completion time of later jobs.

The PF heuristic can be described as follows. In order to append a job to a partial sequence $\beta=(\beta_1,\beta_2, \ldots, \beta_k)$ containing k jobs already scheduled, all the jobs in the set of the unscheduled jobs U are examined. For each job j from U, we compute the idle time that a machine spends between processing the last job in β and the job j, and the blocking time that the job j causes, as if the job j were appended to β and became the $(k+1)$th job of β (see the example in Fig. 1). Then the job with the smallest sum of idle and blocking time on all machines is chosen. However, the original PF heuristic does not define how to select the first job for the partial sequence β. We follow Ribas et al. [22] and Ronconi [6] to choose the job with the smallest total processing time.

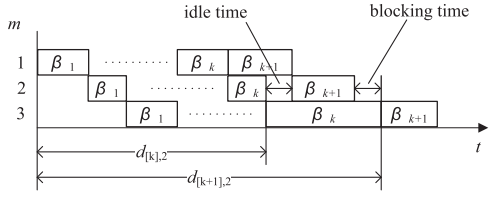

Fig. 1. A partial sequence for a 3-machine blocking flowshop.

The procedure of the PF heuristic is described as follows (see Fig. 2):

Procedure PF
Select the job with the smallest sum of processing time as the first job of a partial sequence, and denote the partial sequence as $\beta=(\beta_1)$.
Let $U=J-\{\beta_1\}$.
% (construct a whole sequence)
for $k:=1$ **to** $n-2$ **do**
Compute the departure time $d_{[k],i}$, $i=1,2, \ldots, m$, for the last job β_k in the partial sequence $\beta=(\beta_1,\beta_2, \ldots, \beta_k)$.
For each job $j\in U$, as if it were appended and became the $(k+1)$th job of β, compute its departure times $d_{[k+1],i}$, for $i=1,2, \ldots, m$, and the sum of idle and blocking times $\delta_{j,k}=\sum_{i=1}^{m}(d_{[k+1],i}-d_{[k],i}-p_{j,i})$.
Select the job resulting in smallest $\delta_{j,k}$ value as the $(k+1)$th job of β, and remove the selected job from U.
endfor
The only one job left in U is appended to β and become the nth job of β.
return

It can be seen that the PF has a loop of $n-2$ iterations and at each iteration the computing can be done in $O(nm)$. So, the complexity of the PF is $O(n^2m)$. The PF heuristic assumes that the idle and blocking time on different machines have the same effect on the makespan criterion. However, from Eqs. (2)–(6), it is clear that the idle and blocking time on the earlier stage machines may lead to more delay in processing successive jobs than it does on the later stage machines. On the other hand, the idle and blocking time caused by the earlier jobs may have larger effect on the makespan value than that by the later jobs. Therefore, we present a weighted PF (wPF) heuristic, which chooses job j for the $(k+1)$th position of the partial sequence $\beta=(\beta_1,\beta_2, \ldots, \beta_k)$

Procedure PF
Select the job with the smallest sum of processing time as the first job of a partial sequence, and denote the partial sequence as $\beta=(\beta_1)$.
Let $U=J-\{\beta_1\}$.
% (construct a whole sequence)
for $k:=1$ **to** $n-2$ **do**
Compute the departure time $d_{[k],i}$, $i=1,2,\ldots,m$, for the last job β_k in the partial sequence $\beta=(\beta_1,\beta_2,\ldots,\beta_k)$.
For each job $j\in U$, as if it were appended and became the $(k+1)^{th}$ job of β, compute its departure times $d_{[k+1],i}$, for $i=1,2,\ldots,m$, and the sum of idle and blocking times $\delta_{j,k}=\sum_{i=1}^{m}(d_{[k+1],i}-d_{[k],i}-p_{j,i})$.
Select the job resulting in smallest $\delta_{j,k}$ value as the $(k+1)^{th}$ job of β, and remove the selected job from U.
endfor
The only one job left in U is appended to β and become the n^{th} job of β.
return

Fig. 2. The PF heuristic.

according to the sum of the weighted idle and blocking time as follows:

$$\delta_{j,k} = \sum_{i=1}^{m} w_i(d_{[k+1],i} - d_{[k],i} - p_{j,i}), \quad (7)$$

where the weight is $w_i = m/(i+k(m-i)/(n-2))$.

This weight differentiates the effect of machines in different stages and jobs in different positions, and it has been used in Liu and Reeves [25] showing very good results. We follow them and no such effort has been devoted to adjusting the weight for the presented heuristic. The procedure of the wPF heuristic is the same as the PF heuristic with the exception that we choose the $(k+1)$th job for β according to $\delta_{j,k}$ calculated by Eq. (7).

The wPF heuristic focuses more on minimizing the idle and blocking time caused by the selected job j, and it does not directly consider the effect of the selected job j on the remaining jobs in U. Obviously, job j affects the starting and completion time of processing the successive jobs, and in turn affects the scheduling criterion. Therefore, we present another heuristic (namely PW) trying to minimize the idle and blocking time and the effects on the starting and completion times of later jobs.

Liu and Reeves [25] presented an effective heuristic for the classic permutation flowshop problem with the criterion of minimizing the total flow time. The heuristic generates a sequence by appending jobs one by one using an index function which includes two parts, one for considering the machine idle time and another for considering the effects on the completion times of later jobs. For the second part of the index function, the authors consider all the other unscheduled jobs in U as an artificial job v as if it were scheduled after job j when job $j \in U$ is considered to be appended to the partial sequence $\beta=(\beta_1,\beta_2, \ldots, \beta_k)$. The processing time of the artificial job v is the average processing time of all these jobs, that is

$$p_{v,i} = \sum_{\substack{q \in U \\ q \neq j}} p_{q,i}/(n-k-1), \quad i = 1,2,\ldots,m. \quad (8)$$

Similar to Liu and Reeves [25], we also use the artificial job v to reflect the other unscheduled jobs and consider to append job v after job j. We first calculate the departure time $d_{[k+2],i}$, $i=1,2, \ldots, m$, for job v. And then the sum of the weighted idle and blocking time caused by job v is given as follows:

$$\chi_{j,k} = \sum_{i=1}^{m} w_i(d_{[k+2],i} - d_{[k+1],i} - p_{v,i}). \quad (9)$$

We use $\chi_{j,k}$ to reflect the effect of job j on later jobs. Combining the idle and blocking time caused by the job j and v, we obtain an index function below:

$$f_{j,k} = (n-k-2)\delta_{j,k} + \chi_{j,k}, \quad (10)$$

where the relative weight $(n-k-2)$ is used to balance the idle and blocking time caused by job j and the effect of job j on later jobs, and $\delta_{j,0} = \Sigma_{i=1}^{m} w_i(d_{[1],i} - p_{j,i})$. This weight comes from the heuristic of Liu and Reeves [25] and shows its effectiveness. In the earlier stage of scheduling, the accuracy of using $\chi_{j,k}$ to reflect the effect on later jobs is poor since job v is the average of many unscheduled jobs. Therefore, a relative large weight is put on the $\delta_{j,k}$. Conversely, as partial sequence β gradually builds up, the number of the unscheduled jobs becomes smaller and $\chi_{j,k}$ reflects the effect on later jobs better. Thus, the relative weight put on the $\delta_{j,k}$ is gradually reduced. With the above index function $f_{j,k}$, we present another heuristic, namely PW. The PW can also be computed in time $O(n^2m)$, and its procedure is described as follows (see Fig. 3).

Procedure PW

Select the job with the smallest $f_{j,0}$ value as the first job of partial sequence $\beta=(\beta_1)$. If there are jobs with the same $f_{j,0}$ value, select the one with the minimum $\chi_{j,0}$ value.

Let $U = J - \{\beta_1\}$.

% (construct a whole sequence)

for $k := 1$ **to** $n-2$ **do**

Compute the departure time $d_{[k],i}$, $i = 1,2,\ldots,m$, for the last job β_k in the partial sequence $\beta=(\beta_1,\beta_2,\ldots,\beta_k)$.

For each job $j \in U$, as if it were appended and became the $(k+1)^{th}$ job of β, calculate the processing time $p_{v,i}$ for the artificial job v, the departure time $d_{[k+1],i}$ of job j and $d_{[k+2],i}$ of the artificial job v, for $i = 1,2,\ldots,m$. Compute the index function $f_{j,k}$.

Select the job resulting in the smallest $f_{j,k}$ value as the $(k+1)^{th}$ job of β (break ties according to minimum $\chi_{j,k}$ value), and remove the selected job from U.

endfor

The only one job left in U is appended to β and become the n^{th} job of β.

return

Fig. 3. The PW heuristic.

3.2. The improvement of the simple constructive heuristics

To better solve the blocking flowshop scheduling problem with makespan criterion, Ronconi [6] suggested a constructive heuristic, called PFE, by combining the PF and NEH heuristics. In this section, we present much more effective methods to combine the PF, wPF, or PW heuristic with the enumeration procedure of the NEH heuristic, and obtain three new constructive heuristics.

The NEH heuristic from Nawaz, Enscore and Ham [13] has been recognized as the highest performing method for the permutation flowshop scheduling problem under the makespan minimization criterion [26]. Initially, the NEH heuristic yields a seed sequence β using the largest processing times (LPT) dispatching rule. Then, the first job in β is taken as the first job of the current sequence π, and the rest jobs in β are taken out one by one and inserted into the best slot of the current sequence π. The procedure of NEH is described as follows (see Fig. 4):

In total, there are $(((n(n+1))/2)-1)$ insertions and each insertion generates a partial sequence. The NEH heuristic has to evaluate all the generated sequences, and its computation complexity is $O(n^2m)$ using the speed-up technology presented in Wang et al. [21]. Note that no problem-specific method is applied to solve the tie cases in the above NEH heuristic, i.e., we choose the first job if two or more jobs result in the same objective value.

The PFE heuristic from Ronconi [6] is a combination of the PF and NEH heuristics, where the PF heuristic is used instead of the LPT rule to produce a seed sequence β for the enumeration procedure of the NEH approach. Since the specific characteristics of the blocking flowshop problem are explored, the PFE generates better results than the original NEH heuristic.

However, in the PFE heuristic, the PF heuristic is only used to generate a list of priorities among jobs or a seed sequence for the NEH enumeration procedure. The specific characteristics of the problem explored by the PF heuristic are not well-utilized in the process of insertion. This combination method may decrease the superiority of the PFE heuristic. To address this problem, we propose another method to combine the PF and NEH heuristics. In the proposed method, only the last λ jobs in the sequence $\beta=(\beta_1,\beta_2,\ldots,\beta_n)$ generated by the PF heuristic, i.e. $\beta_{n-\lambda+1}$, $\beta_{n-\lambda+2}$, …, and β_n, undergo the NEH enumeration procedure, whereas the partial sequence $(\beta_1,\beta_2,\ldots,\beta_{n-\lambda})$ is used as a starting point. In this way, the relative positions of jobs from $(\beta_1,\beta_2,\ldots,\beta_{n-\lambda})$ are not changed as the algorithm progresses. Since these relative positions are generated using the PF heuristic to explore the specific characteristics of the blocking flowshop scheduling problem, the new method utilizes the nature of problem in a better way. Furthermore, the number of evaluations of the new method is less than the original PFE heuristic because it only considers $(((2n-\lambda+1)\lambda)/2)$ partial sequences. The new method gives a total computational complexity of $O(n^2m)$, the same as that of the original PFE heuristic, but the empirical running time is expected to be slightly lower. We denote the new method as PF-NEH, whose procedure is described as follows (see Fig. 5):

It can be seen from Fig. 5 that the PF-NEH heuristic is equal to the PFE heuristic if $\lambda=n-1$, and equal to the PF heuristic if $\lambda=0$. To determine an appropriate λ value for the PF-NEH heuristic, we conduct a simple experiment based on the instances and measures in Section 5.1. In the experiment, we set λ from 0 to 100 with a step equal to 5. And let $\lambda=n-1$ if the number of jobs of an instance, n, is not larger than λ. That is, we perform the PFE heuristic for the instance. This results in 21 different configurations for the PF-NEH heuristic. For each configuration, we run the PF-NEH to solve all the 120 instances, and for each instance we calculate the relative percentage increase (RPI). We use the average RPI (ARPI) over all the instances to show the algorithm's performance in Fig. 6.

It can been seen from Fig. 6 that the PF-NEH heuristic with $\lambda\in[5,100]$ produces much lower ARPI value than with $\lambda=0$, suggesting that the PF-NEH heuristic outperforms the PF heuristic at a substantial margin. It can be also observed that the PF-NEH heuristic presents its best performance at $\lambda=25$. From this point, the performance of the PF-NEH heuristic gradually decreases as λ increases. However, the difference is not large when λ is in the range of [15,35]. This behavior suggests that it is better to keep the relative positions of most jobs generated by the PF heuristic in the NEH enumeration

Procedure NEH

Generate a seed sequence $\beta=(\beta_1,\beta_2,\ldots,\beta_n)$ using the LPT rule.

$\pi:=(\beta_1)$.

for $k:=2$ **to** n **do** %(the NEH enumeration procedure)

Take job β_k from β and test it in all the k possible slots of π.

Inset job β_k in π at the slot resulting in the lowest objective value.

endfor

return π

Fig. 4. The NEH heuristic.

Procedure PF-NEH

Generate a job sequence $\beta=(\beta_1,\beta_2,\ldots,\beta_n)$ using the PF heuristic.

$\pi:=(\beta_1,\beta_2,\ldots,\beta_{n-\lambda})$.

for $k:=n-\lambda+1$ **to** n **do** % (the NEH enumeration procedure)

Take job β_k from β and test it in all the k possible slots of π.

Inset job β_k in π at the slot resulting in the lowest objective value.

endfor

return π

Fig. 5. The PF-NEH heuristic.

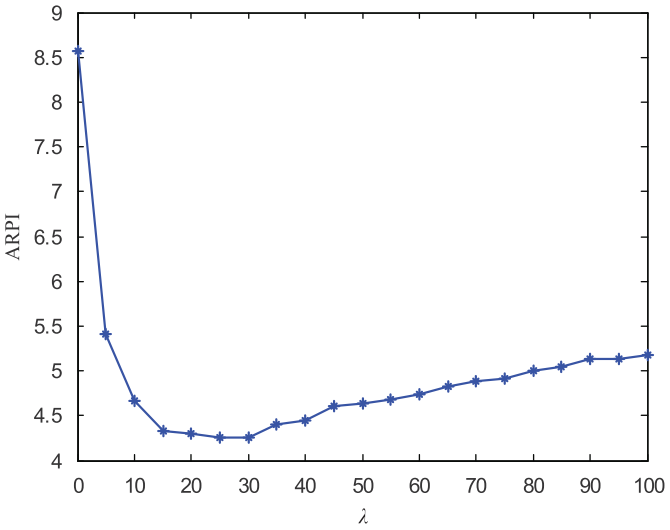

Fig. 6. Behavior of ARPI as a function of λ for the PF-NEH heuristic.

Procedure PF-NEH(x)

Generate a job sequence $\alpha=(\alpha_1,\alpha_2,...,\alpha_n)$ according to their non-decreasing total processing times.

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

Let α_l be the first job and generate a sequence $\beta=(\beta_1,\beta_2,...,\beta_n)$ using the PF heuristic.

$\pi^l:=(\beta_1,\beta_2,...,\beta_{n-\lambda})$.

for $k:=n-\lambda+1$ **to** n **do** % (the NEH enumeration procedure)

Take job β_k from β and test it in all the k possible slots of π^l.

Inset job β_k in π^l at the slot resulting in the lowest objective value.

endfor

endfor

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

Fig. 7. The PF-NEH(x) heuristic.

process for the instances with $n\geq 50$. Furthermore, the PF-NEH heuristic shows some robustness regarding the choice of the parameter λ. Finally, we set $\lambda=25$ for the PF-NEH heuristic in this paper.

From the calculation of makespan, it is clear that the first job of a sequence affects the initial idle time of machines and the starting time to process other jobs. Starting from different jobs, the PF-NEH heuristic may generate quite different sequences. To obtain a good schedule, we can generate a number (from 1 to n) of sequences by using different first jobs, and select the best one as the final result. To make sure that most promising jobs are used as the first jobs in generating sequences, we first rank all the jobs according to the non-decreasing sum of their processing times, and then choose the first jobs according to their ranks. We denote the new version of the PF-NEH heuristic as PF-NEH(x), where $x\in[1,n]$ is the number of sequences to be generated. The PF-NEH(x) heuristic is with computational complexity of $O(xn^2m)$, and its procedure is described as follows (see Fig. 7):

The PF-NEH(x) heuristic does not fix the number of sequences to be generated, and it can be adjusted to the requirements of the problem. So the heuristic is flexible in the computational effort required. Similar to the PF-NEH(x) heuristic, we obtain the wPF-NEH(x) and PW-NEH(x) heuristics, and we set $\lambda=20$ for both the wPF-NEH(x) and PW-NEH(x) heuristics after our initial experiments. The complexity for the presented PF-NEH(x), wPF-NEH(x) and PW-NEH(x) heuristics is $O(xn^2m)$, x being the number of generated sequences.

4. The presented composite heuristics

Recently, many composite heuristics [24–31] that combine a constructive heuristic and local search have been proposed for different scheduling problems. In the composite heuristics, the local search is commonly used as an improvement procedure. More specially, the local search procedure starts from the solution generated by the constructive heuristic and then iteratively moves in the neighbor solutions until a local optimum is found. In this section, we consider to combine the presented heuristics in Section 3 with a local search procedure. For the permutation flowshop scheduling problems, the insertion-based local search is widely used [32], and the referenced local search presented in Pan et al. [33] is very effective and efficient for the classic flowshop scheduling problem. We adopt the referenced local search (RLS) to the blocking flowshop problem considered here. Let $\pi=(\pi_1,\pi_2,\ldots,\pi_n)$ be a

Procedure $RLS(\pi,\pi^{ref})$

do

for $i:=1$ **to** n *do*

$\pi':=\pi$

Find job π_i^{ref} in π' and remove it.

Test π_i^{ref} in all the possible slots of π'.

Inset $\pi^{s}{}_i$ in π' at the slot resulting the lowest makespan value.

if $f(\pi')<f(\pi)$ **then** $\pi:=\pi'$ % ($f(\pi)$ denotes the makespan of π)

endfor

while π is improved

return π

Fig. 8. Referenced local search.

sequence to be improved and $\pi^{ref}=(\pi_1^{ref},\pi_2^{ref},\ldots,\pi_n^{ref})$ a referenced sequence. The procedure of the RLS is given as follows (see Fig. 8):

Wang et al. [21] presented a speed-up technology for evaluating $k+1$ sequences generated by inserting a job j into all the $k+1$ possible slots of a partial sequence $\beta=(\beta_1,\beta_2,\ldots,\beta_k)$. It first calculates the departure times for all the jobs in β and tails, duration between the latest starting time of the operations and the end of the operations, by proceeding from the end of the sequence to the beginning. Then, for a sequence β' generated by inserting job j into position $k'=1,2,3,\ldots,k+1$, the departure times of job j are calculated using the previously calculated departure times, and the makespan is equal to the maximum sum of the departure time of job j and the tail of job $[k']$ in β on the same machine. The computational complexity to evaluate β' is $O(m)$, and it is $O(km)$ to evaluate all the $k+1$ sequences. The speed-up technology can decrease the computational complexity of the above referenced local search to $O(n^2m)$.

With this local search and the presented heuristics PF-NEH(x), wPF-NEH(x), and PW-NEH(x), we can obtain there composite heuristics, i.e., PF-NEH$_{LS}$(x), wPF-NEH$_{LS}$(x), and PW-NEH$_{LS}$(x). In the composite heuristics, the RLS is used to improve each generated sequence. The procedure of the PF-NEH$_{LS}$(x) heuristic is given as follows (see Fig. 9), and the other two can be formed in the identical way. For the computational complexity, all the presented three composite methods can be computed in $O(xn^2m)$.

Procedure PF-NEH$_{LS}(x)$

Generate a job sequence $\alpha=(\alpha_1,\alpha_2,...,\alpha_n)$ according to their non-decreasing total processing times.

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

Let α_l be the first job and generate a sequence $\beta=(\beta_1,\beta_2,...,\beta_n)$ using the PF heuristic.

$\pi^l:=(\beta_1,\beta_2,...,\beta_{n-\lambda})$.

for $k:=n-\lambda+1$ **to** n **do** % (the NEH enumeration procedure)

Take job β_k from β and test it in all the k possible slots of π^l.

Inset job β_k in π^l at the slot resulting in the lowest objective value.

endfor

$\pi^l:=RLS(\pi^l,\pi^l)$ % perform local search to π^l

endfor

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

Fig. 9. The PF-NEH$_{LS}(x)$ heuristic.

5. Numerical simulation and comparisons

We have realized a broad number of heuristics including the PF heuristic presented by McCormick et al. [11], the MM, MME (combination of MM and NEH), and PFE (combination of PF and NEH) heuristics of Ronconi [6], the NEH heuristic of Nawaz et al. [13], and the presented heuristics in this paper. All the heuristics have been coded in VC++ 6.0 and run on a Pentium® 4 CPU 3.0 GHz with 512 MB main memory in Windows XP Operation System. The speed-up technology presented by Wang et al. [21] is employed to save computation time, whenever possible. We use the test bed for flowshop scheduling presented by Taillard [20], 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. The performance measure is the relative percentage increase (RPI) as follows:

$$RPI(c_i)=(c_i-c^*)/c^*\times 100 \quad (11)$$

where c_i is the solution generated by the heuristic i, and c^* is the best-known solution taken from Ribas et al. [22].

Obviously, the smaller the $PRI(c_i)$ is, the better result the heuristic i produces. If $RPI(c_i)$ is less than zero, the best know solution is improved by the heuristic i. In order to better estimate the performance and elapsed CPU time of the heuristics, a total of 10 replications for each instance are carried out. Then the results, averaged across the 10 replications for each instance and grouped for each subset, are reported through Tables 1–6.

5.1. Comparison of simple constructive heuristics

Tables 1 and 2 report the computational results and CPU times of the simple constructive heuristics including the MM, PF, wPF, and PW.

As shown in Table 1, the worst performing algorithm is the MM heuristic with the largest average RPI (ARPI) value equal to 13.03%. The PF outperforms the MM, and achieves an 8.57% increase over the best-known solutions. However, the PF is surpassed by the wPF and PW and the PW performs better than the wPF. On the other side, for the 120 instances, the PF performs better than the MM in 105 cases, and gives equal solutions in 0 cases, and worse in 15 instances. These three values (better than MM, equal to MM, and worse than MM) are 112, 0, and 8 for the wPF, and 116, 1, and 3 for the PW. When comparing to the PF, the better, equal, and worse solutions generated by the wPF and PW heuristics are 74, 4, 42, and 83, 2, 35, respectively. Finally, the PW

Table 1
The RPI of the simple constructive heuristics.

Instance	MM	PF	wPF	PW
20 × 5	13.53	10.90	10.95	9.82
20 × 10	15.99	15.74	13.37	9.54
20 × 20	16.03	15.46	12.11	9.99
50 × 5	12.50	8.20	8.23	7.99
50 × 10	15.11	10.71	9.49	8.40
50 × 20	16.93	15.38	12.31	11.42
100 × 5	10.44	4.78	4.83	4.76
100 × 10	12.82	5.63	5.77	5.09
100 × 20	13.13	8.50	7.46	5.73
200 × 10	10.90	3.15	3.61	3.34
200 × 20	10.83	4.60	3.80	3.52
500 × 20	8.18	−0.18	0.09	−0.10
Average	13.03	8.57	7.67	6.62

Table 2
The CPU time of the simple constructive heuristics (in ms).

Instance	MM	PF	wPF	PW
20 × 5	0.00	0.00	0.31	0.31
20 × 10	0.00	0.16	0.00	0.16
20 × 20	0.32	0.16	0.16	0.78
50 × 5	0.16	0.15	0.63	1.57
50 × 10	0.15	0.32	0.78	2.18
50 × 20	0.47	0.78	1.56	4.53
100 × 5	0.63	0.94	1.88	5.16
100 × 10	0.77	1.25	3.12	8.91
100 × 20	1.57	2.35	6.25	17.34
200 × 10	2.97	4.53	12.03	35.79
200 × 20	6.09	7.96	23.59	68.28
500 × 20	36.25	46.10	144.37	422.97
Average	4.12	5.39	16.22	47.33

Table 3
The RPI of the improved constructive heuristics.

Instance	NEH	MME	PFE	wPFE	PWE	PF-NEH(1)	PF-NEH(2)	PF-NEH(5)	WPF-NEH(1)	WPF-NEH(2)	WPF-NEH(5)	PW-NEH(1)	PW-NEH(2)	PW-NEH(5)
20 × 5	5.31	5.66	5.69	5.64	5.59	5.69	4.84	3.90	5.64	5.27	3.55	5.59	4.84	3.87
20 × 10	5.33	5.49	6.24	5.38	5.53	6.24	4.67	4.06	5.38	4.63	3.96	5.53	4.69	3.63
20 × 20	3.37	3.35	5.55	5.09	4.56	5.55	4.80	3.95	5.09	4.49	3.69	4.56	3.73	2.96
50 × 5	8.75	6.10	6.91	6.84	7.38	5.19	4.97	4.33	5.97	5.08	4.43	5.19	5.05	4.41
50 × 10	7.92	6.71	6.72	6.99	7.27	6.82	6.11	5.48	6.42	5.79	5.22	5.53	5.19	4.48
50 × 20	7.08	5.39	6.77	6.00	6.03	8.10	7.33	6.45	6.21	6.00	5.36	6.18	6.06	5.37
100 × 5	8.51	6.73	6.88	6.57	6.85	2.86	2.36	2.21	2.80	2.70	2.41	3.12	2.92	2.32
100 × 10	7.58	5.86	6.03	6.32	6.57	3.26	3.06	2.45	3.52	3.36	2.69	3.37	3.12	2.66
100 × 20	5.89	4.50	5.14	4.84	4.68	4.90	4.61	4.29	4.20	3.29	3.07	3.54	3.43	3.16
200 × 10	7.74	6.07	6.27	6.98	7.02	1.60	1.42	1.06	1.91	1.73	1.54	1.94	1.65	1.38
200 × 20	5.91	4.63	4.79	5.01	4.73	2.50	2.25	1.94	2.00	1.82	1.56	1.90	1.68	1.62
500 × 20	4.46	3.05	3.07	3.74	3.64	−1.07	−1.22	−1.33	−0.83	−0.93	−1.07	−0.85	−1.09	−1.21
Average	6.49	5.30	5.84	5.78	5.82	4.30	3.77	3.23	4.03	3.60	3.03	3.80	3.44	2.89

Table 4
The CPU time of the improved constructive heuristics (in ms).

Instance	NEH	MME	PFE	wPFE	PWE	PF-NEH(1)	PF-NEH(2)	PF-NEH(5)	WPF-NEH(1)	WPF-NEH(2)	WPF-NEH(5)	PW-NEH(1)	PW-NEH(2)	PW-NEH(5)
20 × 5	0.15	0.16	0.16	0.31	0.32	0.15	0.32	0.78	0.31	0.31	0.78	0.16	0.47	1.41
20 × 10	0.16	0.16	0.31	0.31	0.62	0.16	0.47	0.94	0.15	0.62	1.41	0.47	1.10	2.50
20 × 20	0.31	0.31	0.31	0.47	0.94	0.31	0.62	1.56	0.47	0.94	2.19	0.93	1.71	4.38
50 × 5	0.32	0.63	0.79	0.94	1.72	0.47	0.78	2.19	0.78	1.09	3.28	1.57	2.97	7.18
50 × 10	0.78	1.09	1.09	1.56	3.12	0.78	1.56	3.60	1.25	2.50	6.10	2.65	5.17	13.13
50 × 20	1.40	1.56	1.87	2.97	5.63	1.25	2.66	6.71	2.35	4.69	11.40	5.16	9.99	25.00
100 × 5	1.26	1.72	2.19	3.13	6.40	1.25	2.34	5.94	2.03	4.22	10.63	5.16	10.63	26.40
100 × 10	2.50	3.44	3.90	5.78	11.72	2.19	4.22	10.16	4.06	7.97	19.68	10.00	19.53	48.60
100 × 20	5.00	6.41	7.19	11.09	22.19	3.75	7.50	18.90	7.81	15.46	38.60	18.75	37.50	93.44
200 × 10	10.00	12.97	14.38	22.03	45.78	6.09	12.03	29.84	13.75	27.35	68.59	37.50	74.06	185.00
200 × 20	20.00	26.41	28.12	43.75	88.28	11.41	22.82	56.72	27.04	54.22	135.47	71.72	142.81	356.41
500 × 20	125.63	162.18	172.66	270.16	548.76	55.15	110.15	275.32	153.60	306.87	766.87	432.34	861.87	2151.72
Average	13.96	18.09	19.41	30.21	61.29	6.91	13.79	34.39	17.80	35.52	88.75	48.87	97.32	242.93

Table 5
The RPI values of the composite heuristics.

Instance	NEH_{LS}	MME_{LS}	PFE_{LS}	$wPFE_{LS}$	PWE_{LS}	PF-$NEH_{LS(1)}$	PF-$NEH_{LS(2)}$	PF-$NEH_{LS(5)}$	wPF-$NEH_{LS(1)}$	wPF-$NEH_{LS(2)}$	wPF-$NEH_{LS(5)}$	PW-$NEH_{LS(1)}$	PW-$NEH_{LS(2)}$	PW-$NEH_{LS(5)}$
20 × 5	3.17	3.04	3.15	2.98	3.41	3.15	2.43	1.83	2.98	2.65	2.01	3.41	3.24	2.14
20 × 10	2.90	2.79	2.83	2.50	2.75	2.83	2.33	1.37	2.50	1.66	1.42	2.75	2.30	1.33
20 × 20	2.66	1.90	2.28	2.13	2.01	2.28	1.87	1.17	2.13	1.62	1.47	2.01	1.77	1.21
50 × 5	4.56	4.10	4.18	4.74	4.51	3.71	3.50	3.03	4.03	3.45	2.96	3.60	3.31	2.94
50 × 10	3.99	4.54	4.48	4.59	4.84	3.82	3.23	2.97	3.96	3.41	3.10	3.81	3.41	2.88
50 × 20	3.94	3.39	3.68	3.31	3.52	3.70	3.16	2.80	3.78	3.38	2.62	3.23	3.07	2.62
100 × 5	5.03	4.27	4.09	4.42	4.62	1.63	1.31	1.20	1.74	1.63	1.29	1.97	1.64	1.29
100 × 10	3.92	3.69	3.91	4.13	4.14	1.87	1.56	1.30	1.84	1.68	1.51	2.14	1.80	1.54
100 × 20	2.69	3.16	2.99	3.13	2.82	2.30	2.11	1.99	2.25	1.83	1.47	2.16	1.90	1.71
200 × 10	4.28	4.11	3.97	4.59	4.33	0.65	0.41	0.25	1.07	0.99	0.77	1.10	0.85	0.60
200 × 20	2.72	2.95	3.06	3.10	3.19	1.16	0.81	0.60	0.92	0.77	0.66	0.84	0.67	0.62
500 × 20	1.64	1.67	1.75	1.95	2.05	−1.92	−2.10	−2.15	−1.54	−1.59	−1.74	−1.50	−1.66	−1.76
Average	3.46	3.30	3.36	3.46	3.52	2.10	1.72	1.36	2.14	1.79	1.46	2.13	1.86	1.43

obtains 72 better, 11 equal, and 37 worse solutions than the wPF heuristic.

To check whether the observed differences from Table 1 are indeed statistically different, we carry out a multifactor analysis of variance (ANOVA) where the type of the heuristics is considered as a factor. This method has been used to compare different algorithms in [28,32,34], and among many others. The ANOVA results show that the heuristic type results in statistically significant difference in the response variable RPI at a 95% confidence level. Fig. 10 reports the means and 95% Tukey HSD confidence intervals. Note that an overlapping interval denotes an insignificant difference between the plotted means. From Fig. 10 it is clear that the differences observed from Table 1 are significantly different at a 95% confidence level. This suggests the effectiveness of the presented weight mechanism and the effectiveness of considering the effect of the selection of the current job on later jobs.

From Table 2, it can be seen that all the simple constructive heuristics are very fast. The overall average CPU time of the MM and PF can be negligible. The wPF and PW also have very short CPU time but larger than the MM and PF. For the instances with

Table 6
The CPU time of the composite heuristics (in ms).

Instance	NEH_{LS}	MME_{LS}	PFE_{LS}	$wPFE_{LS}$	PWE_{LS}	PF-$NEH_{LS(1)}$	PF-$NEH_{LS(2)}$	PF-$NEH_{LS(5)}$	wPF-$NEH_{LS(1)}$	wPF-$NEH_{LS(2)}$	wPF-$NEH_{LS(5)}$	PW-$NEH_{LS(1)}$	PW-$NEH_{LS(2)}$	PW-$NEH_{LS(5)}$
20×5	0.15	0.31	0.31	0.31	0.15	0.16	0.32	0.94	0.00	0.62	1.10	0.31	0.63	1.40
20×10	0.31	0.16	0.47	0.31	0.63	0.32	0.46	1.41	0.47	0.63	1.87	0.47	0.78	2.35
20×20	0.47	0.47	0.62	0.63	0.78	0.78	1.41	3.12	0.78	1.40	3.44	0.63	1.72	4.22
50×5	1.09	1.25	1.25	1.25	1.72	0.93	2.03	4.68	0.79	2.03	4.84	1.56	2.97	7.18
50×10	2.03	1.87	2.19	2.50	2.97	2.19	4.06	9.53	1.87	3.76	10.01	2.82	5.63	14.22
50×20	3.60	4.52	4.85	5.78	6.25	4.85	9.22	23.13	4.53	8.74	22.81	5.77	12.81	29.38
100×5	4.68	4.54	4.83	4.69	6.73	3.60	6.72	17.81	3.75	7.34	19.37	5.47	11.25	27.97
100×10	9.06	9.06	10.00	10.79	13.74	7.03	14.69	39.53	8.28	16.88	40.79	11.26	22.34	53.28
100×20	20.32	15.95	22.97	23.28	27.04	17.50	36.25	88.44	20.78	39.84	104.06	24.06	45.32	113.59
200×10	34.38	46.72	47.97	51.72	66.09	33.90	66.40	162.19	34.06	72.66	171.09	45.47	87.81	218.75
200×20	118.12	99.24	92.50	132.97	131.87	64.84	147.66	388.75	75.15	147.03	367.50	102.66	192.50	484.84
500×20	832.81	902.97	794.23	916.87	1068.91	522.35	1082.34	2617.50	576.41	1113.74	2547.19	627.97	1209.69	3372.66
Average	85.58	90.59	81.85	95.92	110.57	54.87	114.30	279.75	60.57	117.89	274.51	69.04	132.79	360.82

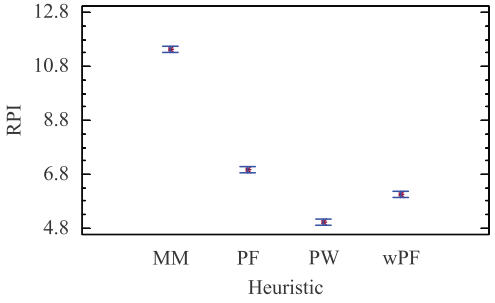

Fig. 10. Means and 95% Tukey HSD Intervals of the type of simple constructive heuristics.

largest size 500×20, the MM and PF are performed in the CPU time of 36.25 and 46.10 ms, respectively, the wPF in 144.37 ms and the PW in 422.97 ms.

5.2. Comparison of improved constructive heuristics

Tables 3 and 4 report the RPI values and CPU time of the NEH, MME, PFE, wPFE, PWE, PF-NEH(x), wPF-NEH(x), and PW-NEH(x) heuristics, where x is equal to 1, 2, and 5, respectively. Like the PFE heuristic, we realize the wPFE and PWE using the wPF and PW instead of the LPT rule to generate seed sequence for the NEH enumeration procedure.

It can be seen from Table 3 that overall the NEH, MME, PFE, wPFE, and PWE are the five worst performing heuristics with the ARPI values equal to 6.49%, 5.30%, 5.84%, 5.78%, and 5.82%, respectively. Although the PFE, wPFE, and PWE produce better ARPI values than the pure PF, wPFE, and PW, respectively, their performance on the instances with sizes 100×5, 100×10, 200×10, 200×20, and 500×20 is really worse. The PF-NEH(1), wPF-NEH(1), and PW-NEH(1) improve the results of the PFE, wPFE, and PWE for most instances and generated much better RPI values than the pure PF, wPFE, and PW for all the subsets. However, for instance sizes 50×10 and 50×20, the PF-NEH(1) performs worse than the PFE, and the wPF-NEH(1) and PW-NEH(1) are ,respectively, surpassed by the wPFE and PWE for instance size 50×20. More specially, the PF-NEH(1) outperforms PFE in 72 cases, and produces equal solutions in 30 cases, and worse in all 18 remaining instances. The wPF-NEH(1) produces 80 better, 25 equal, and 15 worse solutions than the wPFE, whereas the PW-NEH(1) yields 81 better, 30 equal, and 9 worse solutions than the PWE. For the PF-NEH(1) against PF, wPF-NEH(1) against wPF, and PW-NEH(1) against PW, these three values (better, equal, and worse) are 118, 0, 2, and 115, 0, 5, and 115, 0, 5, respectively. So we can conclude that on average the presented PF-NEH(1), wPF-NEH(1), and PW-NEH(1) performs better than the PFE, wPFE, and PWE, respectively. This indicates that the superiority of the combination method should be attributed to the appropriate balance between the problem-based heuristics (PF, wPF, and PW) and the NEH enumeration procedure.

The MME is the winner among the existing heuristics with 5.30% ARPI. When compared with the presented PF-NEH(1), wPF-NEH(1), and PW-NEH(1), the better, equal, and worse solutions generated by the MME are 42, 1, 77, and 39, 0, 81, and 29, 1, 90, respectively. And, on average, the MME performs much worse than the PF-NEH(1), wPF-NEH(1), and PW-NEH(1) in terms of the ARPI values.

Regarding to the PF-NEH(x), wPF-NEH(x), and PW-NEH(x), the ARPI value decreases as the value of parameter x increases. The PF-NEH(5), wPF-NEH(5), and PW-NEH(5) are three best performing heuristics with the PW-NEH(5) producing the smallest ARPI values equal to 2.89%. The other two heuristics achieve 3.23% (PF-NEH(5)) and 3.03% (wPF-NEH(5)) ARPI. The similar behavior can be observed by comparing the number of better solutions generated by the above heuristics. This suggests the effectiveness of using different first jobs to generate a number of sequences for the PF-NEH(x), wPF-NEH(x), and PW-NEH(x).

We also carry out a multifactor ANOVA and report the means and 95% Tukey HSD confidence intervals of the improved constructive heuristics in Fig. 11. It is clear from Fig. 11 that the PF-NEH(x), wPF-NEH(x), and PW-NEH(x) are significantly better than the other algorithms. In addition, the PF-NEH(x), wPF-NEH(x), and PW-NEH(x) can be divided into seven homogenous groups where no significant differences can be found within each group. These are {PF-NEH(1)}, {wPF-NEH(1)}, {PW-NEH(1), PF-NEH(2)}, {PF-NEH(2), PF-NEH(5)}, {wPF-NEH(2), wPF-NEH(5)}, {wPF-NEH(5), PW-NEH(2)}, and {PW-NEH(5)} with the former having lower performance than the latter.

As for the efficiency of the heuristics, we can see from Table 4 that the overall average CPU time of most algorithms is less than 100 ms. The PW-NEH(5) is the slowest performer which needs 242.93 ms on average for all problem sizes, and 2151.72 ms for the largest instances 500×20. This computational effort is not large and the difference between this heuristic and the others has little relevance in reality. So we can conclude that the presented PF-NEH(x), wPF-NEH(x), and PW-NEH(x), are very effective and efficient for the blocking flowshop scheduling problem with makespan criterion.

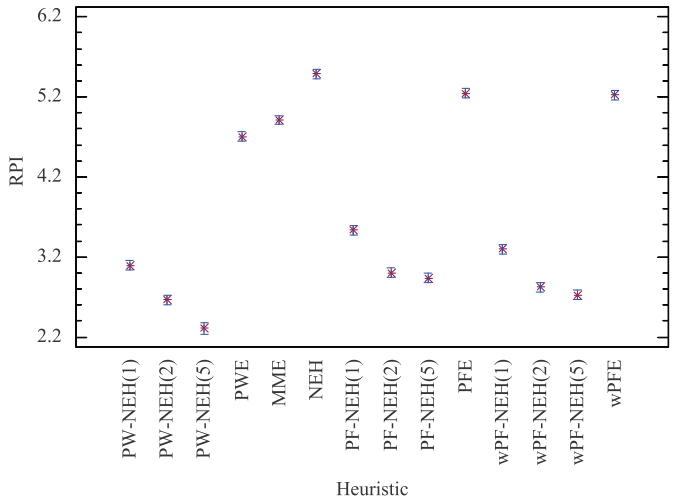

Fig. 11. Means and 95% Tukey HSD Intervals of the improved constructive heuristics.

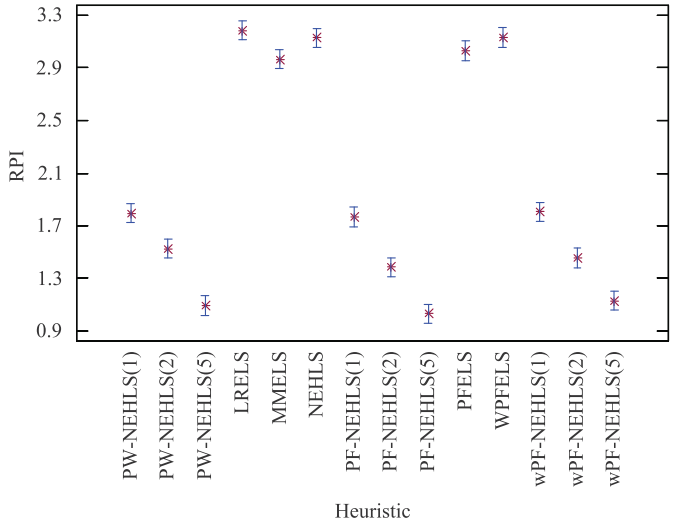

Fig. 12. Means and 95% Tukey HSD Intervals of the composite heuristics.

5.3. Comparison of the composite heuristics

Tables 5 and 6 report the computational results and CPU time of the composite heuristics including PF-NEH$_{LS}(x)$, wPF-NEH$_{LS}(x)$, PW-NEH$_{LS}(x)$, NEH$_{LS}$, MME$_{LS}$, PFE$_{LS}$, wPFE$_{LS}$, and PWE$_{LS}$. For the last five heuristics, we use the sequence generated by the NEH, MME, PFE, wPFE, or PWE respectively, as the starting point for the referenced local search.

It can be seen from Tables 3 and 5 that all the composite algorithms perform much better than their counterparts without local search in terms of the ARPI value. The best three performing algorithms are the PF-NEH$_{LS}(5)$, wPF-NEH$_{LS}(5)$, and PW-NEH$_{LS}(5)$ with the ARPI values equal to 1.36%, 1.46%, and 1.43%, respectively. These results are very encouraging with regard to the very short computational time (less than 400 ms on average), since the IG algorithm [22] found these best-known solutions at the cost of a very long CPU time $30n^2m$ ms for each instance. The worst five heuristics are the NEH$_{LS}$, MME$_{LS}$, PFE$_{LS}$, wPFE$_{LS}$, and PWE$_{LS}$. Their ARPI values are much larger than those of the PF-NEH$_{LS}$, wPF-NEH$_{LS}$, and PW-NEH$_{LS}$. This can be easily explained since the PF-NEH$_{LS}$, wPF-NEH$_{LS}$, and PW-NEH$_{LS}$, respectively, use the PF-NEH, wPF-NEH, and PW-NEH to generate starting sequences for the local search procedure, which perform better than the NEH, MME, PFE, wPFE, and PWE. The similar conclusion can also be drawn by comparing the number of better solutions generated by the heuristics.

Fig. 12 reports the means and 95% Tukey HSD confidence intervals of the composite heuristics. It can be seen from Fig. 12 that the PF-NEH$_{LS}(x)$, wPF-NEH$_{LS}(x)$, and PW-NEH$_{LS}(x)$ are significantly better than the NEH$_{LS}$, MME$_{LS}$, PFE$_{LS}$, wPFE$_{LS}$, and PWE$_{LS}$. No significant difference is observed among the presented PF-NEH$_{LS}(x)$, wPF-NEH$_{LS}(x)$, and PW-NEH$_{LS}(x)$ with the same parameter x values.

New best solutions are found by our composite heuristics in 17 out of 120 Taillard instances. We report the best solutions found so far for all the Taillard instances in Table 7, where the new solutions are in bold and the other solutions are from Ribas et al. [22]. It can be seen from Table 7, that all the solutions of the instances with largest size 500×20 and 5 out of 10 instances with size 200×20 are improved by our heuristics. This indicates the potential of the presented heuristics for larger problems.

In a nutshell, the presented PF-NEH$_{LS}(x)$, wPF-NEH$_{LS}(x)$, and PW-NEH$_{LS}(x)$ are the best performing heuristics for the blocking flowhsop scheduling problem with the objective of minimizing makespan.

Table 7
Best solutions for Taillard's benchmark (new solutions are in bold).

Instance	Best solution	Instance	Best solution	Instance	Best solution	Instance	Best solution	Instance	Best solution	Instance	Best solution
Ta01	1374	Ta21	2436	Ta41	3638	Ta61	6151	Ta81	7844	Ta101	14192
Ta02	1408	Ta22	2234	Ta42	3507	Ta62	6022	Ta82	7894	Ta102	15002
Ta03	1280	Ta23	2479	Ta43	3488	Ta63	5927	Ta83	7794	Ta103	**15161**
Ta04	1448	Ta24	2348	Ta44	3656	Ta64	5772	Ta84	7899	Ta104	15082
Ta05	1341	Ta25	2435	Ta45	3629	Ta65	5960	Ta85	7901	Ta105	**14897**
Ta06	1363	Ta26	2383	Ta46	3621	Ta66	5852	Ta86	7888	Ta106	15101
Ta07	1381	Ta27	2390	Ta47	3696	Ta67	6004	Ta87	7930	Ta107	15099
Ta08	1379	Ta28	2328	Ta48	3572	Ta68	5915	Ta88	8022	Ta108	**15104**
Ta09	1373	Ta29	2363	Ta49	3532	Ta69	6123	Ta89	7969	Ta109	**15026**
Ta10	1283	Ta30	2323	Ta50	3624	Ta70	6165	Ta90	7993	Ta110	**15064**
Ta11	1698	Ta31	3002	Ta51	4500	Ta71	7042	Ta91	13406	Ta111	**35838**
Ta12	1833	Ta32	3201	Ta52	4276	Ta72	6791	Ta92	13313	Ta112	**35970**
Ta13	1659	Ta33	3011	Ta53	4289	Ta73	6936	Ta93	13416	Ta113	**35757**
Ta14	1535	Ta34	3128	Ta54	4377	Ta74	7187	Ta94	13344	Ta114	**36070**
Ta15	1617	Ta35	3166	Ta55	4268	Ta75	6810	Ta95	13360	Ta115	**35869**
Ta16	1590	Ta36	3169	Ta56	4280	Ta76	6666	Ta96	**13133**	Ta116	**36099**
Ta17	1622	Ta37	3013	Ta57	4308	Ta77	6801	Ta97	13598	Ta117	**35760**
Ta18	1731	Ta38	3073	Ta58	4326	Ta78	6874	Ta98	13504	Ta118	**35921**
Ta19	1747	Ta39	2908	Ta59	4316	Ta79	7055	Ta99	13310	Ta119	**35739**
Ta20	1782	Ta40	3120	Ta60	4428	Ta80	6965	Ta100	**13419**	Ta120	**36155**

6. Conclusions

This paper considered the blocking flowshop scheduling problem with the objective of minimizing makespan. This problem has important applications in plastic, steel, chemical, and many other industries where no intermediate buffer exists between machines due to technical requirements or the process characteristics. Two simple constructive heuristics, namely, wPF and PW, were first presented based on the well-known profile fitting (PF) rule. The wPF is a combination of the PF and given weights, which differentiate the effect of machines and position of jobs in the proceeding order, whereas the PW further takes into account parameters of unscheduled jobs. Both wPF and PW produced much better results than the existing MM and PF heuristics. The insertion technique has been considered as one of the most effective methods for scheduling problem. The MME and PFE are two heuristics presented by combining the MM and PF with the NEH insertion procedure. These two heuristics surpass the well-known NEH heuristic since the problem-specific heuristics (MM and PF) exploring specific characteristics of the problem provide a better seed sequence for the NEH enumeration procedure. However, in the MME and PFE, the NEH enumeration procedure is carried out in a greedy fashion which results, in many cases, in solutions far from optimum values. In order to counteract this excessive greediness, we proposed three improved constructive heuristics, i.e., PF-NEH, wPF-NEH, and PW-NEH, by re-inserting a few last jobs of the initial sequences generated by PF, wPF, and PW heuristics, and preserving the relative positions of the remaining most jobs. As expected, the PF-NEH, wPF-NEH, and PW-NEH clearly outperform the MME and PFE by a considerable margin. In addition, the PF-NEH, wPF-NEH, and PW-NEH were further improved by performing several iterations and starting from different jobs. Finally, we presented three composite heuristics by combining the improved constructive heuristic and the referenced local search procedure. The composite heuristics achieved very encouraging relative percentage increase lower than 1.5% over the best-known solutions at a very short CPU time (less than 400 ms on average). Especially, 17 best-known solutions for Taillard benchmarks with large scale were further improved. The presented heuristics have few parameters to be adjusted, which makes them simpler to be used in industrial systems.

The superiority of the presented composite heuristics is mainly due to the fact that they combined some effective techniques including the problem-specific heuristics, the enumeration procedure of the NEH heuristic, and the insertion-based local search method. The appropriate utilization of the NEH enumeration generated solutions with high quality. The insertion-based local search method with speed-up technology further improved the solution in very short time. The flexibility in generating sequences could balance the effectiveness and efficiency of the heuristics according to the requirements of the real problem.

Although we addressed the blocking flowshop scheduling problem with the makespan criterion, the combination methods of the problem-specific heuristics and the NEH enumeration procedure as well as local search might be valid for other kinds of flowshop scheduling problems. The future work is to develop heuristics for other kinds of scheduling problems, and to design meta-heuristics for the blocking flowshop scheduling problem by taking advantage of good initial solutions generated by the presented heuristics.

Acknowledgments

The authors would like to thank the anonymous referees for their valuable and constructive comments and suggestions, which greatly improve this paper. This research is partially supported by National Science Foundation of China under Grants 60874075, 60834004, 70871065, and Science Foundation of Shandong Province, China (BS2010DX005), and Postdoctoral Science Foundation of China (20100480897).

References

[1] Blazewicz J, Ecker KH, Pesch E, et al. Handbook on scheduling: from theory to applications. Springer; 2007.
[2] Pan QK, Ruiz R. An estimation of distribution algorithm for lot-streaming flow shop problems with setup times. OMEGA—The International Journal of Management Science 2011, in press. doi:10.1016/j.omega.2011.05.002.
[3] Vallada E, Ruiz R. Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. OMEGA-The International Journal of Management Science 2010;38(1–2):57–67.
[4] Ruiz-Torres AJ, Ho JC, Ablanedo-Rosas JH. Makespan and workstation utilization minimization in a flowshop with operations flexibility. OMEGA-The International Journal of Management Science 2011;39(3):273–82.
[5] Grabowski J, Pempera J. Sequencing of jobs in some production system. European Journal of Operational Research 2000;125(3):535–50.
[6] Ronconi DP. A note on constructive heuristics for the flowshop problem with blocking. International Journal of Production Economics 2004;87:39–48.
[7] Gong H, Tang L, Duin CW. A two-stage flow shop scheduling problem on batching machine and a discrete machine with blocking and shared setup times. Computers and Operations Research 2010;37(5):960–9.
[8] Hall NG, Sriskandarajah C. A survey of machine scheduling problems with blocking and no-wait in process. Operations Research 1996;44:510–25.
[9] Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 1979;5:287–362.
[10] Gilmore PC, Lawler EL, Shmoys DB. Well-solved special cases. In: Lawler EL, Lenstra KL, Rinooy Kan AHG, Shmoys DB, editors. The traveling salesman problem: a guided tour of combinatorial optimization. Wiley. p. 87–143.
[11] McCormick ST, Pinedo ML, Shenker S, Wolf B. Sequencing in an assembly line with blocking to minimize cycle time. Operations Research 1989;37:925–36.
[12] Leisten R. Flowshop sequencing problems with limited buffer storage. International Journal of Production Research 1990;28:2085–100.
[13] Nawaz M, Enscore EEJ, Ham I. A heuristic algorithm for the *m*-machine, *n*-job flow shop sequencing problem. OMEGA-International Journal of Management Science 1983;11:91–5.
[14] Ronconi DP, Armentano VA. Lower bounding schemes for flowshops with blocking in-process. Journal of the Operational Research Society 2001;52:1289–97.
[15] Abadi INK, Hall NG, Sriskandarajh C. Minimizing cycle time in a blocking flowshop. Operations Research 2000;48:177–80.
[16] Ronconi DP. Henriques LRS. Some heuristic algorithms for total tardiness minimization in a flowshop with blocking. OMEGA-International Journal of Management Science 2009;37(2):272–81.
[17] Caraffa V, Ianes S, Bagchi TP, Sriskandarajah C. Minimizing makespan in a blocking flowshop using genetic algorithms. International Journal of Production Economics 2001;70:101–15.
[18] Grabowski J, Pempera J. The permutation flow shop problem with blocking. A tabu search approach. OMEGA-International Journal of Management Science 2007;35:302–11.
[19] Ronconi DP. A branch-and-bound algorithm to minimize the makespan in a flowshop problem with blocking. Annals of Operations Research 2005;138(1):53–65.
[20] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64:278–85.
[21] Wang L, Pan QK, Suganthan PN, Wang WH, Wang YM. A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. Computers and Operations Research 2010;37(3):509–20.
[22] Ribas I, Companys R, Tort-Martorell X. An iterated greedy algorithm for the flowshop scheduling with blocking. OMEGA-The international Journal of Management Science 2011;39:293–301.
[23] Li X, Wang Q, Wu C. Efficient composite heuristics for total flowtime minimization in permutation flow shops. OMEGA-The International Journal of Management Science 2009;37:155–64.
[24] Smutnicki C. A two-machine permutation flow shop scheduling problem with buffers. OR Spectrum 1998;20(4):229–35.
[25] Liu JY, Reeves CR. Constructive and composite heuristic solutions to the P//SCi scheduling problem. European Journal of Operational Research 2001;132(2):439–52.
[26] Farahmnad RS, Ruiz R, Boroojerdian N. New high performing heuristics for minimizing makespan in permutation flowshops. OMEGA-The international Journal of Management Science 2009;37:331–45.
[27] 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.
[28] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 2005;165:479–94.
[29] Framinan JM, Leisten R, Ruiz-Usano R. Comparison of heuristics for flowtime minimisation in permutation flowshops. Computers & Operations Research 2005;32(5):1237–54.

[30] 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.
[31] Laha D, Sarin SC. A heuristic to minimize total flow time in permutation flow shop. OMEGA-The international Journal of Management Science 2009;37:734–9.
[32] Ruiz R, Stutzle T. A simple and effective iterated greedy algorithm for the permutation flowsop scheduling problem. European Journal of Operational Research 2007;177(3):2033–49.
[33] Pan QK, Tasgetiren MF, Liang YC. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Computers and Industrial Engineering 2008;55(4):795–816.
[34] Ruiz R, Maroto C, Alcaraz J. Two new robust genetic algorithms for the flowshop scheduling problem. OMEGA-The international Journal of Management Science 2006;165(2):479–94.