An effective and efficient heuristic for no-wait flow shop production to minimize total completion time

Honghan Ye[a], Wei (Mike) Li[a,*], Amin Abedini[a], Barrie Nault[b]

[a] *University of Kentucky, Lexington, KY, USA*
[b] *Haskayne School of Business, Calgary, AB, Canada*

Abstract

No-wait flow shop production has been widely applied in manufacturing. However, minimization of total completion time for no-wait flow shop production is *NP*-complete. Consequently, achieving good effectiveness and efficiency is a challenge in no-wait flow shop scheduling, where effectiveness means the deviation from optimal solutions and efficiency means the computational complexity or computation time. We propose a current and future idle time (CFI) constructive heuristic for no-wait flow shop scheduling to minimize total completion time. To improve effectiveness, we take current idle times and future idle times into consideration and use the insertion and neighborhood exchanging techniques. To improve efficiency, we introduce an objective increment method and determine the number of iterations to reduce the computation time. Compared with three recently developed heuristics, our CFI heuristic can achieve greater effectiveness in less computation time based on Taillard′s benchmarks and 600 randomly generated instances. Moreover, using our CFI heuristic for operating room (OR) scheduling, we decrease the average patient flow times by 11.2% over historical ones in University of Kentucky Health Care (UKHC).

Keywords: computational complexity, heuristics, no-wait flow shop, total completion time.

*Corresponding author
Email address: wei.mike.li@uky.edu (Wei (Mike) Li)

1. Introduction

No-wait flow shop production is essential for many processes in manufacturing, such as steel rolling (Aldowaisan and Allahverdi, 2003), metal (Framinan and Nagano, 2008) and food processing (Ye et al., 2016). In these manufacturing processes, all n jobs are processed in the same order on each of m machines, and there should be no waiting time between any two operations until the process is finished. Therefore, the start time on the first machine could be delayed to avoid waiting times during the process. For operating room (OR) scheduling across the three-stage perioperative (periop) process in healthcare systems, patients are also not supposed to wait during the process, especially from ORs in the intraoperative (intraop) stage after the surgery to the postoperative (postop) stage for recovery. Waiting between any stages across the periop process increases patient flow times and generates unnecessary cost. For more details about applications of no-wait flow shop production, please see Hall and Sriskandarajah (1996).

Total completion time (TCT) is defined as the sum of completion times of all jobs on the last machine. Minimization of total completion time, min(TCT), is the same as to minimize the average flow time of products in manufacturing, or to smooth the flow of customers in services (Shyu et al., 2004). Other objectives to optimize the performance of no-wait flow shop production are correlated to minimization of total completion time, such as minimization of maximum completion time or makespan (Lin and Ying, 2016; Grabowski and Pempera, 2005), minimization of total tardiness (Ding et al., 2015), minimization of makespan and total completion time (Laha and Gupta, 2016), and minimization of weighted mean completion time and weighted mean tardiness (Tavakkoli-Moghaddam et al., 2007).

No-wait flow shop production to minimize total completion time can be written as $F_m \mid nwt \mid \sum C_j$, where F_m is for a flow shop problem with m machines, nwt for the constraint of no-wait, and $\sum C_j$ for the objective to minimize total completion time (Graham et al., 1979). $F_m \mid nwt \mid \sum C_j$ problems are N-

P-complete when the number of machines is larger than or equal to two (Röck, 1984). Due to the NP-completeness for $F_m \mid nwt \mid \sum C_j$ problems, it is extremely time consuming to seek optimal solutions by using exact methods even for moderate-scale problems. Therefore, it is practical to seek near-optimal solutions by using heuristics, especially for large-scale problems. Effectiveness and efficiency should be used to evaluate heuristics, where effectiveness means the deviation from the optimum, and efficiency means the computational complexity or computation time (Li et al., 2011a). Currently, few heuristics are both effective and efficient.

To achieve greater effectiveness in less computation time, we propose a current and future idle (CFI) constructive heuristic for no-wait flow shop to min(TCT). The basis of our study is that current and future idle times should be treated differently as introduced in Li et al. (2011b). Consequently, in the initial sequence, we assign higher weights to current idle times generated by jobs in the head of a sequence than those generated by jobs in the tail of the sequence. This initial sequence improves effectiveness of our CFI heuristic. To improve efficiency of our CFI heuristic, we introduce an objective increment method, integrated with the neighborhood exchanging technique. In this method, we calculate the increment of TCT after a pair of jobs in the sequence are exchanged, instead of calculating TCT for the whole sequence after neighborhood exchanging.

The remainder of this paper is organized as follows. Section 2 provides literature review of $F_m \mid nwt \mid \sum C_j$ problems, including discussion of three typical heuristics: PH1(p), FNM and LS heuristics. Section 3 presents our proposed CFI heuristic. Section 4 discusses the computational results based on a large number of instances of various sizes. Section 5 presents the results of case study based on historical data from University of Kentucky Health Care (UKHC). And finally, section 6 draws conclusions and proposes future work.

2. Literature review

This section provides a limited review of no-wait flow shop production to minimize total completion time, as the literature on $F_m \mid nwt \mid \sum C_j$ problems is vast. We firstly define $F_m \mid nwt \mid \sum C_j$ problems with notation and formulations, secondly describe the current status of heuristics for $F_m \mid nwt \mid \sum C_j$ problems in the literature, and finally we illustrate three heuristics that are typical and recently developed.

2.1. Problem description

For n-job m-machine $F_m \mid nwt \mid \sum C_j$ problems, we use the following notation.

- n: the number of jobs;
- m: the number of machines;
- π: a sequence of n jobs, $\pi = [J_1, J_2, \ldots, J_{j-1}, J_j, \ldots, J_n]$;
- $p_{j,i}$: the processing time of job j on machine i, where j=1,…,n and i=1,…,m;
- $C_{j,i}$: the completion time of job j on machine i;
- $d_{j-1,j}$: the distance between the completion times of two adjacent jobs on the last machine.

The following assumptions are used for no-wait flow shop scheduling (Ying et al., 2016). The processing time of job j on machine i, $p_{j,i}$, is deterministic. The set-up times are included in the processing times, and the transportation times between machines are negligible. In addition, all jobs are available to be processed at time zero on the first machine, each job can only be processed once and only once on each machine, each machine can processes only one job at a time, and there is no machine breakdown. Once a job starts to be processed, it cannot be interrupted before completion, which means no preemption. Based on these assumptions, our objective is to find a sequence of jobs that minimizes total completion time.

The completion time of job j on machine i ($C_{j,i}$) can be calculated by Eq. (1) (Ye et al., 2016; Li et al., 2008):

$$C_{j,i} = C_{j-1,m} + \sum_{k=1}^{i} p_{j,k} - \min_{i=1,\ldots,m} \Big(\sum_{k=1}^{i-1} p_{j,k} + \sum_{k=i+1}^{m} p_{j-1,k} \Big), \qquad (1)$$

where $\sum_{k=1}^{0} p_{j,k} = 0$ and $\sum_{k=m+1}^{m} p_{j,k} = 0$. The distance between the completion times of two adjacent jobs on the last machine ($d_{j-1,j}$) can be calculated by Eq. (2):

$$\begin{aligned} d^i_{j-1,j} = C_{j,m} - C_{j-1,m} &= \sum_{k=1}^{m} p_{j,k} - \min_{i=1,\ldots,m} \Big(\sum_{k=1}^{i-1} p_{j,k} + \sum_{k=i+1}^{m} p_{j-1,k} \Big) \\ &= \max_{i=1,\ldots,m} \Big(\sum_{k=1}^{m} p_{j,k} - \sum_{k=1}^{i-1} p_{j,k} - \sum_{k=i+1}^{m} p_{j-1,k} \Big) \\ &= \max_{i=1,\ldots,m} \Big(\sum_{k=1}^{m} p_{j,k} - \sum_{k=i+1}^{m} p_{j-1,k} \Big). \end{aligned} \tag{2}$$

From Eq. (2), the $d_{j-1,j}$ depends only on two adjacent jobs, but not on the positions of other jobs in the sequence, thus a pre-calculated matrix $D_{n\times n}$ can provide values of $d_{j-1,j}$ for any two adjacent jobs (Ying et al., 2016). Although the calculation of distances between two adjacent jobs is not sequence dependent, the calculation of TCT is sequence dependent. Therefore, the total completion time for a given sequence π can be calculated by Eq. (3):

$$\begin{aligned} TCT_\pi &= \sum_{i=1}^{m} p_{\pi(1),i} + \sum_{j=2}^{n} \Big(\sum_{i=1}^{m} p_{\pi(1),i} + \sum_{k=2}^{j} D_{\pi(k-1),\pi(k)} \Big) \\ &= n \sum_{i=1}^{m} p_{\pi(1),i} + \sum_{j=2}^{n} (n-j+1) D_{\pi(j-1),\pi(j)}. \end{aligned} \tag{3}$$

2.2. Current status of heuristics in the literature

According to Lin and Ying (2016), heuristics for $F_m \ |nwt| \ \sum C_j$ problems can be divided into two categories: constructive heuristics and meta-heuristics. In the category of constructive heuristics, Rajendran and Chaudhuri (1990) proposed two heuristics. According to their computational experiments, these two heuristics were more effective than other existing heuristics. Aldowaisan and Allahverdi (2004) proposed six improved heuristics by using three different search methods, first by the same insertion scheme as in the NEH heuristic (Nawaz et al., 1983), second by the same insertion technique as in Rajendran and Ziegler (1997), and third by the adjacent pair-wise neighborhood exchanging

method. The NEH heuristic is considered to be the best constructive heuristic to minimize makespan for permutation flow shop production (Kalczynski and Kamburowski, 2007). Among the six improved heuristics proposed by Aldowaisan and Allahverdi (2004), the PH1(p) heuristic performed significantly better than the heuristic proposed by Rajendran and Chaudhuri (1990) and the genetic algorithm proposed by Chen et al. (1996). Framinan et al. (2010) proposed an FNM constructive heuristic to minimize total completion time, and the results of their case studies showed that the FNM heuristic performed better than the heuristics proposed by Rajendran and Chaudhuri (1990), Aldowaisan and Allahverdi (2004), Bertolissi (2000), and Fink and Voß (2003). Using the constructive procedure as in Laha and Chakraborty (2009), Gao et al. (2013) proposed two constructive heuristics, the improved standard deviation (ISD) heuristic and the improved Bertolissi (IB) heuristic, which were developed from the standard deviation heuristic in Gao et al. (2011) and the Bertolissi heuristic (Bertolissi, 2000), respectively. The results of their case studies showed that the IB heuristic performed better than the NEH (Nawaz et al., 1983) and Bertolissi heuristics (Bertolissi, 2000). Laha et al. (2014) proposed a penalty-shift-insertion (PSI) heuristic for $F_m \mid nwt \mid \sum C_j$ problems, and their computational experiments showed that the PSI heuristic was relatively more effective and efficient than other heuristics in the literature at the time. More recently, Laha and Sapkal (2014) proposed an improved LS heuristic, and results showed that the LS heuristic performed better than the PH1(p) heuristic (Aldowaisan and Allahverdi, 2004) and the FNM heuristic (Framinan et al., 2010).

In the category of meta-heuristics, there are many studies that address $F_m \mid nwt \mid \sum C_j$ problems (Shyu et al., 2004; Ying et al., 2016; Chen et al., 1996; Fink and Voß, 2003; Pan et al., 2008; Akhshabi et al., 2014; Zhu and Li, 2015; Qi et al., 2016). Chen et al. (1996) applied a genetic algorithm. Fink and Voß (2003) proposed various meta-heuristics using different techniques, such as the steepest descent, simulated annealing (SA) and reactive tabu search (RTS). Their experimental results showed that SA and RTS obtained better solutions with a CPU time of 1000 seconds on a 200-job instance. Recently, Ying et al.

(2016) proposed a self-adaptive ruin-and-recreate algorithm to minimize total flow time, and the results showed that their algorithm performed better than other existing algorithms. Qi et al. (2016) proposed a fast local neighborhood search algorithm, and experimental results showed that their algorithm was more effective than major existing algorithms in terms of both quality and robustness. However, in general, it takes much more computation time for meta-heuristics than constructive heuristics to generate near-optimal solutions. This prevents the general application of meta-heuristics for production scheduling in manufacturing. For a more comprehensive review of current heuristics for $F_m|nwt|\sum C_j$ problems, please see Allahverdi (2016).

2.3. Three typical heuristics

2.3.1. The PH1(p) heuristic

The PH1(p) heuristic (Aldowaisan and Allahverdi, 2004) mainly includes three parts: initial sequence generation, NEH insertion and adjacent pair-wise neighborhood exchanging. The steps of the PH1(p) heuristic are described below:

Step 1: Set the position index k=1, the set of sequenced jobs S=$\emptyset$ and the set of unsequenced jobs $U = \{all\ jobs\}$.

Step 2: Choose job j in U such that $\sum_{i=1}^{m} p_{j,i} \leq \sum_{i=1}^{m} p_{h,i}$ where h means any of the rest jobs in U. Remove job j from U and place it in the position k of S.

Step 3: Set k=k+1. Calculate TCT by inserting each job $j \in U$ in the position k in S, remove the job with the minimum TCT from U to put in the position k in S.

Step 4: If k=n, go to Step 5, otherwise return to Step 3.

Step 5: The sequence S is considered as the initial sequence π_0, with a value of total completion time, TCT_0. Set the current best total completion time TCT_b=TCT_0, the current best sequence π_b=π_0, and the number of iterations r=1.

Step 6: Apply NEH (Nawaz et al., 1983) insertion method to the sequence π_{r-1} to produce π_r and calculate TCT_r.

Step 7: If $TCT_r < TCT_b$, update TCT_b=TCT_r and π_b=π_r, otherwise, keep TCT_b and π_b unchanged.

Step 8: Set r=r+1. If r>10 go to Step 9, otherwise go to Step 6.

Step 9: For k=1 to n-1, apply an adjacent pair-wise neighborhood method by exchanging the positions of job k and job k+1 to the sequence π_b to obtain n-1 sequences. If the best sequence among them has a smaller TCT, update TCT_b and π_b.

Step 10: Output the sequence π_b and total completion time TCT_b.

The computational burden of the PH1(p) heuristic is determined by Steps 6-8 where the NEH insertion is applied 10 times. The computational complexity of the NEH insertion is $O(n^3)$ for $F_m \mid nwt \mid \sum C_j$ problems. Hence, the computational complexity of the PH1(p) heuristic is $O(n^3)$. Although the PH1(p) heuristic was significantly more effective than the heuristics proposed by Rajendran and Chaudhuri (1990) and the genetic algorithm proposed by Chen et al. (1996), it requires more computation time.

2.3.2. The FNM heuristic

The FNM heuristic (Framinan et al., 2010) uses the same procedure to generate the initial sequence as the Bertolissi heuristic (Bertolissi, 2000), and improves solutions by using the techniques of neighborhood insertion and neighborhood interchanging. The steps of the FNM heuristic are as follows:

Step 1: Set current best solution π_b=$\emptyset$; for each pair of jobs j, k ($j = 1, \cdots, n; k = 1, \cdots, n; j \neq k$), compute the total completion time of each pair $TCT_{j,k}$=$2p_{j,1}$+$\sum_{i=2}^{m} p_{j,i}$ +$R_{m(j,k)}$, where $R_{m(j,k)}$=$p_{k,m}$+max($R_{m-1(j,k)}$, $\sum_{r=2}^{m} p_{j,r}$), and $R_{1(j,k)}$=$p_{k,1}$.

Step 2: Select the pair of jobs q, s that have the minimum $TCT_{q,s}$, i.e., $TCT_{q,s} \leq TCT_{j,k}$, $\forall$ j, k. Set the first two positions of the partial sequence π_b to q, s, i.e., $\pi_b(1)$=q, and $\pi_b(2)$=s; set j=2.

Step 3: Obtain a job r not in π_b such that $TCT_{\pi_b(j),r} \leq TCT_{\pi_b(j),k}\ \forall\ k$, and append job r to π_b. Insert each job at each position into the rest of positions of π_b, and compute the TCT of the generated sequences. Among them (including π_b), select the sequence with the best TCT as a new sequence π', and then interchange each job at each position with the rest jobs of π'. Set π_b as the sequence with the smallest TCT (including π').

Step 4: Set j=j+1. If j=n, accept π_b as the final sequence, otherwise go to Step 3.

The computational burden of the FNM heuristic is determined by Step 3, where the insertion and interchange techniques are applied. The computational complexity of the FNM heuristic is $O(n^4)$. The results of their case studies showed that the FNM heuristic performed better than the heuristics proposed by Rajendran and Chaudhuri (1990), Aldowaisan and Allahverdi (2004), Bertolissi (2000), and Fink and Voß (2003). However, the computational complexity of the FNM heuristic is higher than other constructive heuristics.

2.3.3. The LS heuristic

The LS heuristic (Laha and Sapkal, 2014) generates the initial sequence by taking bottleneck machines into consideration (Kalir and Sarin, 2001; Rajendran and Alicke, 2007), and the priority of jobs in the initial sequence is given by the sum of their processing times on the bottleneck machines. The solutions are improved by using the same insertion technique as in Laha and Chakraborty (2009). The steps of the LS heuristic are as follows:

Step 1: Set the band width w=1. From machines 1 to m, calculate Sum_w=$\sum_{j=1}^{n}$ $\sum_{i\in w} p_{j,i}$. Set up a set of bottleneck machines $\{B\}$ that consists of machines with the largest Sum_w, and generate a sequence by arranging $\sum_{i\in B} p_{j,i}$ in a non-descending order.

Step 2: For w=2 to m. By including one machine left or right adjacent to $\{B\}$ into the set of bottleneck machines respectively, calculate the two

possible values of Sum_w. Update the set of bottleneck machines that have a larger Sum_w, and generate a sequence by arranging $\sum_{i \in B} p_{j,i}$ in a non-descending order. Take 5-machine flow shop as an example. Assume that as w=1, $\{B\}$=$\{M_4\}$. As w=2, we compare Sum_w based on $\{B\}$=$\{M_3, M_4\}$ and $\{B\}$=$\{M_4, M_5\}$. If w=2 and $\{B\}$=$\{M_3, M_4\}$, as w=3, we compare $\{B\}$=$\{M_2, M_3, M_4\}$ and $\{B\}$=$\{M_3, M_4, M_5\}$. If w=3 and $\{B\}$=$\{M_2, M_3, M_4\}$, as w=4, we compare Sum_w based on $\{B\}$=$\{M_1, M_2, M_3, M_4\}$ and $\{B\}$=$\{M_2, M_3, M_4, M_5\}$. Finally, we sequence jobs by $\sum_{i \in B} p_{j,i}$ and $\{B\}$=$\{all\ machines\}$.

Step 3: A number of m sequences are obtained, and the one with the smallest TCT is selected as the initial sequence. Set the number of iterations r from 1 to 10, and repeat Steps 4 to 7.

Step 4: Set k=1. Select the first two jobs from the initial sequence, obtain the better sequence of two-job partial sequences, and consider it as the current sequence.

Step 5: Set k=k+1. Insert the next two jobs as a block from the initial sequence to each of the 2k-1 possible positions of the current sequence. Select the best partial sequence as the current sequence. Next, place the first job in the block into all possible positions of the current sequence, update the current sequence if the obtained sequence yields a smaller TCT, otherwise keep the current sequence. Place the second job in the block into all possible positions of the current sequence, update the current sequence if the obtained sequence yields a smaller TCT, otherwise keep the current sequence.

Step 6: Repeat Step 5 until all jobs are sequenced. If there is a single job left in the initial sequence, consider it as a block.

Step 7: For j=1 to n-1, insert the jth job in the current sequence into n-j possible positions in the forward direction, which generates $n(n\text{-}1)/2$ sequences. Update the current sequence if the obtained sequence yields a smaller TCT, otherwise keep the current sequence. Consider the current sequence as the initial sequence, and go to Step 8.

Step 8: Set r=r+1. If r≤10, go to Step 4, otherwise, output the current sequence and its TCT.

The computational burden of the LS heuristic is determined by Steps 4-5, where the insertion technique is applied 10 times. Hence, the computational complexity of the LS heuristic is $O(n^3)$. The results of their case studies showed that the LS heuristic performed better than the PH1(p) heuristic (Aldowaisan and Allahverdi, 2004) and the FNM heuristic (Framinan et al., 2010). However, in their case studies, the maximum number of jobs is 70. Thus, more investigation is necessary for large-scale problems.

3. Our CFI heuristic

Our CFI heuristic consists of three phases: phase 1 for initial sequence generation, phase 2 for the insertion and neighborhood exchanging, and phase 3 for iteration improvement. To improve effectiveness of our CFI heuristic, we take both current idle times and future idle times into consideration to generate the initial sequence, and apply the insertion and neighborhood exchanging techniques. To improve efficiency of our CFI heuristic, we introduce an objective increment method to calculate TCT while applying neighborhood exchanging. In addition, we determine the number of iterations as 6 rather than 10 as in the PH1(p) (Aldowaisan and Allahverdi, 2004) and LS (Laha and Sapkal, 2014) heuristics. Six iterations reduce the computation time and maintain effectiveness of our CFI heuristic.

3.1. Initial sequence algorithm (ISA)

While constructing the initial sequence, we assign higher weights to current idle times generated by jobs in the head of the sequence than those generated by jobs in the tail of the sequence. The steps of ISA are as follows:

Step 1: Set the position index k=1, the set of sequenced jobs S=∅ and the set of unsequenced jobs U={*all jobs*}.

Step 2: Select the jth job (denoted as $J_{[j]}$ in U (j=1,$\cdots$,n-k+1), place it into the position k in S, and calculate the average processing time (APT_i) of all jobs in U except the selected $J_{[j]}$ on each machine. Set up an artificial job, and its processing time on each machine equals to APT_i (Liu and Reeves, 2001; Li and Freiheit, 2016). Append this artificial job to $J_{[j]}$, that is the artificial job is located on the (k+1)th position in S.

Step 3: Calculate the idle time between $J_{[j]}$ and the (k-1)th job in S, which is considered as the current idle time $CI(j)$=$\sum_{i=1}^{m}(C_{j,i} - p_{j,i} - C_{k-1,i})$, where $C_{0,i} = 0 \; \forall \; i$. Calculate the idle time between $J_{[j]}$ and the artificial job, which is considered as the future idle time $FI(j)$= $\sum_{i=1}^{m}(C_{k+1,i} - APT_i - C_{k,i})$. The index function $f(j)$=(n-k)$CI(j)$+$FI(j)$ is computed. For j=1,$\cdots$,n-k+1, each job in U has its own index function value, and we remove the job which has the minimum value of $f(j)$ from U and put it into the kth position in S. Set k=k+1.

Step 4: If k<n, go to Step 2, otherwise, append the last one job in U to the last position in S, and output S as the initial sequence π_0.

3.2. Proposed CFI heuristic

We use techniques of insertion and neighborhood exchanging to further improve effectiveness of the CFI heuristic. In addition, an objective increment method is used to calculate the increment of TCT after a pair of jobs in the sequence are exchanged, instead of calculating TCT for the whole sequence after exchanging. The objective increment method reduces the computational complexity of calculating TCT from $O(n)$ to $O(1)$ when using the neighborhood exchanging technique. The steps of the CFI heuristic in our study are as follows:

Step 1: Compute the distance matrix $D_{n\times n}$ and obtain the initial sequence π_0 using ISA. Let TCT_0 be the total completion time of π_0. Set the current best total completion time TCT_b=TCT_0, the current best sequence π_b=π_0, and the number of iterations r from 1 to 6 for Steps 2 to 6.

Step 2: Select first two jobs from π_b, and choose the partial sequence with a smaller TCT.

Step 3: First, apply the NEH insertion technique (Nawaz et al., 1983) to the obtained partial sequences, select the best partial sequence with minimum TCT as current sequence. Next, exchange the position of each job in the current sequence with that of the rest jobs. Among sequences generated by interchanging, the objective increment method is used to calculate ΔTCT. If one sequence yields the smallest negative ΔTCT, set this sequence as the current sequence, otherwise, keep the current one.

Step 4: Repeat Step 3 until all jobs are scheduled, and set the current sequence as π_r with TCT_r.

Step 5: If $TCT_r < TCT_b$, set TCT_b=TCT_r and π_b=π_r.

Step 6: For j=1 to n-1, insert the jth job in π_r into n-j possible positions in the forward direction. If these sequences generate a lower TCT than TCT_b, then update π_b and TCT_b.

Step 7: Update r=r+1. If $r\leq$6, return to Step 2; otherwise, go to Step 8. (Note: the condition $r\leq$6 is concluded from a case study.)

Step 8: Output the final π_b and TCT_b.

While using the neighborhood exchanging technique in Step 3, we introduce an objective increment method to calculate TCT. When two jobs of different positions in the sequence are exchanged, we calculate the increment of TCT based on the positions of these two jobs, instead of calculating TCT for the whole sequence. For example, assume there are five jobs scheduled and the distance matrix $D_{5\times5}$ is computed. For the sequence π=$\{J_1, J_2, J_3, J_4, J_5\}$, TCT_π=$5\sum_{i=1}^{m} p_{1,i} + \sum_{j=2}^{5}(n-j+1)D_{j-1,j}$ using Eq.(3). Let J_1 and J_2 be exchanged, and update the sequence as π'=$\{J_2, J_1, J_3, J_4, J_5\}$. The objective increment is ΔTCT=$5(\sum_{i=1}^{m} p_{2,i} - \sum_{i=1}^{m} p_{1,i})$+$4(D_{2,1} - D_{1,2})$+$3($ $D_{1,3} - D_{2,3})$. Then $TCT_{\pi'} = TCT_\pi + \Delta TCT$. Therefore, using the objective increment method, the TCT can be calculated without computing TCT for the

whole sequence. For the details of our objective increment method, please see Appendix A and a numerical illustration of the CFI heuristic can be found in Appendix B.

The main computational burden of the CFI heuristic is determined by the NEH insertion and neighborhood exchanging techniques in Step 3. The computational complexity for the NEH insertion is $O(n^3)$ including calculating TCT with $O(n)$ when selecting the best insertion position. The computational complexity for neighborhood exchanging technique is also $O(n^3)$ including calculating TCT with $O(1)$ when selecting the best exchanged pair. Therefore, the overall computational complexity of the CFI heuristic is $O(n^3)$, which is the same as that of the PH1(p) and LS heuristics, and less than that of the FNM heuristic.

4. Computational results using benchmarks and generated data

To verify the improvement on effectiveness as a result of the ISA, we compare our CFI heuristic with an alternative version of the CFI heuristic, the CFI-SPT heuristic. The initial sequence in the CFI-SPT heuristic is generated by the shortest processing time (SPT) rule, because SPT rule is good to min(TCT) in general (Li and Freiheit, 2016). Afterwards, we compare our CFI heuristic with the PH1(p) (Aldowaisan and Allahverdi, 2004), the FNM (Framinan et al., 2010), and LS (Laha and Sapkal, 2014) heuristics for solving $F_m \ |nwt| \ \sum C_j$ problems. For effectiveness, we use average relative percentage deviation (ARPD), maximum percentage deviation (MPD), and percentage of the best solutions (PBS) to evaluate the performance of each heuristic based on both small-scale and large-scale instances. ANOVA and paired t-tests are used to statistically verify the improvement on effectiveness based on large-scale instances. To evaluate efficiency, we use the computation times based only on large-scale instances, as the computation times for the small-scale instances are negligible, less than 0.1 second.

For small-scale instances, the number of jobs is 5, 6, 7, or 8, and the number

of machines is 5, 10, 15, 20, or 25. Thus, there are 20 combinations. For each combination, 30 instances are generated randomly, and the processing times for each instance are integers, following a uniform distribution in [1, 99]. In total, there are 600 instances in small-scale.

For large-scale instances, Taillard′s benchmarks (Taillard, 1993) are classic and commonly used to test the performance of heuristics for flow shop scheduling (Ye et al., 2016; Lin and Ying, 2016; Ying et al., 2016; Qi et al., 2016). Taillard′s benchmarks consist of 120 instances in 12 combinations, with 10 instances for each combination, where the number of jobs is 20, 50, 100, 200 or 500, and the number of machines is 5, 10 or 20.

Three criteria are used to evaluate effectiveness of each heuristic (Ye et al., 2016):

(1) Average relative percent deviation (ARPD):

$$ARPD = \frac{1}{N}\sum_{i=1}^{N}\frac{TCT_i(H) - Best_{known_i}}{Best_{known_i}} \times 100$$

(2) Maximum percent deviation (MPD):

$$MPD = \max_{i=1,\ldots,N}\left(\frac{TCT_i(H) - Best_{known_i}}{Best_{known_i}}\right) \times 100$$

$TCT_i(H)$ is the total completion time obtained by heuristic H for an instance i in a combination. N is the number of instances for each combination. N is 30 for small-scale instances but is 10 for large-scale instances. $Best_{known_i}$ is the optimal solution for small-scale instances by using exhaustive enumeration. However, for large-scale instances, the best known solutions are from Qi et al. (2016), who proposed the best known upper bounds for $F_m \mid nwt \mid \sum C_j$ problems based on Taillard′s benchmarks.

(3) Percentage of the best solutions (PBS)

PBS is the percentage of instances for which a heuristic achieves the best performance among the four heuristics. The row total for PBS does not necessarily sum to 100% since some heuristics may tie on the best performance for some instances.

To verify the improvement of initial sequences, we compare our CFI and CFI-SPT heuristics based on ARPD. The improvement is calculated by (ARPD of CFI-SPT − ARPD of CFI)/ARPD of CFI-SPT×100%. The results of this comparison are listed in Table 1. From Table 1, we can see that the ISA improves effectiveness by 33.3% on 600 small-scale instances and 8.8% on 120 large-scale instances, respectively.

Table 1: ARPDs of our CFI and CFI-SPT heuristics (%)

	CFI	CFI-SPT	Improvement
Small-scale	0.06	0.09	33.3
Large-scale	2.08	2.28	8.8

4.1. Small-scale instances

For small-scale instances, the results are shown in Table 2. Our CFI heuristic achieves the best performance on ARPD of 0.06%, on MPD of 2.98%, and on PBS of 88%.

As shown in Table 2 for small-scale instances, with respect to ARPD, the LS heuristic achieves 0.21%, better than the PH1(p) heuristic of 0.37% and FNM heuristic of 0.23%. Our CFI heuristic achieves the smallest ARPD of 0.06% from the optimal. With regard to MPD, the LS heuristic achieves 4.11%, smaller than the PH1(p) of 5.99% and FNM heuristic of 5.61%. Our CFI heuristic also achieves the smallest MPD of 2.98%. With respect to PBS, the LS and FNM heuristics are very close, 75% and 73%, respectively, better than the PH1(p) heuristic of 65%. Our CFI heuristic reaches 88% of the best solutions, 17% improvement over the LS heuristic.

4.2. Large-scale instances

For large-scale instances, the results are shown in Table 3. Our CFI heuristic achieves the best performance on ARPD of 2.08% and on PBS of 53%, but not

Table 2: Average relative and maximum percent deviation (ARPD & MPD) for small-scale instances (%)

Size		PH1(p)		FNM		LS		CFI	
n	m	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD
5	5	0.41	5.94	0.18	2.08	0.18	2.09	0.05	1.36
	10	0.13	1.36	0.09	1.93	0.06	1.36	0.06	0.79
	15	0.27	2.77	0.28	3.38	0.07	0.77	0.01	0.30
	20	0.07	0.43	0.03	0.43	0.01	0.24	0.01	0.24
	25	0.06	1.31	0.01	0.41	0.04	1.31	0.04	1.02
6	5	0.46	3.93	0.24	3.93	0.06	0.75	0.01	0.37
	10	0.38	5.61	0.38	5.61	0.20	2.43	0.01	0.32
	15	0.28	4.11	0.20	2.33	0.26	4.11	0.12	2.33
	20	0.40	3.09	0.16	0.98	0.25	3.09	0.02	0.29
	25	0.16	1.69	0.05	0.93	0.03	0.64	0.00	0.00
7	5	0.51	5.99	0.30	1.95	0.23	1.95	0.03	0.54
	10	0.50	2.60	0.22	1.20	0.31	2.00	0.06	1.82
	15	0.46	4.65	0.14	0.92	0.34	3.71	0.12	1.28
	20	0.49	3.76	0.34	2.54	0.25	2.52	0.08	1.11
	25	0.16	0.98	0.09	0.87	0.21	1.97	0.05	0.85
8	5	0.53	3.07	0.39	3.21	0.35	1.98	0.25	2.98
	10	0.40	3.86	0.34	3.64	0.42	3.64	0.11	1.01
	15	0.61	3.47	0.44	3.47	0.31	2.19	0.13	1.19
	20	0.50	2.77	0.41	1.81	0.32	1.81	0.08	0.69
	25	0.57	2.80	0.28	2.80	0.23	2.32	0.07	0.72
All instances		0.37	5.99	0.23	5.61	0.21	4.11	**0.06**	**2.98**
PBS		65		73		75		**88**	

on MPD of 7.05%. The solutions of our CFI heuristic for each instance in Taillard′s benchmarks can be found in Appendix C.

As shown in Table 3, the PH1(p) heuristic achieves ARPD of 3.47%, MPD of 7.15% and PBS of 7%. The FNM heuristic achieves ARPD of 2.74%, MPD of 5.33% and PBS of 12%, better than the PH1(p) heuristic. The LS heuristic obtains ARPD of 2.33%, MPD of 4.91%, and PBS of 30%, better than the PH1(p) and FNM heuristics on effectiveness. Our CFI heuristic achieves the smallest ARPD of 2.08% and the largest PBS of 53% among all four heuristics, although the MPD of our CFI heuristic is not as good as the MPD from the FNM and LS heuristics.

ARPDs of heuristics for large-scale instances are used to plot the trend of deviations as the number of jobs or machines increases, as shown in Fig 1.

As the number of jobs increases from 20 to 500, Fig 1(a) shows that the

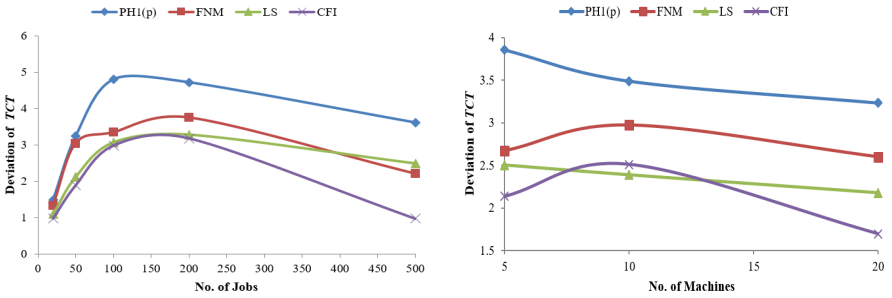
Table 3: Average relative and maximum percent deviation (ARPD & MPD) in Taillard's benchmark (%)

Size		PH1(p)		FNM		LS		CFI	
n	m	ARPD	MPD	ARPD	MPD	ARPD	MPD	ARPD	MPD
20	5	1.60	5.21	1.27	2.55	1.38	3.26	0.87	3.03
	10	1.36	3.25	1.61	3.89	1.06	2.92	1.34	3.52
	20	1.50	4.43	1.13	2.87	0.85	2.30	0.74	1.47
50	5	4.02	6.84	3.31	5.33	2.38	4.91	2.19	4.12
	10	3.03	5.51	3.03	4.76	2.02	3.82	2.14	3.69
	20	2.69	4.35	2.79	4.88	1.97	4.19	1.35	1.91
100	5	5.94	7.15	3.44	4.55	3.75	4.72	3.35	5.35
	10	4.58	5.43	3.41	4.63	2.87	4.73	3.07	7.05
	20	3.89	5.18	3.19	3.97	2.61	3.37	2.53	4.89
200	10	4.98	6.04	3.84	4.16	3.61	4.44	3.49	4.30
	20	4.47	5.12	3.67	4.60	2.96	4.01	2.88	4.38
500	20	3.62	4.86	2.21	2.41	2.50	3.11	0.98	1.78
All instances		3.47	7.15	2.74	5.33	2.33	**4.91**	**2.08**	7.05
PBS		7		12		30		**53**	

deviations of all four heuristics are very close when the number of jobs is 20, and then increase when the number of jobs changes from 20 to 100. In addition, as the number of jobs increases from 100 to 500, the deviation of our CFI heuristic drops the fastest compared with those of other three heuristics.

Fig 1(b) plots the trend of ARPDs against the number of machines, ranging from 5 to 20 machines. The trends obtained by the PH1(p) and LS heuristics go downwards as the number of machines increases from 5 to 15, whereas the deviations of both the FNM and CFI heuristics go up. However, when the number of machines increases from 10 to 20, the deviation of our CFI heuristic drops faster than those of other heuristics and reaches to the lowest point of deviations.

To verify effectiveness of our CFI heuristic, two statistical analyses are conducted based on the large-scale instances. First, the analysis of variance (ANOVA) is used to test whether the ARPDs of the PH1(p), FNM, LS and CFI heuristics are the same or whether some ARPDs are different. The ANOVA results from Table 4 show that the difference of ARPDs among heuristics is



(a) Deviation of TCT by no. of jobs (%) (b) Deviation of TCT by no. of machines (%)

Figure 1: Average deviation from the best performance based on Taillard's benchmarks

statistically significant with p-value=0.000.

Table 4: ANOVA results (95% Confidence Interval)

Source	DF	SS	MS	P
Heuristics	3	134.27	44.76	0.000
Error	476	890.40	1.87	
Total	479	1024.68		

Second, paired t-tests on the ARPD are performed to validate whether or not there are significant differences among the PH1(p), FNM, LS and CFI heuristics. As shown in Table 5, the estimates for mean differences between our CFI heuristic and other heuristics are all smaller than 0, indicating that our CFI heuristic significantly outperforms the other three heuristics at confidence level α=0.05.

Table 5: Paired t-tests results (α=0.05)

CFI vs.	PH1(p)	FNM	LS
p-value	0.000	0.000	0.012
Estimate for mean difference	−1.397	−0.664	−0.2517
95% CI for mean difference	(−1.659,−1.135)	(−0.886,−0.442)	(−0.446,−0.057)

4.3. Computation times

Computation times are used to evaluate efficiency of each heuristic. All four heuristics are programmed in Matlab and run on a Dell Precision T1700 with Intel Core i5-4590 CPUs of 3.3 GHz.

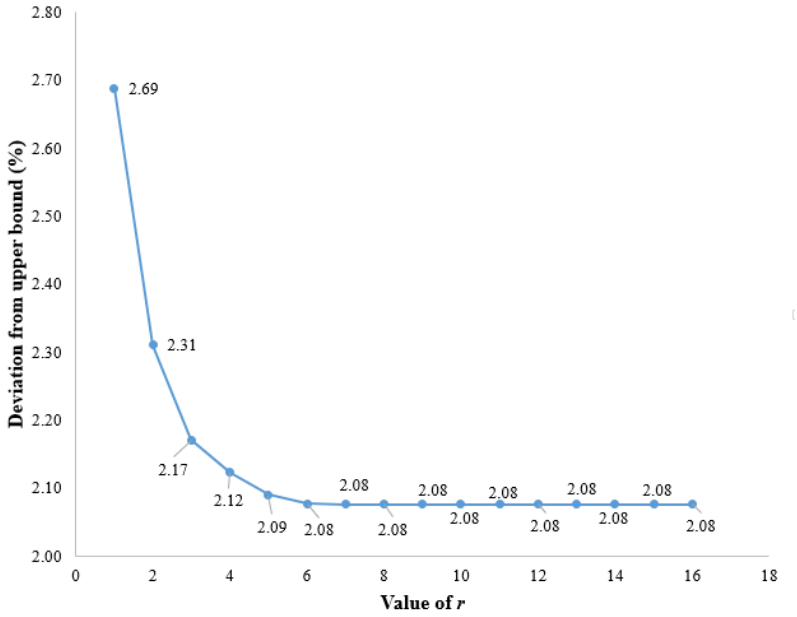


Figure 2: The deviation from upper bound with the value of r

In order to improve efficiency of our CFI heuristic, we determine the number of iterations r by changing r from 1 to 16 based on large-scale instances. We calculate the deviations from upper bound for each value of r. Figure 2 indicates that when the value of r is larger than or equal to 6, the deviation asymptotically reaches to the same level of 2.08%. Therefore, we set the number of iterations r as 6.

The average computation time (in seconds) required for large-scale instances by each heuristic is given in Table 6. On average, our CFI heuristic uses less CPU time than the LS heuristic, but more CPU time than the PH1(p) and FNM heuristics. Although the computational complexity of the FNM heuristic

Table 6: CPU times (in seconds) of four heuristics for large-scale instances

n	m	PH1(p)	FNM	LS	CFI
20	5	0.00	0.02	0.10	0.06
	10	0.00	0.01	0.10	0.05
	20	0.00	0.01	0.09	0.06
50	5	0.02	0.10	0.71	0.44
	10	0.02	0.10	0.72	0.44
	20	0.02	0.10	0.72	0.46
100	5	0.09	0.80	4.10	2.66
	10	0.10	0.83	4.23	2.78
	20	0.12	0.86	4.17	2.81
200	10	0.69	9.24	28.78	19.26
	20	0.77	9.38	28.65	19.51
500	20	10.64	283.49	467.04	319.58
All instances		1.04	25.41	44.95	30.68

is $O(n^4)$, higher than that of the LS and CFI heuristics, respectively, the FNM heuristic takes less computation times, because there are 10 iterations in the LS heuristic and 6 in our CFI heuristic. Overall, the PH1(p) heuristic takes the least computation times among the four heuristics. However, the effectiveness on performance justifies the computation times on efficiency for each heuristic.

5. Case study for OR scheduling using UKHC historical data

To validate our CFI heuristic for OR scheduling across the periop process in a healthcare system, we carry out a case study based on historical OR data from University of Kentucky Health Care (UKHC), in which the first come first serve (FCFS) rule is used for OR scheduling, especially for emergencies. For operating room (OR) scheduling in healthcare systems, the periop process consists of three stages: preoperatives (preop), intraoperatives (intraop), and postoperatives (postop), where the collection of patient information and the preparation for surgeries occur in the preop stage, surgeries occur in operating rooms in the intraop stage, and post-anesthesia care units, intensive care units, or wards for recovery are in the postop stage (Gupta, 2007). Patients are not supposed to wait during the process, especially from the intraop stage to the

postop stage. Therefore, the peri-operative process can be modelled as a three-stage no-wait flow shop.

The historical data set obtained from UKHC consists of almost 30,000 cases in 365 consecutive days from 2013 to 2014. Removing data from weekends and holidays, we have more than 27,000 cases in 50 weeks with 5 days a week, i.e., in 250 days. First, we compare the sequences of the PH1(p), FNM, LS, and CFI heuristics with the UKHC one based on average patient flow time (APFT), which equals to the total completion time divided by the number of patients served in a day. Second, we use statistical process control (SPC) techniques to compare the process capability based on APFTs generated by our CFI heuristic and the actual UKHC data.

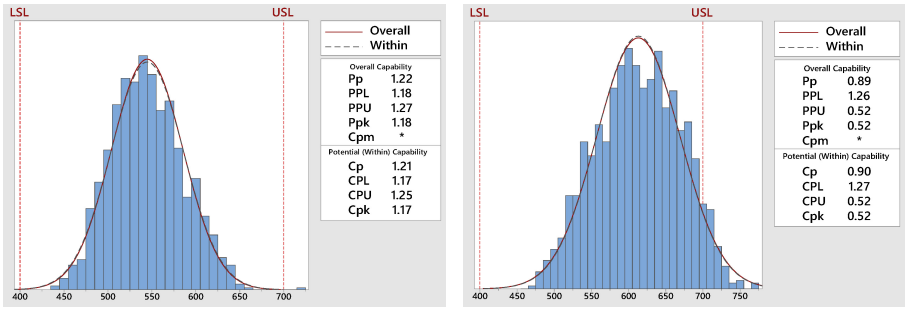
Table 7 shows the APFTs and standard deviations for four heuristics and the actual UKHC data. As shown in Table 7, our CFI heuristic can achieve the smallest APFT with the smallest standard deviation.

Table 7: APFT (minutes) and standard deviation for four heuristics and UKHC

	PH1(p)	FNM	LS	CFI	UKHC
APFT	545.19	544.83	544.91	**544.74**	613.09
Standard deviation	40.91	40.85	40.88	**40.82**	56.20

Using SPC techniques, we generate process capabilities for both CFI and the UKHC data as shown in Figure 3. The user-defined lower specification limit (LSL) and upper specification limit (USL) are set as 400 and 700 minutes, respectively, according to the historical data from UKHC. The process capabilities c_p and c_{pk} are defined as $c_p = \frac{USL-LSL}{6\times\sigma}$ and $c_{pk} = \min\left(\frac{USL-\mu}{3\sigma}, \frac{\mu-LSL}{3\sigma}\right)$, where μ is the average patient flow time and σ is the standard deviation for the process performance (Montgomery, 2007).

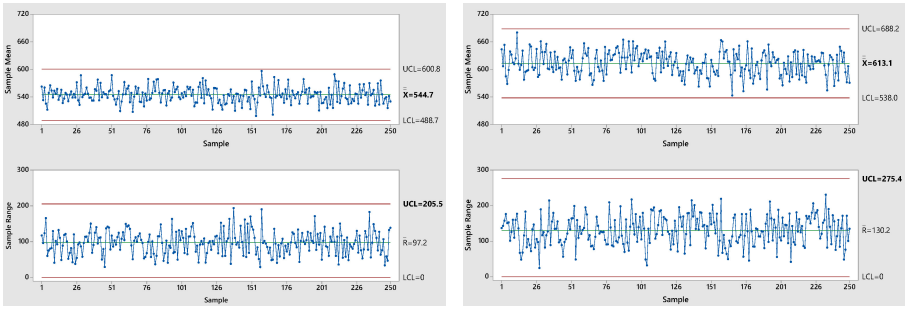
Process capability c_p indicates if the outcomes of a process are within the control limits. With the fixed range of specification limits, which is USL − LSL, the larger the c_p, the less the variation in process, which is 6σ. Process capability index c_{pk} indicates if the outcomes are centered around the average



(a) Process Capability of CFI (b) Process Capability of UKHC

Figure 3: Capability analysis of average patients flow times in 250 days

performance. The larger the c_{pk}, the less likely that the outcomes will drop out of the limits, LSL or USL. As shown in Figure 3, the c_p is 1.21 for our CFI heuristic and 0.90 for the UKHC data, and the c_{pk} is 1.17 for our CFI heuristic and 0.52 for the UKHC data. Obviously, the APFTs generated by our CFI heuristic are more centered within the specification limits and with less variation, compared to those from historical UKHC data.



(a) By CFI (b) By UKHC

Figure 4: Xbar-R charts of average patient flow times

Moreover, we generate the Xbar-R charts based on APFTs in 250 days as shown in Figure 4. The APFT is 544.7 minutes for our CFI heuristic, and 613.1 minutes for the data from UKHC. The improvement on average patient flow times can be calculated by (613.1 − 544.7)/613.1=11.2%. The range of

variation on our CFI heuristic is 205.5 minutes, less than that of 275.4 minutes for the UKHC data.

These results from Xbar-R charts support those of c_p and c_{pk}, and the 11.2% improvement on average patient flow time indicates that potentially 3,000 additional patients could be served in the year if our CFI heuristic was applied for sequencing. However, in reality, OR scheduling and control is affected by many other factors in addition to sequencing, such as emergencies, the availability of patients in the waiting list, surgical staff, and equipment, etc. These realities are reflected in the UKCH data.

6. Conclusion

No-wait flow shop production is common in industry, where no waiting time is allowed between intermediate operations. Minimization of total completion time (TCT) for no-wait flow shop production has been proven to be *NP*-complete. Therefore, heuristics are widely used to find near optimal solutions for production scheduling in manufacturing. The PH1(p), FNM, and LS heuristics are three typical heuristics recently developed in the literature. These heuristics can obtain good solutions in a reasonable time, even for large-scale instances. We propose a CFI heuristic to minimize TCT for no-wait flow shop production. To improve effectiveness, we first take the current idle times and future idle times into consideration, proposing an initial sequence algorithm, and then use the insertion and neighborhood exchanging methods to further improve the solutions. To increase efficiency, we first introduce an objective increment method to reduce the computational complexity from $O(n)$ to $O(1)$ in calculating TCT, and then set the number of iterations in our CFI heuristic to further reduce the computation time.

Compared with the PH1(p), FNM and LS heuristics, based on 600 small-scale instances, our CFI heuristic achieves the best performance on average relative percentage deviation (ARPD), maximum percentage deviation (MPD), and the percentage of the best solutions (PBS). Based on large-scale instances

in Taillard′s benchmarks, our CFI heuristic achieves the best performance on ARPD and PBS, although not on the MPD. In addition, on average, the CPU time of our CFI heuristic is 30.68 seconds, based on Taillard′s benchmarks, less than 44.95 seconds of the LS heuristic.

In a case study using historical data from UKHC, we found our CFI heuristic can achieve 11.2% improvement on average patient flow times over UKHC′s performance, and the average patient flow times generated by our CFI heuristic are under better process control with less variation, which means additional patients can potentially be served and there is a greater control of OR management across the peri-operative process. Overall, our CFI heuristic can achieve good effectiveness and efficiency for no-wait flow shop scheduling.

Variation in processing times is a common disturbance to flow shop production in manufacturing or healthcare systems. Our future research will focus on adaptive control by using variants of our CFI heuristic.

Appendix A. Objective increment method

Assume there is a sequence $\pi=\{\pi_1,\pi_2,\cdots,\pi_{j-1},\pi_j,\cdots,\pi_n\}$, and the corresponding TCT is TCT_π. When π_k and π_j $(0<k<j\leq n)$ in π are exchanged, the new sequence π' is generated, the difference of TCT between π' and π, i.e., $\Delta TCT(k,j)$, can be calculated by one of the following conditions:

- When k=1 and j=2

$$\Delta TCT(k,j)=n\Big(\sum_{i=1}^{m}p_{\pi'_k,i}-\sum_{i=1}^{m}p_{\pi_k,i}\Big)+(n-1)(D_{\pi'_k,\pi'_j}-D_{\pi_k,\pi_j})+(n-2)(D_{\pi'_j,\pi'_{j+1}}-D_{\pi_j,\pi_{j+1}})$$

- When k=1 and j=3,$\cdots$,n-1

$$\Delta TCT(k,j)=n\Big(\sum_{i=1}^{m}p_{\pi'_k,i}-\sum_{i=1}^{m}p_{\pi_k,i}\Big)+(n-1)(D_{\pi'_k,\pi'_{k+1}}-D_{\pi_k,\pi_{k+1}})+(n-j+1)(D_{\pi'_{j-1},\pi'_j}-D_{\pi_{j-1},\pi_j})+(n-j)(D_{\pi'_j,\pi'_{j+1}}-D_{\pi_j,\pi_{j+1}})$$

- When k=1 and j=n

$$\Delta TCT(k,j)=n\Big(\sum_{i=1}^{m}p_{\pi'_k,i}-\sum_{i=1}^{m}p_{\pi_k,i}\Big)+(n-1)(D_{\pi'_k,\pi'_{k+1}}-D_{\pi_k,\pi_{k+1}})+(D_{\pi'_{j-1},\pi'_j}-D_{\pi_{j-1},\pi_j})$$

- When k=2,$\cdots$,n-2 and j=k+1

$$\Delta TCT(k,j)=(n-k+1)(D_{\pi'_{k-1},\pi'_k}-D_{\pi_{k-1}\pi_k})+(n-k)(D_{\pi'_k,\pi'_j}-D_{\pi_k,\pi_j})+(n-j)(D_{\pi'_j,\pi'_{j+1}}-D_{\pi_j,\pi_{j+1}})$$

- When k=2,$\cdots$,n-3 and j=k+2, $\cdots$,n-1

$$\Delta TCT(k,j)=(n-k+1)(D_{\pi'_{k-1},\pi'_k}-D_{\pi_{k-1}\pi_k})+(n-k)(D_{\pi'_k,\pi'_{k+1}}-D_{\pi_k,\pi_{k+1}})+(n-j+1)(D_{\pi'_{j-1},\pi'_j}-D_{\pi_{j-1},\pi_j})+(n-j)(D_{\pi'_j,\pi'_{j+1}}-D_{\pi_j,\pi_{j+1}})$$

- When k=2,$\cdots$,n-2 and j=n

$$\Delta TCT(k,j)=(n-k+1)(D_{\pi'_{k-1},\pi'_k}-D_{\pi_{k-1}\pi_k})+(n-k)(D_{\pi'_k,\pi'_{k+1}}-D_{\pi_k,\pi_{k+1}})+(D_{\pi'_{j-1},\pi'_j}-D_{\pi_{j-1},\pi_j})$$

- When k=n-1 and j=n

$$\Delta TCT(k,j) = 2(D_{\pi'_{k-1},\pi'_k} - D_{\pi_{k-1}\pi_k}) + (D_{\pi'_k,\pi'_j} - D_{\pi_k,\pi_j})$$

Hence, the TCT of new sequence π' can be calculated by the following equation:

$$TCT_{\pi'} = TCT_{\pi} + \Delta TCT$$

Therefore, the calculation of TCT for the new sequence can be reduced from $O(n)$ to $O(1)$.

Appendix B. A numerical illustration

To illustrate the main steps of our CFI heuristic, we provide a 5-job 4-machine instance as shown in Table B.1, which is the same as in Bertolissi (2000).

Table B.1: Processing time of a 5-job 4-machine instance.

	M_1	M_2	M_3	M_4
J_1	12	24	12	13
J_2	20	3	19	11
J_3	19	20	3	15
J_4	14	23	16	14
J_5	19	15	17	22

- *Initial sequence algorithm*

Step 1: S=$\emptyset$ and U=$\{J_1, J_2, J_3, J_4, J_5\}$.

Step 2: Consider J_1 in the 1^{st} position of S, the processing times of J_2, J_3, J_4, and J_5 on each machine are the average processing times. APT_i=[18, 15.25, 13.75, 15.5]. Append this artificial job to J_1, and we can obtain the current idle time of 48, and future idle time of 13.25. The index function value for J_1, namely $f(1)$, is 205.25. We can consider J_2 in the 1^{st} position of S and obtain $f(2)$=211. Similarly, we can obtain $f(3)$=199.25, $f(4)$=222.25, and $f(5)$=230.25. Hence, we remove J_3 that has the minimum f value from U and put it into the 1^{st} position of S.

Step 3: For the 2^{nd} position in S, we can do the similar procedure as in Step 2, and obtain the index function values f for each job in U, which are f=[155, 53.33, 153, 100.33]. Hence we remove J_2 from U and put it into the 2^{nd} position of S. Similarly, we generate the initial sequence π_0 as $\{J_3, J_2, J_1, J_5, J_4\}$.

- *CFI heuristic*

The distance matrix $D_{n\times n}$ calculated by Eq.(2) is shown in Table B.2. From the ISA, we obtain initial sequence π_0 =$\{J_3, J_2, J_1, J_5, J_4\}$ and TCT_0=501. Set TCT_b=501, π_b =$\{J_3, J_2, J_1, J_5, J_4\}$, and r=1.

Table B.2: Distance matrix $D_{n\times n}$

	J_1	J_2	J_3	J_4	J_5
J_1	-	17	15	28	29
J_2	28	-	24	34	40
J_3	31	15	-	35	36
J_4	19	16	15	-	25
J_5	13	11	15	14	-

We can select first two jobs, J_3 and J_2, from the initial sequence, and obtain a TCT of 129 for a partial sequence $\{J_3, J_2\}$. Exchange the two jobs and obtain a TCT of 130 for a partial sequence $\{J_2, J_3\}$. Hence, we fix the relative positions of two jobs as a partial sequence of $\{J_3, J_2\}$.

Inserting J_1 from the initial sequence to each possible position of the partial sequence $\{J_3, J_2\}$, we can have the following partial sequences, $\{J_1, J_3, J_2\}$, $\{J_3, J_1, J_2\}$ and $\{J_3, J_2, J_1\}$ with partial TCTs of 228, 250, 229, respectively. Hence, we choose the partial sequence of $\{J_1, J_3, J_2\}$ as the current sequence with the minimum partial TCT of 228. The neighborhood exchanging method is applied, and the following partial sequences are examined, $\{J_3, J_1, J_2\}$, $\{J_2, J_3, J_1\}$ and $\{J_1, J_2, J_3\}$ with ΔTCTs of 22, 10 and 13, respectively. None of these values is lower than 0, therefore, the current sequence remains as $\{J_1, J_3, J_2\}$.

Insert J_5 from the initial sequence to each possible position of the current sequence, and the following partial sequences are examined: $\{J_5, J_1, J_3, J_2\}$, $\{J_1, J_5, J_3, J_2\}$, $\{J_1, J_3, J_5, J_2\}$ and $\{J_1, J_3, J_2, J_5\}$ with partial TCTs of 376, 376, 372, and 359, respectively. Hence, we choose $\{J_1, J_3, J_2, J_5\}$ as the current sequence with the minimum partial TCT of 359. The neighborhood ex-

changing method is applied, and the following partial sequences are examined: $\{J_3, J_1, J_2, J_5\}$, $\{J_2, J_3, J_1, J_5\}$, $\{J_5, J_3, J_2, J_1\}$, $\{J_1, J_2, J_3, J_5\}$, $\{J_1, J_5, J_2, J_3\}$ and $\{J_1, J_3, J_5, J_2\}$ with ΔTCTs of 36, 16, 36, 20, 18, and 13, respectively. None of these values is lower than 0, therefore, the current sequence remains as $\{J_1, J_3, J_2, J_5\}$.

Insert J_4 from the initial sequence to each possible position of the current sequence and the following candidates are tried: $\{J_4, J_1, J_3, J_2, J_5\}$, $\{J_1, J_4, J_3, J_2, J_5\}$, $\{J_1, J_3, J_4, J_2, J_5\}$, $\{J_1, J_3, J_2, J_4, J_5\}$ and $\{J_1, J_3, J_2, J_5, J_4\}$ with TCTs of 526, 532, 542, 503 and 504. Hence, we choose $\{J_1, J_3, J_2, J_4, J_5\}$ as the current sequence with minimum TCT of 503. The neighborhood exchanging method is applied and the following partial sequences are examined: $\{J_3, J_1, J_2, J_4, J_5\}$,$\{J_2, J_3, J_1, J_4, J_5\}$, $\{J_4, J_3, J_2, J_1, J_5\}$,$\{J_5, J_3, J_2, J_4, J_1\}$, $\{J_1, J_2, J_3, J_4, J_5\}$,$\{J_1, J_4, J_2, J_3, J_5\}$, $\{J_1, J_5, J_2, J_4, J_3\}$,$\{J_1, J_3, J_4, J_2, J_5\}$, $\{J_1, J_3, J_5, J_4, J_2\}$,and $\{J_1, J_3, J_2, J_5, J_4\}$ with ΔTCTs of 50, 32, 22, 54, 37, 46, 34, 39, 14 and 1. None of these values is lower than 0, therefore, the current sequence remains as $\{J_1, J_3, J_2, J_4, J_5\}$ with TCT 503.

After using insertion and neighborhood interchanging methods, we obtain π_1=$\{J_1, J_3, J_2, J_4, J_5\}$ and TCT_1=503. Since TCT_1 is larger than TCT_b, the π_b remains unaltered with TCT_b 501. For j=1 to 4, insert jth job into each possible position of π_1 in the forward direction and get the following sequences: $\{J_3, J_1, J_2, J_4, J_5\}$, $\{J_3, J_2, J_1, J_4, J_5\}$, $\{J_3, J_2, J_4, J_1, J_5\}$, $\{J_3, J_2, J_4, J_5, J_1\}$, $\{J_1, J_2, J_3, J_4, J_5\}$, $\{J_1, J_2, J_4, J_3, J_5\}$, $\{J_1, J_2, J_4, J_5, J_3\}$, $\{J_1, J_3, J_4, J_2, J_5\}$, $\{J_1, J_3, J_4, J_5, J_2\}$ and $\{J_1, J_3, J_2, J_5, J_4\}$ with TCTs of 553, 510, 514, 510, 540, 541, 540, 542, 531, and 504. None of these values is lower than TCT_b, the π_b remains unaltered with TCT_b 501 and is used for further process till r=6 iterations are completed. Hence, the final sequence is $\{J_3, J_2, J_1, J_5, J_4\}$ with TCT 501.

Appendix C. Solutions of our CFI heuristic in Taillard's benchmarks

Table C.1: Solutions of our CFI heuristic in Taillard's benchmarks

Instance	Solution	Instance	Solution	Instance	Solution	Instance	Solution
Ta001	15674	Ta031	77126	Ta061	320218	Ta091	1526495
Ta002	17558	Ta032	84418	Ta062	308057	Ta092	1530470
Ta003	15998	Ta033	79506	Ta063	301279	Ta093	1530953
Ta004	17970	Ta034	84054	Ta064	284759	Ta094	1511854
Ta005	15781	Ta035	86164	Ta065	296549	Ta095	1535620
Ta006	15501	Ta036	81774	Ta066	292155	Ta096	1500762
Ta007	15872	Ta037	79860	Ta067	304038	Ta097	1563778
Ta008	16068	Ta038	81915	Ta068	297351	Ta098	1547407
Ta009	16385	Ta039	77092	Ta069	315224	Ta099	1522429
Ta010	15463	Ta040	85249	Ta070	304551	Ta100	1538367
Ta011	25741	Ta041	116714	Ta071	420511	Ta101	2024174
Ta012	26804	Ta042	114148	Ta072	397544	Ta102	2070240
Ta013	22975	Ta043	108470	Ta073	414974	Ta103	2054907
Ta014	22528	Ta044	114734	Ta074	426970	Ta104	2091421
Ta015	23721	Ta045	118171	Ta075	404131	Ta105	2088352
Ta016	22785	Ta046	114012	Ta076	414439	Ta106	2087322
Ta017	21965	Ta047	119034	Ta077	404978	Ta107	2064240
Ta018	24205	Ta048	117341	Ta078	408479	Ta108	2052833
Ta019	23550	Ta049	112017	Ta079	416797	Ta109	2070279
Ta020	24954	Ta050	117618	Ta080	423957	Ta110	2046873
Ta021	39165	Ta051	175780	Ta081	570200	Ta111	11646936
Ta022	38009	Ta052	163344	Ta082	574956	Ta112	11945671
Ta023	38566	Ta053	161056	Ta083	570868	Ta113	11629905
Ta024	38812	Ta054	164654	Ta084	570673	Ta114	11804810
Ta025	39071	Ta055	170161	Ta085	562705	Ta115	11857565
Ta026	38620	Ta056	163056	Ta086	575052	Ta116	11753926
Ta027	39976	Ta057	168547	Ta087	572473	Ta117	11758172
Ta028	37240	Ta058	170710	Ta088	586881	Ta118	11798447
Ta029	39629	Ta059	167103	Ta089	574629	Ta119	11702236
Ta030	38422	Ta060	170845	Ta090	594755	Ta120	11806623

Acknowledgments

We appreciate the support from Agency for Healthcare Research and Quality, UK HealthCare, Department of Mechanical Engineering at University of Kentucky and Haskayne School of Business at University of Calgary.

References

Akhshabi, M., Tavakkoli-Moghaddam, R., and Rahnamay-Roodposhti, F. (2014). A hybrid particle swarm optimization algorithm for a no-wait flow shop scheduling problem with the total flow time. *The International Journal of Advanced Manufacturing Technology*, 70(5-8):1181–1188.

Aldowaisan, T. and Allahverdi, A. (2003). New heuristics for no-wait flowshops to minimize makespan. *Computers & Operations Research*, 30(8):1219–1231.

Aldowaisan, T. and Allahverdi, A. (2004). New heuristics for m-machine no-wait flowshop to minimize total completion time. *Omega*, 32(5):345–352.

Allahverdi, A. (2016). A survey of scheduling problems with no-wait in process. *European Journal of Operational Research*, 255:665–686.

Bertolissi, E. (2000). Heuristic algorithm for scheduling in the no-wait flow-shop. *Journal of Materials Processing Technology*, 107(1):459–465.

Chen, C.-L., Neppalli, R. V., and Aljaber, N. (1996). Genetic algorithms applied to the continuous flow shop problem. *Computers & Industrial Engineering*, 30(4):919–929.

Ding, J., Song, S., Zhang, R., Gupta, J. N., and Wu, C. (2015). Accelerated methods for total tardiness minimisation in no-wait flowshops. *International Journal of Production Research*, 53(4):1002–1018.

Fink, A. and Voß, S. (2003). Solving the continuous flow-shop scheduling problem by metaheuristics. *European Journal of Operational Research*, 151(2):400–414.

Framinan, J. M. and Nagano, M. S. (2008). Evaluating the performance for makespan minimisation in no-wait flowshop sequencing. *Journal of Materials Processing Technology*, 197(1):1–9.

Framinan, J. M., Nagano, M. S., and Moccellin, J. V. (2010). An efficient heuristic for total flowtime minimisation in no-wait flowshops. *The International Journal of Advanced Manufacturing Technology*, 46(9-12):1049–1057.

Gao, K., Pan, Q., Suganthan, P., and Li, J. (2013). Effective heuristics for the no-wait flow shop scheduling problem with total flow time minimization. *The International Journal of Advanced Manufacturing Technology*, 66(9-12):1563–1572.

Gao, K.-z., Pan, Q.-k., and Li, J.-q. (2011). Discrete harmony search algorithm for the no-wait flow shop scheduling problem with total flow time criterion. *The International Journal of Advanced Manufacturing Technology*, 56(5-8):683–692.

Grabowski, J. and Pempera, J. (2005). Some local search algorithms for no-wait flow-shop problem with makespan criterion. *Computers & Operations Research*, 32(8):2197–2212.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326.

Gupta, D. (2007). Surgical suites' operations management. *Production and Operations Management*, 16(6):689–700.

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

Kalczynski, P. J. and Kamburowski, J. (2007). On the neh heuristic for minimizing the makespan in permutation flow shops. *Omega*, 35(1):53–60.

Kalir, A. A. and Sarin, S. C. (2001). A near-optimal heuristic for the sequencing problem in multiple-batch flow-shops with small equal sublots. *Omega*, 29(6):577–584.

Laha, D. and Chakraborty, U. K. (2009). A constructive heuristic for minimizing makespan in no-wait flow shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 41(1-2):97–109.

Laha, D. and Gupta, J. N. (2016). A hungarian penalty-based construction algorithm to minimize makespan and total flow time in no-wait flow shops. *Computers & Industrial Engineering*, 98:373–383.

Laha, D., Gupta, J. N., and Sapkal, S. U. (2014). A penalty-shift-insertion-based algorithm to minimize total flow time in no-wait flow shops. *Journal of the Operational Research Society*, 65(10):1611–1624.

Laha, D. and Sapkal, S. U. (2014). An improved heuristic to minimize total flow time for scheduling in the m-machine no-wait flow shop. *Computers & Industrial Engineering*, 67:36–43.

Li, W. and Freiheit, T. I. (2016). An effective heuristic for adaptive control of job sequences subject to variation in processing times. *International Journal of Production Research*, 54(12):3491–3507.

Li, W., Luo, X., Xue, D., and Tu, Y. (2011a). A heuristic for adaptive production scheduling and control in flow shop production. *International Journal of Production Research*, 49(11):3151–3170.

Li, W., Nault, B. R., Xue, D., and Tu, Y. (2011b). An efficient heuristic for adaptive production scheduling and control in one-of-a-kind production. *Computers & Operations Research*, 38(1):267–276.

Li, X., Wang, Q., and Wu, C. (2008). Heuristic for no-wait flow shops with makespan minimization. *International Journal of Production Research*, 46(9):2519–2530.

Lin, S.-W. and Ying, K.-C. (2016). Optimization of makespan for no-wait flow-shop scheduling problems using efficient matheuristics. *Omega*, 64:115–125.

Liu, J. and Reeves, C. R. (2001). Constructive and composite heuristic solutions to the p//ci scheduling problem. *European Journal of Operational Research*, 132(2):439–452.

Montgomery, D. C. (2007). *Introduction to Statistical Quality Control*. John Wiley & Sons.

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

Pan, Q.-K., Tasgetiren, M. F., and Liang, Y.-C. (2008). A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research*, 35(9):2807–2839.

Qi, X., Wang, H., Zhu, H., Zhang, J., Chen, F., and Yang, J. (2016). Fast local neighborhood search algorithm for the no-wait flow shop scheduling with total flow time minimization. *International Journal of Production Research*, 54(16):4957–4972.

Rajendran, C. and Alicke, K. (2007). Dispatching in flowshops with bottleneck machines. *Computers & Industrial Engineering*, 52(1):89–106.

Rajendran, C. and Chaudhuri, D. (1990). Heuristic algorithms for continuous flow-shop problem. *Naval Research Logistics (NRL)*, 37(5):695–705.

Rajendran, C. and Ziegler, H. (1997). An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. *European Journal of Operational Research*, 103(1):129–138.

Röck, H. (1984). The three-machine no-wait flow shop is np-complete. *Journal of the ACM (JACM)*, 31(2):336–345.

Shyu, S. J., Lin, B. M., and Yin, P.-Y. (2004). Application of ant colony optimization for no-wait flowshop scheduling problem to minimize the total completion time. *Computers & Industrial Engineering*, 47(2):181–193.

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

Tavakkoli-Moghaddam, R., Rahimi-Vahed, A., and Mirzaei, A. H. (2007). A hybrid multi-objective immune algorithm for a flow shop scheduling problem with bi-objectives: weighted mean completion time and weighted mean tardiness. *Information Sciences*, 177(22):5072–5090.

Ye, H., Li, W., and Miao, E. (2016). An effective heuristic for no-wait flow shop production to minimize makespan. *Journal of Manufacturing Systems*, 40:2–7.

Ying, K.-C., Lin, S.-W., and Wu, W.-J. (2016). Self-adaptive ruin-and-recreate algorithm for minimizing total flow time in no-wait flowshops. *Computers & Industrial Engineering*, 101:167–176.

Zhu, X. and Li, X. (2015). Iterative search method for total flowtime minimization no-wait flowshop problem. *International Journal of Machine Learning and Cybernetics*, 6(5):747–761.