Science in China Series F: Information Sciences

© 2008 SCIENCE IN CHINA PRESS

Springer

www.scichina.com
info.scichina.com
www.springerlink.com

Heuristic for no-wait flow shops with makespan minimization based on total idle-time increments

LI XiaoPing[1,2†] & WU Cheng[3]

[1] School of Computer Science & Engineering, Southeast University, Nanjing 210096, China;
[2] Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, Nanjing 210096, China;
[3] Department of Automation, Tsinghua University, Beijing 100084, China

No-wait flow shops with makespan minimization are classified as NP-hard. In this paper, the optimization objective is equivalently transformed to total idle-time minimization. The independence relationship between tasks is analyzed, and objective increment properties are established for the fundamental operators of the heuristics. The quality of the new schedules generated during a heuristic is judged only by objective increments and not by the traditional method, which computes and compares the objective of a whole schedule. Based on objective increments, the time complexity of the heuristic can be decreased by one order. A seed phase is presented to generate an initial solution according to the transformed objective. Construction and improvement phases are introduced by experimental analysis. The FCH (fast composite heuristic) is proposed and compared with the most effective algorithms currently available for the considered problem. Experimental results show that the effectiveness of the FCH is similar to that of the best methods but requires far less computation time. The FCH can also be efficient in real time scheduling and rescheduling for no-wait flow shops.

no-wait flow shops, heuristic, makespan, Tabu search

1 Introduction

No-wait flow shops are constrained scheduling problems that exist widely in manufacturing systems. The operations of each job should be processed without interruption between consecutive machines; in this way, the start of a job by the first machine must be delayed when necessary so

Received February 27, 2007; accepted April 12, 2008
doi: 10.1007/s11432-008-0080-7
†Corresponding author (email: xpli@seu.edu.cn)
Supported by the National Natural Science Foundation of China (Grant Nos. 60504029 and 60672092), and the National High Technology Research and Development Program of China (863 Program) (Grant No. 2008AA04Z103)

that the job need not wait for processing by subsequent machines. This kind of problem exists in industry settings, such as chemical, metal, and food processing. Modern manufacturing systems, such as just-in-time, flexible manufacturing environments, and robotic cells, can also be modeled as no-wait flow shops[1].

According to Gray & Johnson[2], no-wait flow shops with makespan minimization ($Fm \mid nwt \mid C_{\max}$ for short) are classified as NP-hard. Although heuristics and meta-heuristics are commonly adopted to solve scheduling problems, only a few algorithms have thus far been proposed for this type of problem. The heuristic proposed by Gangadharan and Rajendran[3] (GR for short) and that described by Rajendran[4] (RAJ for short) seem to be the best existing heuristics, outperforming those presented by Bonney & Gundry[5] and King & Spachis[6]. Recently, Aldowiasan and Allahverdi[7] presented six meta-heuristics, three of which are based on simulated annealing, and the other three based on a genetic algorithm. Among the six algorithms, SA2 and GEN2 are the best. They are almost identical in performance and outperform both GR and RAJ. The computation time for each of the six algorithms is less than 10 s when the number of tasks is fewer than 120. Grabowski & Pempera[8] introduced three Tabu search algorithms. Among these, TSM is the best meta-heuristic so far and outperforms even SA2 and GA2. Although meta-heuristics can usually produce better effectiveness than heuristics, these procedures require too much computation time to be acceptable for practice. For a 500×20 Tailard instance[9] (500 tasks and 20 machines), it takes RAJ only 1 s, SA2 less than 5 s, SA2 about 44 s, and TSM more than 10061 s to complete the task. It is desirable to obtain both good effectiveness and high efficiency.

In this paper, a fast iterative composite heuristic is constructed. The OIM (objective increment method) is presented. The objective of a generated solution is obtained by computing the objective increment rather than the whole objective function. The time complexity can thus be decreased by order.

This paper is organized as follows. Section 2 describes no-wait flow shop problems and transforms the optimization objective. Objective increment properties are analyzed in section 3. Section 4 gives the proposed algorithm, and experimental results follow in section 5. Finally, section 6 is devoted to the conclusions.

2 Problem description and objective transformation

A no-wait flow shop is a constrained flow shop scheduling problem with n jobs $\{J_1,\cdots,J_n\}$ being processed on m machines $M_1,\ldots, M_m$. Each job is processed sequentially on m machines without delay between adjacent machines. The start of a job must be delayed on the first machine when necessary so that the job need not wait for processing by the subsequent machines. Let $O_{i,j}$ be the operation of job j $(j=1,2,\cdots,n)$ processed on machine i $(i=1,2,\cdots,m)$, $t_{i,j}$ be the processing time of $O_{i,j}$, and $S_{i,j}$ and $C_{i,j}$ be the starting time and finish time of $O_{i,j}$. The optimization objective is to find the schedule with the minimum makespan; in other words, the objective is to find the job sequence $\pi=(\pi_{[1]},\cdots,\pi_{[n]})$, where $\pi_{[i]}\in\{J_1,\cdots,J_n\}$ is the ith job of π, with minimum $C_{\max}$. Figure 1 depicts the Gantt chart for a six-job, six-machine $Fm \mid nwt \mid C_{\max}$ problem.

Because a job sequence π is a permutation of the n jobs, there are $n!$ schedules in the whole feasible solution space Ω. To illustrate the following properties conveniently, two dummy jobs $\pi_{[0]}$ and $\pi_{[n+1]}$ are introduced to denote the start and end of π. The processing times of the two dummy jobs are zero. The start dummy job is regarded as $\pi_{[0]}$, and the end one is regarded as $\pi_{[n+1]}$ in π. Therefore, π can also be adapted to $(\pi_{[0]}, \pi_{[1]}, \cdots, \pi_{[n]}, \pi_{[n+1]})$. The makespan of π is equal to $C_{m,[n]}$ or $C_{m,[n+1]}$.

Figure 1 indicates that the size of the rectangle surrounded by axis X, axis Y, line $x = C_{\max}$, and line $y = m$ reaches a minimum when $C_{\max}$ is optimized. The size of the rectangle is composed of the sum of the processing time $\sum_{j=1}^{m}\sum_{i=1}^{n} t_{i,j}$ and total idle time. Because $\sum_{j=1}^{m}\sum_{i=1}^{n} t_{i,j}$ is constant for any given instance, $C_{\max}$ minimization can be equivalently transformed to the total idle time minimization; this is represented by the white area in the rectangle of Figure 1.

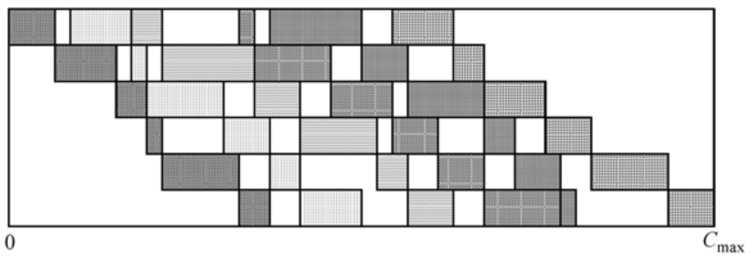

Figure 1 Objective equivalent transformation.

It can be concluded that the total idle time between two adjacent jobs J_i and J_j in a schedule is dependent only on their processing times as follows. Assume J_j starts its process on the first machine just as J_i finishes on the last machine (i.e. $C_{m,i}$). J_j should be shifted left until it is adjacent to J_i. In other words, there exists at least one machine M_k $(1 \leqslant k \leqslant m)$ on which the starting time of $O_{k,j}$ equals the finish time of $C_{k,i}$ (shown as in Figure 2). The left shift length $L_{i,j}$ can be determined.

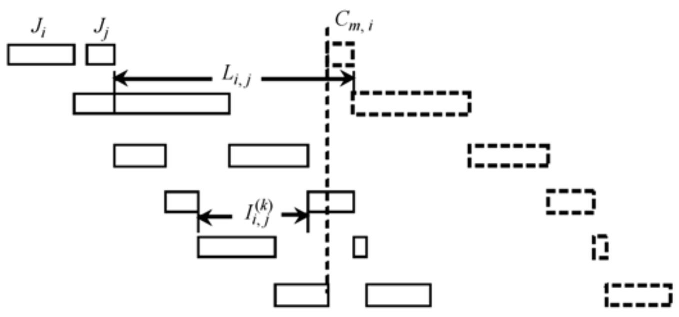

Figure 2 Left shift length between adjacent jobs.

The finish time of J_i on M_k is $C_{k,i} = S_{1,i} + \sum_{h=1}^{k} t_{h,i}$ $(k = 1,\cdots,m)$, and the starting time of J_i on M_1 before shifting is $S_{k,j} = C_{m,i} + \sum_{p=1}^{k} t_{p,j} - t_{k,j}$. From the above analysis, $L_{i,j}$ is actually the minimum among $S_{k,j} - C_{k,i}$ $(k = 1,\cdots,m)$. Thus,

$$L_{i,j} = \min_{k=1,\cdots,m}\left\{\left(S_{1i} + \sum_{h=1}^{m} t_{h,i} + \sum_{h=1}^{k} t_{h,j} - t_{k,j}\right) - \left(S_{1i} + \sum_{h=1}^{k} t_{h,i}\right)\right\}$$
$$= \min_{k=1,\cdots,m}\left\{\sum_{h=k}^{m} t_{h,i} + \sum_{h=1}^{k} t_{h,j} - t_{k,i} - t_{k,j}\right\}. \quad (1)$$

If $I_{i,j}^{(k)}$ is the idle time between two adjacent jobs J_i and J_j on machine M_k and $I_{i,j} = \sum_{k=1}^{m} I_{i,j}^{(k)}$ is the total idle time between J_i and J_j, then

$$I_{i,j}^{(k)} = S_{k,j} - C_{k,i} - L_{i,j} = \sum_{p=k}^{m} t_{p,i} + \sum_{p=1}^{k} t_{p,j} - L_{i,j} - t_{k,i} - t_{k,j}, \quad (2)$$

and

$$I_{i,j} = \sum_{k=1}^{m} I_{i,j}^{(k)} = \sum_{k=2}^{m}\sum_{p=k}^{m} t_{p,i} + \sum_{k=1}^{m-1}\sum_{p=1}^{k} t_{p,j} - mL_{i,j}. \quad (3)$$

From eqs. (1) and (3), the following property is true:

Property 1. The total idle time between two adjacent jobs is dependent only on their processing times and independent of other jobs in no-wait flow shops.

There are $n+1$ pairs of adjacent jobs in $(\pi_{[0]}, \pi_{[1]}, \cdots, \pi_{[n]}, \pi_{[n+1]})$, and these are independent of one another. The total idle time for all the pairs can be determined in advance and stored in matrix *I*. Because $\pi_{[0]}$ cannot be a successor of any other job, there is no column for $\pi_{[0]}$ in *I*. For the same reason, there is also no row for $\pi_{[n+1]}$. Therefore, the size of *I* is $(n+1)\times(n+1)$. $I_{i,i} (i = 1,2,\cdots,m)$ and $I_{[0],[n+1]}$ are set as ∞ for convenience, and *I* is

$$I = \begin{bmatrix} I_{[0],1} & I_{[0],2} & \cdots & I_{[0],n} & \infty \\ \infty & I_{1,2} & \cdots & I_{1,n} & I_{1,[n+1]} \\ I_{2,1} & \infty & \cdots & I_{2,n} & I_{2,[n+1]} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ I_{n,1} & I_{n,2} & \cdots & \infty & I_{n,[n+1]} \end{bmatrix}. \quad (4)$$

The time complexities of computing $L_{i,j}$ and $I_{i,j}$ are $O(m)$. Therefore, the time complexity of *I* is $O(mn^2)$.

For a schedule $(\pi_{[0]}, \pi_{[1]}, \cdots, \pi_{[n]}, \pi_{[n+1]})$ with total idle time $\sum_{k=0}^{n} I_{[k],[k+1]}$, the makespan $C_{\max}$ is

$$C_{\max} = \left(\sum_{k=0}^{n} I_{[k],[k+1]} + \sum_{j=1}^{m}\sum_{i=1}^{n} t_{i,j}\right)\Big/ m. \quad (5)$$

This means that $Fm\,|\,nwt\,|\,C_{\max}$ can be equivalently transformed to find the schedule with $\min_{\pi\in\Omega}\left\{\sum_{k=0}^{n} I_{[k],[k+1]}\right\}$ for no-wait flow shops.

3 Objective increment properties

Heuristics are efficient for scheduling problems[10], in which job insertion and exchange are usually adopted to generate new schedules.

NEH[11] and RZ[12] are fundamental insertions. NEH utilizes incremental job insertion. The first job of the seed sequence, generated by some rule or algorithm, is set as the current solution. The procedure then tries to insert the second job of the seed both before and after the current solution (two possible slots). The better outcome is set as the current solution. The remaining jobs in the seed are sequentially inserted in the same way, such that each job is inserted into every possible slot of the current solution (a schedule/partial schedule) and the best possible new solution generated is selected as the new current solution. After all of the jobs of the seed have been inserted, the current solution is the final solution of NEH. RZ inserts jobs in a different way. The seed is initially assigned as the current solution. Each job of the seed is treated as follows. It is picked out from the current solution and then inserted to all n possible slots. If the best of the possible n newly generated sequences is better than the current solution, it is set as the current solution. This procedure is repeated until all jobs in the seed have been tried. Figure 3 illustrates the NEH and RZ insertion procedures.

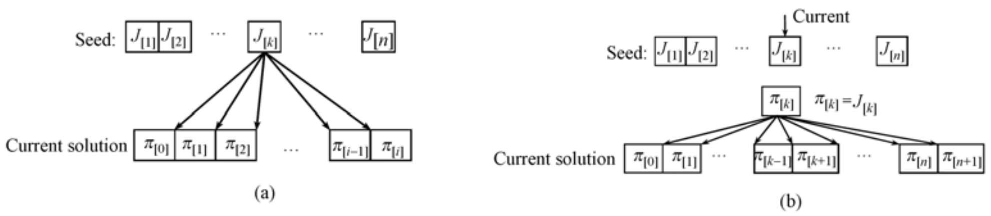
Seed: $J_{[1]}$ $J_{[2]}$ ··· $J_{[k]}$ ··· $J_{[n]}$
Current solution $\pi_{[0]}$ $\pi_{[1]}$ $\pi_{[2]}$ ··· $\pi_{[i-1]}$ $\pi_{[i]}$
(a)
Current
Seed: $J_{[1]}$ $J_{[2]}$ ··· $J_{[k]}$ ··· $J_{[n]}$
$\pi_{[k]}$ $\pi_{[k]}=J_{[k]}$
Current solution $\pi_{[0]}$ $\pi_{[1]}$ ··· $\pi_{[k-1]}$ $\pi_{[k+1]}$ ··· $\pi_{[n]}$ $\pi_{[n+1]}$
(b)

Figure 3 Two insertion methods. (a) NEH-insertion; (b) RZ-insertion.

There are eight exchanges[13], which are all combinations of operation direction (forward or backward), operation manner (swap or insertion), and the next operation position (continue or restart). Forward exchange starts from the first job of a sequence, and backward exchange starts from the last job of a sequence. The job on a position is exchanged with jobs after that position (or inserting it to each of the positions after that position). The operation is successful if the objective can be optimized. The next trial continues from the next position or restarts from the first position of the current solution. Figure 4 depicts the process of forward exchange.

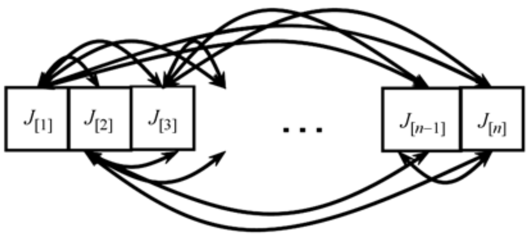

Figure 4 Forward exchange.

According to Property 1, the total idle time between two adjacent jobs is dependent only on their processing times and independent of other jobs. In other words, only the total idle times of the adjacent jobs in the changed positions are different after insertion or exchange within a schedule. The total idle times of the other pairs of adjacent jobs maintain the same value before and after such operations. Therefore, the quality of a new schedule can be evaluated merely by computing objective increments on the operating slots instead of by computing the whole objective value.

Theorem 1. For a *k*-job schedule other than the two dummy jobs, the total idle time increment $\sigma_{[j],i}^{(N)}$ is $I_{[j],i} + I_{i,[j+1]} - I_{[j],[j+1]}$ when an unscheduled job J_i is inserted after $\pi_{[j]}$ $(j = 0, \cdots, k)$.

Proof. For a *k*-job schedule π other than the two dummy ones, $\pi_{[j+1]}$ is no longer the direct successor of $\pi_{[j]}$ once J_i is inserted after $\pi_{[j]}$ $(j = 0, \cdots, k)$. The adjacent pair $(\pi_{[j]}, \pi_{[j+1]})$ is destroyed. Two new pairs $(\pi_{[j]}, J_i)$ and $(J_i, \pi_{[j+1]})$ are created. Therefore, the total idle time increment $\sigma_{[j],i}^{(N)}$ is $I_{[j],i} + I_{i,[j+1]} - I_{[j],[j+1]}$. QED

Similarly, the following theorems are true.

Theorem 2. For an *n*-job schedule π other than the two dummy ones, the total idle time increment $\sigma_{[j],[i]}^{(R)}$ is $I_{[j],[i]} + I_{[i],[j+1]} + I_{[i-1],[i+1]} - (I_{[j],[j+1]} + I_{[i-1],[i]} + I_{[i],[i+1]})$ when $\pi_{[i]}$ $(I = 1, \cdots, n)$ is removed and inserted after $\pi_{[j]}$ $(j = 0, \cdots, n; j \neq i-1)$.

Theorem 3. After exchanging $\pi_{[i]}$ and $\pi_{[j]}$ $(i \neq j,\ i, j = 1, \cdots, k)$ of a *k*-job schedule π other than the two dummy ones, the total idle time increment $\sigma_{[i],[j]}^{(S)}$ is: 1) $I_{[i-1],[j]} + I_{[j],[i]} + I_{[i],[j+1]} - (I_{[i-1],[i]} + I_{[i],[j]} + I_{[j],[j+1]})$ if $\pi_{[i]}$ and $\pi_{[j]}$ were adjacent before the exchange; and 2) $I_{[i-1],[j]} + I_{[j],[i+1]} + I_{[j-1],[i]} + I_{[i],[j+1]} - (I_{[i-1],[i]} + I_{[i],[i+1]} + I_{[j-1],[j]} + I_{[j],[j+1]})$ otherwise.

The following corollary can be inferred from Theorem 3.

Corollary 1. $\sigma_{[i],[j]}^{(S)} \geqslant 0$ is true for any $\pi_{[i]}$ and $\pi_{[j]}$ $(1 \leqslant i < j \leqslant n)$ in the optimum schedule π^*.

Proof. Assume that there exist two jobs $\pi_{[i]}$ and $\pi_{[j]}$ $(1 \leqslant i < j \leqslant n)$ with $\sigma_{[i],[j]}^{(S)} < 0$. The total idle time of π^* can be reduced by exchanging $\pi_{[i]}$ and $\pi_{[j]}$ according to Theorem 3. Thus there exists another schedule that is better than π^*, which contradicts the fact that π^* is the optimum schedule. QED

Heuristics generate new solutions via fundamental operations, such as insertion or exchange, and evaluate the quality of the solutions by calculating their objective values. For the above three theorems, the total idle time increment can be obtained by calculating just a few values. The time complexity of an operator is reduced by one order from $O(n)$ to $O(1)$.

4 FCH algorithm

In this section, a composite heuristic called the FCH (fast composite heuristic) is presented following the three phases given by Framinan, Leisten, and Ruiz-Usano[14]. These three phases include the index development phase, solution construction phase, and solution improvement phase. The FCH is composed of an initial schedule generating procedure, iterative construction procedure, and improvement procedure.

4.1 Initial schedule generating procedure

An initial solution generating algorithm is constructed by integrating properties of no-wait flow shops with NEH, the most efficient heuristic for flow shops with makespan minimization[15]. A schedule χ is produced by the non-increasing job order of the sums of processing time. Figure 1 implies that the first and last jobs play important roles in the optimized objective of a schedule, and thus the total idle time of a schedule is closely dependent on $I_{[0],[1]}$ and $I_{[n],[n+1]}$. Therefore, the first and last jobs are chosen to form schedule $\left(\tau_{[0]}^{(2)},\tau_{[1]}^{(2)},\tau_{[2]}^{(2)},\tau_{[3]}^{(2)}\right)$, in which $\tau_{[0]}^{(2)}$ and $\tau_{[3]}^{(2)}$ are dummy jobs. The job with the minimum value in the first row of I is selected as $\tau_{[1]}^{(2)}$; if there is more than one job with the same value, the first one in χ is selected and removed from χ. The job different from $\tau_{[1]}^{(2)}$ with the minimum value in the last column of I is selected as $\tau_{[2]}^{(2)}$; if there is more than one job with the same value, the first one in χ is selected and removed from χ. k is initialized as 2. The job J_i in the first position of χ is removed. The best insertion slot is found by computing $\min\limits_{j=0,\cdots,k}\left\{\sigma_{[j],i}^{(N)}\right\}$, and $\tau^{(k+1)}$ is constructed by inserting J_i into that slot. $k \leftarrow k+1$. The process is repeated until $k=n$ and $\tau^{(n)}$ is the initial schedule. The initial schedule generating procedure is formally described as follows:

```
1. Generate $\chi$ by ordering the sums of job processing time in a non-increasing manner.
2. Select the job with $\min\limits_{1\leqslant j\leqslant n}\left\{I_{[0],j}\right\}$ as $\tau_{[1]}^{(2)}$ (if more than one job has the same value, select
   the first one in $\chi$ to break the tie). $\chi \leftarrow \chi-\left\{\tau_{[1]}^{(2)}\right\}$. $k \leftarrow 1$.
3. Select the job with $\min\limits_{1\leqslant j\leqslant n}\left\{I_{j,[n+1]}\right\}$ different from $\tau_{[1]}^{(2)}$ as $\tau_{[2]}^{(2)}$ (if more than one job has
   the same value, select the first one in $\chi$ to break the tie). $\chi \leftarrow \chi-\left\{\tau_{[2]}^{(2)}\right\}$. $k \leftarrow k+1$.
4. $\tau^{(2)}=\left(\tau_{[0]}^{(2)},\tau_{[1]}^{(2)},\tau_{[2]}^{(2)},\tau_{[3]}^{(2)}\right)$, in which $\tau_{[0]}^{(2)}$ and $\tau_{[3]}^{(2)}$ are dummy jobs.
5. While $\chi \neq \varnothing$ {
   Remove job $J_i$ in the first position from $\chi$.
   Compute the total idle time increment $\sigma_{[j],i}^{(N)}$ $(j=0,1,\cdots,k)$ by matrix $I$.
   Determine the optimum position $p$ by $\min\limits_{j=0,\cdots,k}\left\{\sigma_{[j],i}^{(N)}\right\}$.
   Generate $\tau^{(k+1)}$ by inserting $J_i$ after $\tau_{[p]}^{(k)}$.
```

```
    $k \leftarrow k+1$;
    }
6. Halt.
```

The time complexity is $O(n\log n)$ for step 1, $O(n)$ for steps 2 and 3, and $O(n^2)$ for step 5. Hence, the time complexity of the initial schedule generating procedure is $O(n^2)$.

4.2 Iterative construction procedure

The NEH-insertion[11] and RZ-insertion[12] are commonly used to construct solutions, but they exhibit distinct performance for different scheduling problems. For example, NEH is based on NEH-insertion. Although it is the most efficient heuristic for flow shops with makespan minimization[15], it is inefficient for $Fm|nwt|C_{\max}$. RZ[12], FL[13], and WY[16] are efficient for constructive heuristics for permutation flow shops with total flowtime minimization; RZ is based on RZ-insertion, whereas FL and WY are based on NEH-insertion. In fact, NEH-insertion and RZ-insertion are generally used in most composite heuristics. Iterative insertion has been demonstrated to improve effectiveness considerably[10].

Various insertion methods exert different influences on computation time and effectiveness within the same problem. Based on Theorems 1—3 and the iterative construction manner proposed in ref. [10], four constructive heuristics (FRZ, FNEH, FWY, and FFL) are presented by modifying NEH, RZ, WY, and FL. In this paper, only FRZ is formally described; its main idea is as follows. π^S is obtained by calling the initial schedule generating procedure. π^C is initialized as π^S. For each job $\pi^S_{[i]}$, let its position in π^C be p. The best inserted slot q in π^C and the total idle time increment Δ can be determined by Theorem 2. If $\Delta < 0$, $\pi^S_{[i]}$ is removed from π^C and inserted after $\pi^C_{[q]}$ $(q=0,\cdots,n; q\neq p-1)$. The new schedule is set as π^C, and $flag \leftarrow True$. If $flag = True$ after all jobs are sequentially inserted by the above RZ-insertion, this implies that the objective becomes smaller by at least one job RZ-insertion. $\pi^S \leftarrow \pi^C$. The above process is repeated until $flag = False$. Experimental results show that the iteration number k seldom exceeds 10 and the improvement is little even if $k>10$. Therefore, the termination condition of FRZ is set as $flag = False$ or $k>10$. FRZ can be formally described as follows:

```
1. $k \leftarrow 0$, $\pi^C \leftarrow \pi^S$.
2. Repeat {
      $flag \leftarrow False$.
      For $i=1$ to $n$ {
         $\Delta \leftarrow 0$.
      For $j=0$ to $n-1$ {
         If $i-1 \neq j$ then {
         Find the position $p$ of $\pi^S_{[i]}$ in $\pi^C$.
         Determine the best inserted slot $q$ of $\pi^S_{[i]}$ in $\pi^C$ and the total idle time increment $\Delta$ by Theorem 2.
```

```
            }
        }
        If $\Delta < 0$ then
            Remove $\pi^C_{[p]}$ and insert it after $\pi^C_{[q]}$. The new schedule is set as $\pi^C$.
            $flag \leftarrow True.$
        }
        $k \leftarrow k+1$.
    }Until ( $flag = False$ or $k > 10$ ).
3. Halt.
```

Obviously, the main computation time of FRZ is spent on step 2, which has time complexity $O(n^2)$. The time complexity of FRZ is thus $O(n^2)$, one order less than that of the traditional RZ with $O(n^3)$.

To demonstrate the effectiveness of the algorithms, the average relative percent deviation (ARPD) is defined as follows:

$$ARPD = \sum_{i=1}^{N} \left(M_i(H) / B_i - 1 \right) / N \times 100, \tag{6}$$

where N is the number of instances for the same problem size (the same number of jobs and machines), $M_i(H)$ is the makespan of instance i obtained by algorithm H, and B_i is the best makespan so far. FNEH, FRZ, FWY, and FL are performed on the 120 Tailard benchmark instances[9], which are classified into 12 groups. ARPD of the four heuristics is depicted in Figure 5. $N = 10$ and B_i provide the minimum makespan among the four for the instance i.

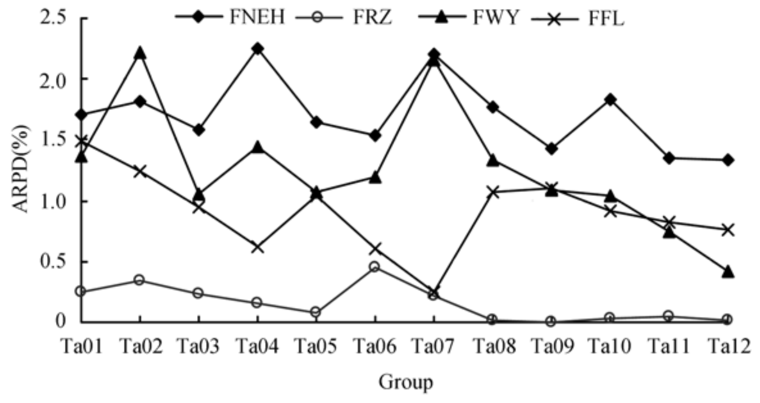

Figure 5 ARPD comparisons for different fast heuristics.

Figure 5 shows that the ARPD of FRZ, with a maximum less than 0.5 and an average of only 0.154, is the lowest of the four heuristics for all the groups. The ARPD of FNEH, with a minimum greater than 1.3 and an average of 1.705, is the largest. Hence, FRZ is the most effective method for the considered problem. It is also indicates that RZ-insertion is more effective than NEH-insertion for $Fm \mid nwt \mid C_{\max}$. This finding verifies the idea that NEH is the most efficient method for permutation flow shops with makespan minimization, whereas it is inefficient for no-wait flow shops with makespan minimization.

4.3 Improvement procedure

The exchange operation is effective for improving the current schedule. There are eight combinations of the operation direction (forward or backward), operation manner (swap or insertion), and next operation position (continue or restart), which are called FIR, FIC, FSR, FSC, BIR, BIC, BSR, and BSC. According to Framinan, Leisten, and Ruiz-Usano[14], solution improvement by the eight exchange operations depends highly on the current solution. Experimentation with the eight operations on the 120 Tailard benchmark instance reveals that: 1) when the eight operations are applied to improve the solution of initial schedule generating procedure, BSC produces the best improvement whereas BIC produces the worst; and 2) when the eight operations are used to improve the solution of FRZ, the ARPD results are given in Table 1.

Table 1 Improvement by eight exchange operations on solutions of FRZ (ARPD: %)

Group	FRZ	FIR	FIC	FSR	FSC	BIR	BIC	BSR	BSC
Ta01	0.160	0.160	0.160	0.000	0.000	0.160	0.160	0.000	0.000
Ta02	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Ta03	0.104	0.104	0.104	0.000	0.000	0.104	0.104	0.000	0.000
Ta04	0.668	0.668	0.668	0.117	0.003	0.668	0.668	0.117	0.003
Ta05	0.027	0.027	0.027	0.002	0.000	0.027	0.027	0.002	0.000
Ta06	0.063	0.063	0.063	0.000	0.000	0.063	0.063	0.000	0.000
Ta07	0.763	0.763	0.763	0.076	0.044	0.763	0.763	0.076	0.020
Ta08	0.142	0.142	0.142	0.035	0.000	0.142	0.142	0.035	0.000
Ta09	0.058	0.058	0.058	0.000	0.000	0.058	0.058	0.000	0.000
Ta10	0.196	0.196	0.196	0.022	0.000	0.196	0.196	0.022	0.000
Ta11	0.084	0.084	0.084	0.000	0.000	0.084	0.084	0.000	0.000
Ta12	0.043	0.043	0.043	0.004	0.000	0.043	0.043	0.004	0.000
Aver.	0.192	0.192	0.192	0.021	0.004	0.192	0.192	0.021	0.002

Table 1 shows that the ARPD of BSC is the lowest, with a value of 0.002. FSC, which has a value of 0.004, is inferior to BSC. Makespans of all the instances except Ta068 are the same as those of BSC. For the other six operations, improvements for both forward and backward are identical, and so the operating direction exerts little influence on the improvement of solutions for FRZ. In this paper, BSC is chosen to improve the final solution π of FRZ. For each of the $n(n-1)/2$ possible new solutions generated by BSC, the total idle times are computed by Theorem 3. The minimum total idle time Δ and the corresponding job pair ($\pi_{[p]}$, $\pi_{[q]}$) are recorded. $\pi_{[p]}$ and $\pi_{[q]}$ are successfully exchanged if $\Delta < 0$. π is replaced with the new schedule, and $flag \leftarrow True$. The above process is repeated until $\Delta \geqslant 0$ or $flag = False$. Additionally, the number of iterations k is less than 20. Therefore, the termination conditions of BSC can be set as $\Delta \geqslant 0$, $flag = False$, or $k > 20$. BSC is formally described as follows:

```
1. $k \leftarrow 0$.
2. Repeat {
        $\Delta \leftarrow 0$, $flag \leftarrow False$.
        For $i = n$ to 2 {
        For $j = i-1$ to 1 {
         Compute $\sigma^{(S)}_{[i],[j]}$ by Theorem 3.
        If $\Delta > \sigma^{(S)}_{[i],[j]}$ then $p \leftarrow i$, $q \leftarrow j$, $\Delta \leftarrow \sigma^{(S)}_{[i],[j]}$.
        }}
        If $\Delta < 0$ then exchange $\pi_{[p]}$ and $\pi_{[q]}$. The new schedule is set as $\pi$.
        $flag \leftarrow True$.
    $k \leftarrow k+1$.
    } Until $flag = False$ or $k > 20$.
3. Halt.
```

Obviously, time complexity of BSC is $O(n^2)$.

4.4 Fast composite heuristic

According to the above analysis, the FCH (fast composite heuristic) is proposed for $Fm \mid nwt \mid C_{\max}$. This is performed as follows:

1. Compute the idle time matrix *I*.
2. Generate the initial schedule via the initial schedule generating procedure.
3. Construct schedule π by FRZ based on the initial schedule.
4. Improve π via BSC.
5. Calculate the makespan by eq. (5).
6. Stop.

The time complexity of step 1 is $O(mn^2)$. It is $O(n^2)$ for steps 2, 3, and 4, and $O(mn)$ for step 5. Therefore, the time complexity for the FCH is $O(mn^2)$.

To illustrate the execution procedure for the FCH, a 7-job and 5-machine no-wait flow shop with the processing times given in Table 2 is considered.

Table 2 A no-wait flow shop instance

	J_1	J_2	J_3	J_4	J_5	J_6	J_7
M_1	64	56	79	36	27	37	54
M_2	80	15	46	40	11	87	54
M_3	50	89	44	30	21	17	67
M_4	73	74	35	7	97	3	68
M_5	73	20	90	74	75	79	45

The execution steps for the FCH are as follows:

1) Compute the idle time matrix *I* by eqs. (3) and (4).

$$I = \begin{bmatrix} 669 & 521 & 577 & 331 & 280 & 446 & 580 & \infty \\ \infty & 127 & 248 & 457 & 311 & 417 & 56 & 691 \\ 174 & \infty & 127 & 261 & 380 & 221 & 100 & 495 \\ 193 & 140 & \infty & 365 & 134 & 325 & 104 & 599 \\ 331 & 183 & 239 & \infty & 102 & 143 & 242 & 417 \\ 293 & 145 & 201 & 410 & \infty & 370 & 204 & 644 \\ 300 & 202 & 133 & 212 & 126 & \infty & 261 & 446 \\ 71 & 173 & 129 & 338 & 332 & 298 & \infty & 572 \end{bmatrix}.$$

2) The initial schedule (0,5,2,3,1,7,6,4,8) is generated by calling the initial schedule generating procedure (0 and 8 are dummy jobs) with a total idle time of 1728. The corresponding makespan $C_{\max}$ is 709.

3) The construction and improvement procedures via FRZ and BSC are shown in Table 3.

4) The makespan of 683 by eq. (5) is obtained, and then the process is stopped.

Table 3 Solution construction and improvement procedures

Operations	π^S	k	*flag*	π^C
FRZ	(0,5,2,3,1,7,6,4,8)	1	true	(0,4,5,2,3,1,7,6,8)
		2	true	(0,4,6,5,2,3,1,7,8)
		3	true	(0,4,6,3,5,2, 1,7,8)
		4	false	(0,4,6,3,5,2, 1,7,8)
BSC	(0,4,6,3,5,2, 1,7,8)	1	true	(0,4,6,3,5,7, 1,2,8)
		2	true	(0,4,6,5,3,7, 1,2,8)
		3	false	(0,4,6,5,3,7, 1,2,8)

5 Experimental results

To compare the proposed FCH with 1) the best existing heuristics GR[3] and RAJ[4], 2) the Annealing Simulation algorithm SA2[7], and 3) the Tabu Search algorithm TSM[8] for $Fm\,|\,nwt\,|\,C_{\max}$, we tested the 120 Tailard benchmark instances. The algorithms are compared in terms of effectiveness and efficiency via the ARPD and computation time (in seconds), respectively. All algorithms are implemented in Visual Basic 6.0 and performed on a PC with 2.5 GHz and 512 M RAM. Our experimental results are shown in Tables 4 and 5.

Table 4 indicates that TSM produces the best in effectiveness. The FCH is similar to TSM. The average ARPD for the FCH (0.642%) is close to that of TSM (0.148%); the difference between these two values is less than 0.5%. The maximum ARPD difference between the FCH and TSM is also no greater than 1%. Overall, the ARPD for the FCH decreases as problem size increases. The average ARPD for RAJ (6.621%) is better than that for GR (7.756%). The ARPD changing tendency of RAJ is similar to that of SA2. GR performs worst in terms of effectiveness, and its ARPD increases with problem size. The ARPD of GR exceeds 10% for instances with sizes greater than 200×10.

Table 4 ARPD (%) comparisons for the algorithms

Group	$n\times m$	RAJ	GR	SA2	TSM	FCH
Ta01	20×5	5.738	3.614	1.735	0.288	1.094
Ta02	20×10	9.072	6.554	1.178	0.398	1.113
Ta03	20×20	6.037	5.251	1.242	0.266	0.665
Ta04	50×5	8.282	6.752	2.178	0.077	0.819
Ta05	50×10	6.225	7.305	1.768	0.011	0.765
Ta06	50×20	5.891	6.953	1.536	0.294	0.685
Ta07	100×5	7.956	7.259	2.020	0.129	0.484
Ta08	100×10	7.306	8.383	2.025	0.142	0.614
Ta09	100×20	6.026	9.272	1.572	0.000	0.534
Ta10	200×10	6.175	8.962	1.863	0.009	0.457
Ta11	200×20	5.525	10.102	1.453	0.106	0.175
Ta12	500×20	5.216	12.662	1.471	0.049	0.297
Aver.		6.621	7.756	1.670	0.148	0.642

Table 5 indicates that the FCH provides the best and TSM the worst efficiency. RAJ is inferior to the FCH. For example, it takes the FCH only 0.430 s on average to compute a 500×20 (Ta12 group) instance. RAJ, GR, and SA2 need 1, 4.408, and 44 s, respectively, to perform the same computation. TSM spends roughly 10062 s (nearly 3 h). The computation time for TSM increases rapidly with problem size, because this method finds better solutions by enlarging the searching scope. The objective increment method has not been adopted for makespan computing.

Table 5 Computation time (in seconds) comparisons for the algorithms

Group	n×m	RAJ	GR	SA2	TSM	FCH
Ta01	20×5	0.000	0.000	0.009	0.569	0.002
Ta02	20×10	0.000	0.002	0.006	0.572	0.002
Ta03	20×20	0.000	0.000	0.010	0.555	0.000
Ta04	50×5	0.002	0.003	0.052	7.567	0.002
Ta05	50×10	0.002	0.008	0.056	7.580	0.000
Ta06	50×20	0.002	0.008	0.055	7.559	0.003
Ta07	100×5	0.009	0.028	0.341	60.064	0.009
Ta08	100×10	0.011	0.030	0.360	60.186	0.011
Ta09	100×20	0.016	0.035	0.348	59.994	0.012
Ta10	200×10	0.061	0.226	2.638	498.816	0.050
Ta11	200×20	0.075	0.236	2.608	504.198	0.056
Ta12	500×20	0.966	4.408	43.905	10061.65	0.430

The proposed FCH is similar to TSM in effectiveness and thus far represents the fastest algorithm for solving the considered problem. The time complexity of the FCH is only $O(mn^2)$. The computation time is very short, even for large scale no-wait flow shops with makespan minimization. Therefore, the FCH is desirable for real-time scheduling and rescheduling no-wait flow shops in practice.

6 Conclusions

No-wait flow shops with makespan minimization were considered in this paper. Makespan minimization is equivalently transformed to the total idle time minimization by analyzing the independence between jobs. An objective increment method was presented for two fundamental heuristic operators, insertion and exchange. Whether newly generated schedules are better than older schedules is judged only by corresponding objective increments rather than via the traditional method of computing whole objectives; this change reduced the time complexity by one order. NEH was adopted as an Initial schedule generating procedure to generate the initial schedule. From simulations, FRZ (a fast iterative heuristic) and BSC were determined as the iterative construction and improvement procedures responsible for constructing and improving the current solution. The FCH (fast composite heuristic) is proposed by integrating the initial schedule generating, iterative construction, and improvement procedures. The time complexity of the FCH is only $O(mn^2)$. The FCH was compared with the efficient or effective existing algorithms RAJ, GR, SA2, and TSM for the considered problem. Our experimental results showed that the FCH performs similar to TSM. It is the best method so far in terms of effectiveness, and it requires little computation time. The ARPD difference between the FCH and TSM is only 0.248% for a 500×20 instance. However, the computation time for the FCH (0.430 s) is far less than that of

TSM (10061.65 s). Therefore, the FCH is desirable for real-time scheduling and rescheduling no-wait flow shops in practice.

The authors are grateful to the anonymous referees for their valuable suggestions and comments.

1 Hall N G, Sriskndarajah C. A survey of machine scheduling problems with blocking and no-wait in process. Oper Res, 1996, 44 (3): 510－525

2 Gray M R, Johnson D S, Sethi R. The complexity of flowshop and jobshop scheduling. Math Oper Res, 1976, 1(2): 117－129

3 Gangadharan R, Rajendran C. Heuristic algorithms for scheduling in the no-wait flowshop. Int J Prod Econ, 1993, 32(3): 285－290

4 Rajendran C. A no-wait flowshop scheduling heuristic to minimize makespan. J Oper Res Soc, 1994, 45(4): 472－478

5 Bonney M C, Gundry S W. Solutions to the constrained flow-shop sequencing problem. Oper Res Q, 1976, 24(4): 869－883

6 King J R, Spachis A S. Heuristics for flow-shop scheduling. Int J Pro Res, 1980, 18(3): 343－357

7 Aldowiasan T, Allahverdi A. New heuristics for no-wait flowshops to minimize makespan. Comp Oper Res, 2003, 30(8): 1219－1231

8 Grabowski J, Pempera J. Some local search algorithms for no-wait flow-shop problem with makespam criterion. Comp Oper Res, 2005, 32(8): 2197－2213

9 Tailard E. Benchmarks for basic scheduling problems. Eur J Oper Res, 1993, 64(2): 278－285

10 Li X P, Wang Q, Wu C. Efficient Composite Heuristics for Total Flowtime Minimization in Permutation Flow Shops. Omega. Accepted. (In press, online)

11 Nawaz M, Enscore E E, Ham I. A heuristic algorithm for the m-machine n-job flow-shop sequencing problem. Omega , 1983, 11(1): 91－95

12 Rajendran C, Ziegler H. An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. Eur J Oper Res, 1997, 103(1): 129－138

13 Framinan J M, Leisten R. An efficient constructive heuristic for flowtime minimisation in permutation flow shops. Omega, 2003, 31(4): 311－317

14 Framinan J M, Leisten R, Ruiz-Usano R. Comparison of heuristics for flowtime minimisation in permutation flowshops. Comp Oper Res, 2005, 32(5): 1237－1254

15 Kalczynski P J, Kamburowski J. On the NEH heuristic for minimizing the makespan in permutation flow shops. Omega, 2007, 35(1): 53－60

16 Woo D S, Yim H S. A heuristic algorithm for mean flowtime objective in flowshop scheduling. Comp Oper Res, 1998, 25(3): 175－182