J Sched (2017) 20:293–302
DOI 10.1007/s10951-016-0493-x

Two-machine flowshop scheduling with three-operation jobs subject to a fixed job sequence

Bertrand M. T. Lin[1] · **F. J. Hwang**[2] · **Jatinder N. D. Gupta**[3]

Published online: 11 August 2016

Abstract This paper studies the problem of scheduling three-operation jobs in a two-machine flowshop subject to a predetermined job processing sequence. Each job has two preassigned operations, which are to be performed on their respective dedicated machines, and a flexible operation, which may be processed on either of the two machines subject to the processing order as specified. Five standard objective functions, including the makespan, the maximum lateness, the total weighted completion time, the total weighted tardiness, and the weighted number of tardy jobs are considered. We show that the studied problem for either of the five considered objective functions is ordinary NP-hard, even if the processing times of the preassigned operations are zero for all jobs. A pseudo-polynomial time dynamic programming framework, coupled with brief numerical experiments, is then developed for all the addressed performance metrics with different run times.

Keywords Flowshop · Three-operation job · Fixed job sequence · NP-hardness · Dynamic program

1 Introduction

This paper investigates a two-stage flowshop problem for scheduling a given sequence of jobs, each of which consists of three operations. The setting without the presumption of a fixed job sequence is formulated and studied by Gupta et al. (2004) for the performance metric of makespan. The three-operation flowshop scheduling problem is described as follows. A set of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ is to be processed in a two-machine flowshop, consisting of machine A at stage 1 and machine B at stage 2. Each job $J_j \in \mathcal{J}$ has three operations, O_j^A, O_j^B, and O_j^C with non-negative processing times a_j, b_j, and c_j, respectively, and is associated with due date d_j and weight w_j. The three operations of each job must follow the processing order (O_j^A, O_j^C, O_j^B). Operations O_j^A and O_j^B are preassigned to be performed on their respective dedicated machines A and B. Operation O_j^C is flexible and may be processed on either of the two machines subject to the processing order as specified. The problem with makespan minimization is at least binary NP-hard for it corresponds to the parallel machine scheduling problem $P2||C_{\max}$ when $a_j = b_j = 0$ for all jobs $J_j \in \mathcal{J}$. This problem becomes the traditional two-machine flowshop scheduling problem, which can be solved by Johnson's rule (Johnson 1954), provided that an optimal assignment of the flexible operations to the two machines is given (Gupta et al. 2004). Gupta et al. (2004) presented a $\frac{3}{2}$-approximation algorithm and developed a polynomial time approximation scheme.

The above problem with identical jobs was discussed by Crama and Gultekin (2010), Gultekin (2012), and Uruk et al. (2013). Crama and Gultekin (2010) proposed some optimality properties and polynomial-time algorithms for various cases where the number of jobs is either finite or infinite, and where the buffer capacity in between machines is either zero, limited, or unlimited. Gultekin (2012) later considered the scenario of non-identical machines, viz., the processing times of the flexible operation are different on the two machines, and developed a constant-time solution procedure for the

✉ F. J. Hwang
feng-jang.hwang@uts.edu.au

[1] Department of Information Management and Finance, Institute of Information Management, National Chiao Tung University, Hsinchu 300, Taiwan

[2] School of Mathematical and Physical Sciences, University of Technology Sydney, Ultimo 2007, Australia

[3] College of Business Administration, The University of Alabama in Huntsville, Huntsville, AL 35899, USA

cases with no buffer and unlimited buffer capacity in between machines. Uruk et al. (2013) investigated the scenario where the processing times of the preassigned and flexible operations are controllable and can be any value within the given interval, and the manufacturing cost of an operation is a nonlinear function of its processing time. Mixed integer nonlinear programs were derived for the bi-criteria objective of makespan and total manufacturing cost, and a heuristic algorithm was designed for large-scale instances. The practical applications of the three-operation flowshop scheduling problem could be found in flexible manufacturing systems with machine linkage, inventory control transit centers with bar-coding operations, and scheduling in farming (Gupta et al. 2004). Crama and Gultekin (2010) also described the industrial settings in the assembly of printed circuit boards and in automated computer numerical control machines.

Since the three-operation flowshop scheduling problem is intractable and involves the decisions of partitioning and sequencing, it could be worth investigating the restricted problem where one of the decisions is predetermined, especially from the perspective on solution approach development which will be addressed later. In this paper, we discuss the problem subject to the assumption of a fixed job sequence, i.e., the processing sequence of all jobs is known and given a priori. Five standard objective functions, namely the makespan ($C_{\max}$), the maximum lateness ($L_{\max}$), the total weighted completion time ($\sum w_j C_j$), the total weighted tardiness ($\sum w_j T_j$), and weighted number of tardy jobs ($\sum w_j U_j$), are considered. Without loss of generality, the predetermined job sequence is $(J_1, J_2, \ldots, J_n)$. The condition of a fixed job sequence requires that on machine A as well as machine B, job J_i should precede job J_j if $i < j$.

In the design of branch-and-bound algorithms, local search-based methods and meta-heuristics for handling intractable problems, the sequence- or permutation-based representations for candidate solutions are commonly adopted. It is demanded to have efficient algorithms for determining the objective values of given job/operation sequences. In the classification of complexity status, special properties, e.g. agreeable conditions, could also suggest the optimality of specific job orderings. One envisaged industrial application of the fixed-job-sequence setting in the three-operation flowshop scheduling problem could be the scheduling of bar-coding operations in inventory or stock control systems, where the First-Come-First-Served principle is applied. For each item $J_j \in \mathcal{J}$, operations O_j^A, O_j^C, and O_j^B are unpacking, bar-coding process, and repacking, respectively. It is commonly regarded fair by customers and easy to be implemented by processors that the unpacking and repacking sequences of items are identical and predetermined by the item/order receiving times. Other theoretical and practical justifications of the fixed-job-sequence assumption from the technological or managerial considerations can be found in, e.g. Shafransky and Strusevich (1998), Lin and Hwang (2011), Hwang et al. (2012, 2014) and Lin et al. (2016).

The remainder of this paper is organized as follows. In Sect. 2, we discuss the NP-hardness of the studied problem for the considered performance metrics. Section 3 contains the development of pseudo-polynomial time dynamic programming algorithms. We conclude the paper and suggest some research issues in Sect. 4.

2 NP-hardness

The requirement of operation processing order and the condition of fixed job sequence jointly imply the standard format of a feasible schedule, where for each job J_j the flexible operation O_j^C is immediately preceded by the corresponding preassigned operation O_j^A or immediately followed by the corresponding preassigned operation O_j^B. The studied fixed-job-sequence problem can thus be regarded as the problem of finding an optimal partition of the flexible operations. The determination of its complexity status, however, is not straightforward as will be shown later. Before proving the NP-hardness of the problem under study, an optimality property is described as follows.

Lemma 1 *For any regular objective function, there exists an optimal schedule in which the flexible operation of the first job, i.e. O_1^C, goes to machine B and that of the last job, i.e. O_n^C, goes to machine A.*

Proof Let σ be an optimal schedule that does not satisfy the specified property. We move operation O_1^C from machine A to machine B to derive another schedule σ'. Depletion of the processing on machine A lets all the operations behind O_1^A on machine A start earlier by c_1 units of time. Merging O_1^C into O_1^B will not defer the completion of any operation on machine B. Therefore, the completion times $C_j(\sigma') \leq C_j(\sigma)$ for all jobs $J_j \in \mathcal{J}$. Next, we consider operation O_n^C. Depleting O_n^C from machine B will decrease machine B's completion time by c_n. Appending it to machine A will delay the start time of operation O_n^B by at most c_n. Therefore, the derived schedule has a regular objective function value not greater than that of σ' and thus σ. □

Theorem 1 *The studied fixed-job-sequence problem for makespan minimization is NP-hard in the ordinary sense, even if $a_j = b_j = 0$ for all jobs $J_j \in \mathcal{J}$.*

Proof Let integer bound E and integer sizes $e_1, e_2, \ldots, e_m$ with $\sum_{j=1}^{m} e_j = 2E$ be an instance of Partition. We create a scheduling instance with $m + 2$ jobs as follows:

Job $J_0 : c_0 = E$;

Job $J_j : c_j = e_j$, $j \in \{1, \ldots, m\}$;

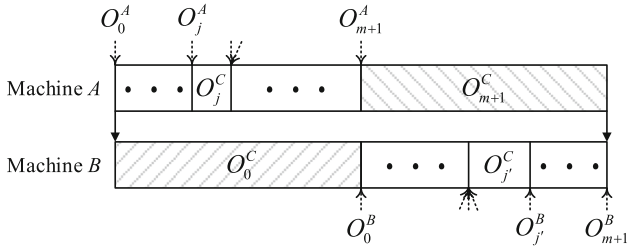

Fig. 1 Configuration of a feasible schedule with a makespan of $2E$

Job $J_{m+1}: c_{m+1} = E$.

Note that $a_j = b_j = 0$ for all $j \in \{0, 1, \ldots, m+1\}$ and the jobs abide by the processing sequence $(J_0, J_1, \ldots, J_{m+1})$. We claim that the answer to Partition is affirmative if and only if there is a feasible schedule with a makespan of $2E$. Recall that it is necessary and sufficient to consider the schedules where each flexible operation O_j^C is immediately preceded by the corresponding operation O_j^A or immediately followed by the corresponding operation O_j^B.

Let X_1 and X_2 be a partition of the m indices in Partition. A feasible schedule is constructed as follows. Machines A and B are initialized with two processing sequences of operations $(O_0^A, O_1^A, \ldots, O_{m+1}^A)$ and $(O_0^B, O_1^B, \ldots, O_{m+1}^B)$, respectively. Then operations O_0^C and operation O_{m+1}^C are assigned to machine B and machine A, respectively. Let each operation O_j^C, $j \in X_1$ processed on machine A immediately preceded by O_j^A, and each operation $O_{j'}^C$, $j' \in X_2$ processed on machine B immediately followed by $O_{j'}^B$. The makespan is exactly $2E$ as depicted in Fig. 1.

Assume now that there is a feasible schedule the makespan of which is exactly $2E$. Since the sum of the processing loads of all jobs is $4E$, no idle time is allowed on either machine. By Lemma 1, we assume that O_0^C and O_{m+1}^C are processed on machine B and machine A, respectively. Let the set $\mathcal{J}_A$ contain the jobs whose flexible operations are assigned to machine A. If $\sum_{J_j \in \mathcal{J}_A} c_j > E$, then the completion time of O_{m+1}^C on machine A is larger than $2E$, which contradicts the assumption. On the other hand, if $\sum_{J_j \in \mathcal{J}_A} c_j < E$, then $\sum_{J_j \in \{J_1, \ldots, J_m\} \setminus \mathcal{J}_A} c_j > E$ and the completion time of machine B is greater than $2E$. Therefore, we must have $\sum_{J_j \in \mathcal{J}_A} c_j = E$, and a partition is obtained. □

The relationships between the standard objective functions lead to the following corollary.

Corollary 1 *The studied fixed-job-sequence problem for* $L_{\max}, \sum w_j C_j, \sum T_j$ *or* $\sum U_j$ *is NP-hard in the ordinary sense, even if* $a_j = b_j = 0$ *for all jobs* $J_j \in \mathcal{J}$.

Proof The minimization of maximum lateness is a natural generalization of makespan minimization since $C_{\max}$ is a special case of $L_{\max}$ by setting $d_j = 0$ for all jobs $J_j \in \mathcal{J}$. In the studied fixed-job-sequence problem, makespan minimization also corresponds to a special case of minimizing $\sum w_j C_j$ with $w_j = 0$, $j \in \{1, 2, \ldots, n-1\}$ and $w_n = 1$. As for the objective function $\sum T_j$ or $\sum U_j$, the proof technique utilized in Theorem 1 can be applied to show the NP-hardness by deploying arguments with $d_j = 2E$ for all jobs $J_j \in \mathcal{J}$ and a feasible schedule retaining $\sum T_j = 0$ or $\sum U_j = 0$. □

3 Pseudo-polynomial time dynamic programs

In this section, pseudo-polynomial time dynamic programs are proposed for the five addressed objective functions $C_{\max}$, $L_{\max}$, $\sum w_j C_j$, $\sum w_j T_j$, and $\sum w_j U_j$. The notion of the developed dynamic programming stems from the observation that a feasible schedule can be decomposed into several subschedules separated by the machine-B idle times. Then a subschedule in some state can be defined by the schedule shape characteristics. In the designed two-phase dynamic programming framework, optimal subschedules of all states are constructed in the first phase. Then an optimal schedule can be assembled by concatenating appropriate optimal subschedules in the second phase.

3.1 Makespan

For a particular subschedule, the difference between the completion times of the two machines is called the lag of this subschedule. Define a subschedule named *schedule block* in state (k, i, j, ℓ) as a subschedule for jobs $\{J_i, J_{i+1}, \ldots, J_j\}$ satisfying the following conditions: (1) no idle time exists between any two consecutive operations on the machines, i.e. block property; (2) the first flexible operation O_i^C goes to machine $k \in \{A, B\}$; (3) the lag is exactly ℓ. The shape characteristics of the schedule blocks in some state can be delineated in terms of the above three conditions. A schedule block in state (k, i, j, ℓ), where $i < j$, can be built up by suffixing J_j to an appropriate schedule block in state $(k, i, j-1, \ell')$. Two possible construction scenarios are depicted in Fig. 2, where $k = A$. Let function $f(k, i, j, \ell)$ denote the minimum machine-A processing span among the blocks in state (k, i, j, ℓ). The schedule block retaining $f(k, i, j, \ell)$ is called the optimal block in state (k, i, j, ℓ). To facilitate further presentation, define the Kronecker delta

$$\delta_{kB} = \begin{cases} 0, & \text{if } k = A; \\ 1, & \text{if } k = B, \end{cases}$$

and the aggregate quantities $a_{[i:j]} = \sum_{h=i}^{j} a_h$, $c_{[i:j]} = \sum_{h=i}^{j} c_h$, $b_{[i:j]} = \sum_{h=i}^{j} b_h$, and $w_{[i:j]} = \sum_{h=i}^{j} w_h$, $1 \le i \le j \le n$. The block property of a schedule block in state (k, i, j, ℓ), where $i < j$, implies

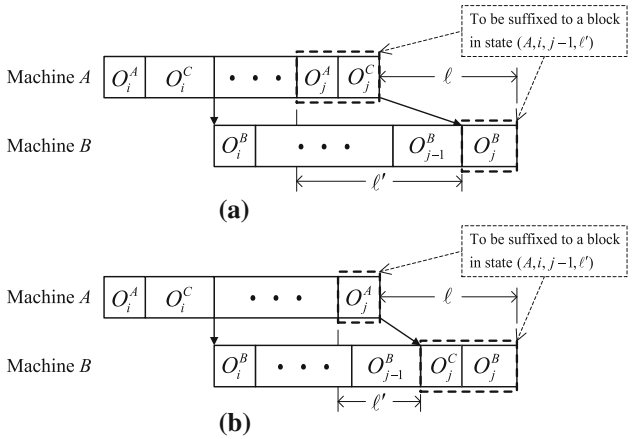

Fig. 2 Construction of a block in state (A, i, j, ℓ) from a block in state $(A, i, j-1, \ell')$

$$c_i\delta_{kB} + b_{[i:j]} - a_{[i+1:j]} - c_{[i+1:j]} \le \ell \le c_i\delta_{kB} + b_{[i:j]} + c_{[i+1:j]} - a_{[i+1:j]},$$

and the block exists only if $\ell \ge b_j$. Denote

$$\underline{\ell} = \max\{b_j, c_i\delta_{kB} + b_{[i:j]} - a_{[i+1:j]} - c_{[i+1:j]}\},$$

and

$$\overline{\ell} = c_i\delta_{kB} + b_{[i:j]} + c_{[i+1:j]} - a_{[i+1:j]}.$$

It thus suffices to consider ℓ over the interval $\left[\underline{\ell}, \overline{\ell}\right]$, which is valid if the condition $\overline{\ell} \ge b_j$ can be ensured. Actually, given a 3-tuple k, i, j, the value $\overline{\ell}$, which denotes the upper limit to ℓ, is calculated by assuming that the flexible operation(s) including O_j^C go to machine B, and thus a schedule block can be constructed only if $\overline{\ell} \ge c_j + b_j$. The dynamic programs for block construction can be described in the following.

BLOCK CONSTRUCTION ($C_{\max}$)
Initial conditions:

$$f(k,i,j,\ell) = \begin{cases} a_i + c_i, & \text{if } k = A, i = j, \text{ and } \ell = b_i; \\ a_i, & \text{if } k = B, i = j, \text{ and } \ell = c_i + b_i; \\ \infty, & \text{otherwise.} \end{cases}$$

Recursions:
for each 4-tuple k, i, j, ℓ satisfying $k \in \{A, B\}$, $1 \le i < j \le n$, and $\underline{\ell} \le \ell \le \overline{\ell}$,
/* The recursion runs only if $\overline{\ell} \ge c_j + b_j$.*/
Case 1 (Operation O_j^C being assigned to machine A):

$$y_1 = f(k,i,j-1,\ell + a_j + c_j - b_j) + (a_j + c_j), \quad (1)$$

Case 2 (Operation O_j^C being assigned to machine B):

$$y_2 = \begin{cases} f(k,i,j-1,\ell + a_j - c_j - b_j) + a_j, & \text{if } \ell \ge c_j + b_j; \\ \infty, & \text{otherwise,} \end{cases}$$

$f(k,i,j,\ell) = \min\{y_1, y_2\}$.

The formulation is justified as follows. Given a feasible combination of k, i, j, ℓ, the derivation of $f(k, i, j, \ell)$ can be done by considering two scenarios regarding the assignment of the last flexible operation O_j^C. Case 1 is for assigning operation O_j^C to machine A, as shown in Fig. 2a. By suffixing the two operations O_j^A and O_j^C on machine A, and operation O_j^B on machine B to the optimal block in state $(k, i, j-1, \ell')$, we have $(a_j + c_j) + \ell = \ell' + b_j$, subject to the condition of block property $\ell' \ge a_j + c_j$. Thus, it can be shown that $\ell' = \ell + a_j + c_j - b_j$, as given in Eq. (1), subject to the condition $\ell + a_j + c_j - b_j \ge a_j + c_j$, i.e. $\ell \ge b_j$, which is satisfied anyway in the recursions owing to the definition of $\underline{\ell}$, the lower limit on ℓ. In Case 2 where operation O_j^C is processed on machine B as shown in Fig. 2b, we have $a_j + \ell = \ell' + (c_j + b_j)$, subject to the condition $\ell' = \ell + a_j - c_j - b_j \ge a_j$, i.e. $\ell \ge c_j + b_j$. Note that the two cases are not disjoint about the associated conditions $\ell \ge b_j$ and $\ell \ge b_j + c_j$ and the first condition subsumes the second one. If $\ell \ge c_j + b_j$ is satisfied, then operation O_j^C is allowed to be dispatched to either of the machines.

After the block construction, a complete schedule can be generated with a concatenation of appropriate optimal blocks in backward recursion. Notice that in a schedule any two adjacent optimal blocks are separated by an inserted idle time on machine B to ensure that each optimal block is maximal for inclusion. Define by a partial schedule in state (k, i) a schedule of jobs $\{J_i, J_{i+1}, \ldots, J_n\}$, the prefix of which is an optimal block having the leading job J_i and operation O_i^C processed on machine k. Denote by $g(k, i)$ the minimum makespan among all the partial schedules in state (k, i). The dynamic program for schedule concatenation, where we need a dummy job J_{n+1} with $a_{n+1} = c_{n+1} = \infty$, is depicted in SCHEDULE CONCATENATION ($C_{\max}$).

The algorithm procedure is validated as follows. There are two cases for assembling a partial schedule in (k, i) by prefixing an optimal block in (k, i, j, ℓ). In Case 1, the optimal block in (k, i, j, ℓ) is attached to the front end of the optimal partial schedule in $(A, j+1)$, as illustrated in Fig 3a, where $k = B$. To satisfy the schedule concatenation property about idle times, the inequality $\ell < a_{j+1} + c_{j+1}$ is required. As for Case 2, we have the optimal block in (k, i, j, ℓ) prefixed to the optimal partial schedule in $(B, j+1)$ as shown in Fig. 3b (where $k = B$), and thus $\ell < a_{j+1}$. Notice again that the first associated condition $\ell < a_{j+1} + c_{j+1}$ subsumes the second one $\ell < a_{j+1}$. The term $\ell\lfloor\frac{j}{n}\rfloor$ is added for the scenario where $j = n$.

The run time of the developed dynamic program can be analysed as follows. In the block construction, there are at most $O(n^2 \sum_{h=1}^{n} c_h)$ states, each of which needs $O(1)$ time for calculation, since the size of the interval $\left[\underline{\ell}, \overline{\ell}\right]$ is in the order of $O(\sum_{h=1}^{n} c_h)$. Thus, the run time for block construction is $O(n^2 \sum_{h=1}^{n} c_h)$. In the schedule concatenation, there

SCHEDULE CONCATENATION ($C_{\max}$)
Initial conditions:
for each $k \in \{A, B\}$,

$$g(k,i) = \begin{cases} 0, & \text{if } i = n+1,; \\ \infty, & \text{otherwise.} \end{cases}$$

Recursions:
for each 2-tuple k, i satisfying $k \in \{A, B\}$, and $1 \le i \le n$,

$$g(k,i) = \min_{\substack{i \le j \le n; \\ \underline{\ell} \le \ell \le \overline{\ell}}} \{z_1, z_2\}, \tag{2}$$

where z_1 and z_2 are calculated as in the following:
Case 1 (The optimal block in (k, i, j, ℓ) being attached to the optimal partial schedule in $(A, j+1)$):

$$z_1 = \begin{cases} f(k,i,j,\ell) + \ell\lfloor \frac{j}{n} \rfloor + g(A, j+1), & \text{if } \ell < a_{j+1} + c_{j+1}; \\ \infty, & \text{otherwise,} \end{cases}$$

Case 2 (The optimal block in (k, i, j, ℓ) being attached to the optimal partial schedule in $(B, j+1)$):

$$z_2 = \begin{cases} f(k,i,j,\ell) + \ell\lfloor \frac{j}{n} \rfloor + g(B, j+1), & \text{if } \ell < a_{j+1}; \\ \infty, & \text{otherwise.} \end{cases}$$

Goal: $\min_{k \in \{A,B\}} \{g(k,1)\}$.

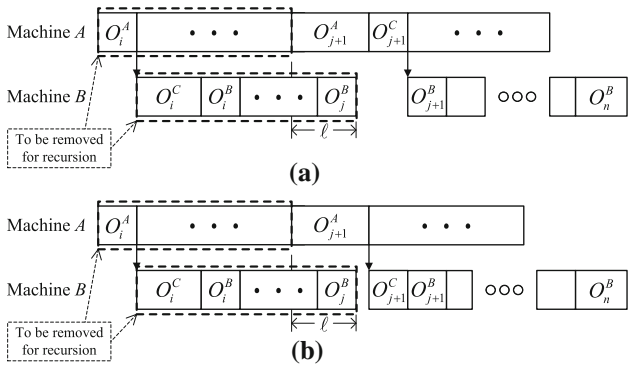

Fig. 3 Schedule concatenation ($C_{\max}$) for assembling a partial schedule in state (B, i) by prefixing an optimal block in (B, i, j, ℓ)

are $O(n)$ states, each of which takes at most $O(n \sum_{h=1}^{n} c_h)$ time due to the loops over all possible subscripts of the min operator in Eq. (2). So the run time for schedule concatenation is $O(n^2 \sum_{h=1}^{n} c_h)$. The total run time of the presented dynamic program is therefore $O(n^2 \sum_{h=1}^{n} c_h)$.

Theorem 2 *The studied fixed-job-sequence problem for makespan minimization is pseudo-polynomially solvable in $O(n^2 \sum_{h=1}^{n} c_h)$ time.*

In the following subsections, we extend the design of the above dynamic programming algorithm to the performance metrics of $L_{\max}$, $\sum w_j C_j$, $\sum w_j T_j$, and $\sum_j w_j U_j$. The development starts with the maximum lateness and the total weighted completion time. The solution procedure will then be adapted for the total weighted tardiness and weighted number of tardy jobs.

3.2 Maximum lateness and total weighted completion time

In the previous subsection, the block construction is carried out by minimizing the processing span of machine A subject to a specified lag. For the objective function of $L_{\max}$, we however cannot obtain a shortest machine-A processing span of a block while minimizing the maximum lateness within this block. The same difficulty also arises in the pursuit of the minimum total weighted completion time. Therefore, we introduce another parameter to freeze machine-A processing spans of the constructed blocks.

Define a schedule block in state (k, i, j, S, ℓ) as a subschedule for jobs $\{J_i, J_{i+1}, \ldots, J_j\}$ satisfying the following four conditions: (1) no idle time is inserted between any two consecutive operations on the machines; (2) the first flexible operation O_i^C is assigned to machine $k \in \{A, B\}$; (3) the machine-A processing span is exactly S; (4) the lag is exactly ℓ. Let function $f(k, i, j, S, \ell)$ denote the optimal maximum lateness of the jobs $\{J_i, J_{i+1}, \ldots, J_j\}$ among the blocks in state (k, i, j, S, ℓ). Then, the dynamic program for block construction is formulated as follows.

BLOCK CONSTRUCTION ($L_{\max}$)
Initial conditions:

$$f(k,i,j,S,\ell) = \begin{cases} a_i + c_i + b_i - d_i, & \text{if } k = A, i = j, S = a_i + c_i, \text{ and } \ell = b_i; \\ a_i + c_i + b_i - d_i, & \text{if } k = B, i = j, S = a_i, \text{ and } \ell = c_i + b_i; \\ \infty, & \text{otherwise.} \end{cases}$$

Recursions:
for each 5-tuple k, i, j, S, ℓ satisfying $k \in \{A, B\}$, $1 \le i < j \le n$, $a_{[i:j]} \le S \le a_{[i:j]} + c_{[i:j]}$, and $\underline{\ell} \le \ell \le \overline{\ell}$,
/* The recursion runs only if $\overline{\ell} \ge c_j + b_j$.*/
Case 1 (Operation O_j^C being assigned to machine A, as shown in Fig. 4):
$y_1 = \max\{f(k, i, j-1, S - a_j - c_j, \ell + a_j + c_j - b_j), S + \ell - d_j\})$,
Case 2 (Operation O_j^C being assigned to machine B):

$$y_2 = \begin{cases} \max\{f(k,i,j-1,S-a_j,\ell+a_j-c_j-b_j), S+\ell-d_j\}, & \text{if } \ell \ge c_j + b_j; \\ \infty, & \text{otherwise,} \end{cases}$$

$f(k, i, j, S, \ell) = \min\{y_1, y_2\}$.

After the block construction procedure, we can then generate complete schedules for a sequence of jobs with idle times inserted between blocks. Let a partial schedule in state (k, i) be a schedule of jobs $\{J_i, J_{i+1}, \ldots, J_n\}$, the prefix of which is an optimal block having the leading job J_i and operation O_i^C

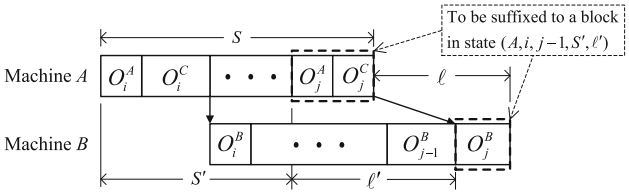

Fig. 4 Construction of a block in (A, i, j, S, ℓ) from a block in $(A, i, j-1, S', \ell')$

processed on machine k. Denote by $g(k, i)$ the optimal maximum lateness among all the partial schedules in state (k, i). The dynamic program for schedule concatenation, where we introduce a dummy job J_{n+1} with $a_{n+1} = c_{n+1} = \infty$, is shown below.

SCHEDULE CONCATENATION ($L_{\max}$)
Initial conditions:
for $k \in \{A, B\}$,

$$g(k, i) = \begin{cases} -\infty, & \text{if } i = n+1, ; \\ \infty, & \text{otherwise.} \end{cases}$$

Recursions:
for each 2-tuple k, i satisfying $k \in \{A, B\}$, and $1 \le i \le n$,

$$g(k, i) = \min_{\substack{i \le j \le n; \\ a_{[i:j]} \le S \le a_{[i:j]} + c_{[i:j]}; \\ \underline{\ell} \le \ell \le \overline{\ell}}} \{z_1, z_2\},$$

where z_1 and z_2 are calculated as in the following:
Case 1 (The optimal block in (k, i, j, S, ℓ) being attached to the optimal partial schedule in $(A, j+1)$, as illustrated in Fig. 5):

$$z_1 = \begin{cases} \max\{f(k, i, j, S, \ell), S + g(A, j+1)\}, & \text{if } \ell < a_{j+1} + c_{j+1}; \\ \infty, & \text{otherwise,} \end{cases}$$

Case 2 (The optimal block in (k, i, j, S, ℓ) being attached to the optimal partial schedule in $(B, j+1)$):

$$z_2 = \begin{cases} \max\{f(k, i, j, S, \ell), S + g(B, j+1)\}, & \text{if } \ell < a_{j+1}; \\ \infty, & \text{otherwise.} \end{cases}$$

Goal: $\min_{k \in \{A, B\}} \{g(k, 1)\}$.

Regarding the run time, there are at most $O(n^2(\sum_{h=1}^{n} c_h)^2)$ states, each of which needs $O(1)$ time for calculation of $f(k, i, j, S, \ell)$. Thus, the run time for block construction is thus $O(n^2(\sum_{h=1}^{n} c_h)^2)$. In the schedule concatenation, there are $O(n)$ states, each of which takes at most $O(n(\sum_{h=1}^{n} c_h)^2)$ time due to the loops over all possible subscripts of the min operator. So the run time for schedule concatenation is $O(n^2(\sum_{h=1}^{n} c_h)^2)$. The total run time of the presented dynamic program is therefore $O(n^2(\sum_{h=1}^{n} c_h)^2)$.

Theorem 3 *The studied fixed-job-sequence problem for minimizing the maximum lateness is pseudo-polynomially solvable in $O(n^2(\sum_{h=1}^{n} c_h)^2)$ time.*

Next, we consider the objective function of total weighted completion time. The algorithm framework for maximum lateness minimization can be readily adapted for the minimization of total weighted completion time with slight modifications. Let $f(k, i, j, S, \ell)$ be the minimum total weighted completion time of the state (k, i, j, S, ℓ). Definitions of initial conditions, z_1, and z_2 used in BLOCK CONSTRUCTION ($L_{\max}$) are modified as

$$f(k, i, j, S, \ell) = \begin{cases} w_i(a_i + c_i + b_i), & \text{if } k = A, i = j, S = a_i + c_i, \text{ and } \ell = b_i; \\ w_i(a_i + c_i + b_i), & \text{if } k = B, i = j, S = a_i, \text{ and } \ell = c_i + b_i; \\ \infty, & \text{otherwise,} \end{cases}$$

$$y_1 = f(k, i, j-1, S - a_j - c_j, \ell + a_j + c_j - b_j) + w_j(S + \ell),$$

$$y_2 = \begin{cases} f(k, i, j-1, S - a_j, \ell + a_j - c_j - b_j) + w_j(S+\ell), & \text{if } \ell \ge c_j + b_j; \\ \infty, & \text{otherwise.} \end{cases}$$

Definitions of initial conditions, z_1, and z_2 used in SCHEDULE CONCATENATION ($L_{\max}$) are also modified as

$$g(k, i) = \begin{cases} 0, & \text{if } i = n+1, ; \\ \infty, & \text{otherwise.} \end{cases}$$

$$z_1 = \begin{cases} f(k, i, j, S, \ell) + g(A, j+1) + w_{[j+1:n]}S, & \text{if } \ell < a_{j+1} + c_{j+1}; \\ \infty, & \text{otherwise,} \end{cases}$$

$$z_2 = \begin{cases} f(k, i, j, S, \ell) + g(B, j+1) + w_{[j+1:n]}S, & \text{if } \ell < a_{j+1}; \\ \infty, & \text{otherwise.} \end{cases}$$

Theorem 4 *The studied fixed-job-sequence problem for minimizing the total weighted completion time is pseudo-polynomially solvable in $O(n^2(\sum_{h=1}^{n} c_h)^2)$ time.*

3.3 Total weighted tardiness and weighted number of tardy jobs

To design pseudo-polynomial time algorithms for the two objective functions, $\sum w_j T_j$ and $\sum w_j U_j$, some further modifications are needed. The tardiness status of the jobs within the blocks and partial schedules created via the procedures for $L_{\max}$ and $\sum w_j C_j$ cannot be fathomed. We therefore instead determine the optimal objective value for $\sum w_j T_j$ or $\sum w_j U_j$ within the blocks and partial schedules subject to the condition that the first job starts at a specified time point. In both BLOCK CONSTRUCTION and SCHEDULE CONCATENATION procedures, we introduce an extra parameter λ, which is the lead time to specify the exact total length before the start of the block or partial schedule. Define state $(k, i, j, S, \ell, \lambda)$ to include the subschedules for jobs $\{J_i, J_{i+1}, \ldots, J_j\}$ which satisfy the following conditions: (1) no idle time is inserted between any two consecutive operations on the machines; (2) the first flexible operation O_i^C is assigned to machine $k \in \{A, B\}$; (3) the machine-A processing span is exactly S; (4) the lag is exactly ℓ; (5) job i starts at time λ, which is the lead time strictly imposed in front of the block. Let function $f(k, i, j, S, \ell, \lambda)$ denote the optimal total weighted tardiness of the jobs $\{J_i, J_{i+1}, \ldots, J_j\}$ among the blocks in state $(k, i, j, S, \ell, \lambda)$. To facilitate notation, we denote the tardiness and the tardiness status of

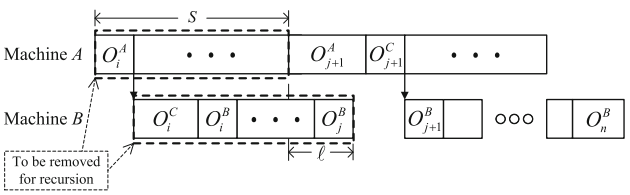

Fig. 5 Case-1 schedule concatenation ($L_{\max}$) for assembling a partial schedule in state (B, i)

job J_j completing at time t in some block, respectively, by $T_j(t) = \max\{0, t - d_j\}$ and

$$U_j(t) = \begin{cases} 1, & \text{if } t > d_j; \\ 0, & \text{otherwise.} \end{cases}$$

A dummy job J_0 with $a_0 = c_0 = 0$ is needed for the following procedures as shown below.

The procedures BLOCK CONSTRUCTION ($\sum w_j T_j$) and SCHEDULE CONCATENATION ($\sum w_j T_j$) both requires $O(n^2(\sum_{h=1}^{n} c_h)^3)$, in which the lead time parameter λ induces an extra term $\sum_{h=1}^{n} c_h$. The following theorem follows.

Theorem 5 *The studied fixed-job-sequence problem for minimizing the total weighted tardiness is pseudo-polynomially solvable in $O(n^2(\sum_{h=1}^{n} c_h)^3)$ time.*

BLOCK CONSTRUCTION ($\sum w_j T_j$)
Initial conditions:
for each λ satisfying $a_{[0:i-1]} \le \lambda \le a_{[0:i-1]} + c_{[0:i-1]}$,

$$f(k,i,j,S,\ell,\lambda) = \begin{cases} w_i T_i(\lambda + a_i + c_i + b_i), & \text{if } k = A, i = j, S = a_i + c_i, \text{ and } \ell = b_i; \\ w_i T_i(\lambda + a_i + c_i + b_i), & \text{if } k = B, i = j, S = a_i, \text{ and } \ell = c_i + b_i; \\ \infty, & \text{otherwise.} \end{cases} \quad (3)$$

Recursions:
for each 6-tuple $k, i, j, S, \ell, \lambda$ satisfying $k \in \{A, B\}$, $1 \le i < j \le n$, $a_{[i:j]} \le S \le a_{[i:j]} + c_{[i:j]}$, $\underline{\ell} \le \ell \le \overline{\ell}$, and $a_{[0:i-1]} \le \lambda \le a_{[0:i-1]} + c_{[0:i-1]}$,
/* The recursion runs only if $\overline{\ell} \ge c_j + b_j$.*/
Case 1 (Operation O_j^C going to machine A, as shown in Fig. 6):

$$y_1 = f(k, i, j-1, S - a_j - c_j, \ell + a_j + c_j - b_j, \lambda) + w_j T_j(\lambda + S + \ell), \quad (4)$$

Case 2 (Operation O_j^C going to machine B):

$$y_2 = \begin{cases} f(k, i, j-1, S - a_j, \ell + a_j - c_j - b_j, \lambda) + w_j T_j(\lambda + S + \ell), & \text{if } \ell \ge c_j + b_j; \\ \infty, & \text{otherwise,} \end{cases} \quad (5)$$

$f(k, i, j, S, \ell, \lambda) = \min\{y_1, y_2\}$.

With the derived blocks, we can then find an optimal complete schedule through the concatenation of optimal blocks. Let state (k, i, λ) include the partial schedules consisting of jobs $\{J_i, J_{i+1}, \ldots, J_n\}$ that are prefixed with an optimal block having operation O_i^C processed on machine k and job J_i starting exactly at time λ. Denote by $g(k, i, \lambda)$ the optimal total weighted tardiness among all the partial schedules in state (k, i, λ). The dynamic program for schedule concatenation with another dummy job J_{n+1} retaining $a_{n+1} = c_{n+1} = \infty$ is shown below.

SCHEDULE CONCATENATION ($\sum w_j T_j$)
Initial conditions:
for each 2-tuple k, λ satisfying $k \in \{A, B\}$, and $a_{[1:n]} \le \lambda \le a_{[1:n]} + c_{[1:n]}$,

$$g(k, i, \lambda) = \begin{cases} 0, & \text{if } i = n+1, ; \\ \infty, & \text{otherwise.} \end{cases}$$

Recursions:
for each 3-tuple k, i, λ satisfying $k \in \{A, B\}$, $1 \le i \le n$, and $a_{[0:i-1]} \le \lambda \le a_{[0:i-1]} + c_{[0:i-1]}$,

$$g(k, i, \lambda) = \min_{\substack{i \le j \le n; \\ a_{[i:j]} \le S \le a_{[i:j]} + c_{[i:j]}; \\ \underline{\ell} \le \ell \le \overline{\ell}}} \{z_1, z_2\},$$

where z_1 and z_2 are calculated as in the following:
Case 1 (The optimal block in $(k, i, j, S, \ell, \lambda)$ being attached to the optimal partial schedule in $(A, j+1, S + \lambda)$, as shown in Fig. 7):

$$z_1 = \begin{cases} f(k, i, j, S, \ell, \lambda) + g(A, j+1, S + \lambda), & \text{if } \ell < a_{j+1} + c_{j+1}; \\ \infty, & \text{otherwise,} \end{cases}$$

Case 2 (The optimal block in $(k, i, j, S, \ell, \lambda)$ being attached to the optimal partial schedule in $(B, j+1, S + \lambda)$):

$$z_2 = \begin{cases} f(k, i, j, S, \ell, \lambda) + g(B, j+1, S + \lambda), & \text{if } \ell < a_{j+1}; \\ \infty, & \text{otherwise.} \end{cases}$$

Goal: $\min_{k \in \{A,B\}} \{g(k, 1, 0)\}$.

The above two procedures for $\sum w_j T_j$ can be easily adapted for the minimization of weighted number of tardy jobs with the same time complexity by replacing function $T_j(\cdot)$ with $U_j(\cdot)$ in Eq. (3), (4), and (5).

Theorem 6 *The studied fixed-job-sequence problem for minimizing the weighted number of tardy jobs is pseudo-polynomially solvable in $O(n^2(\sum_{h=1}^{n} c_h)^3)$ time.*

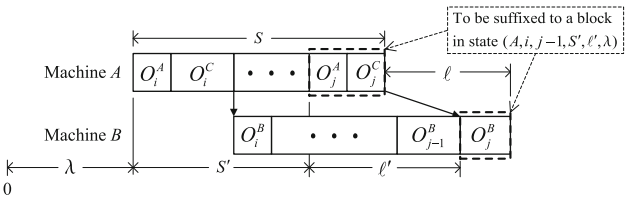

Fig. 6 Construction of a block in $(A, i, j, S, \ell, \lambda)$ from a block in $(A, i, j-1, S', \ell', \lambda)$

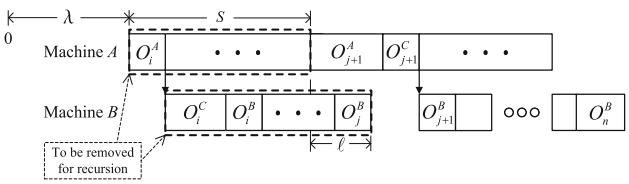

Fig. 7 Case-1 schedule concatenation ($\sum w_j T_j$) for assembling a partial schedule in state (B, i, λ)

Table 1 Average run times (in seconds) of 25 instances for $10 \leq n \leq 100$ and $100 \leq \sum_{h=1}^{n} c_h \leq 300$

n	$\sum_{h=1}^{n} c_h$																				
	100	110	120	130	140	150	160	170	180	190	200	210	220	230	240	250	260	270	280	290	300
10	0.11	0.13	0.13	0.13	0.14	0.16	0.18	0.18	0.19	0.21	0.21	0.23	0.24	0.27	0.25	0.27	0.27	0.31	0.31	0.31	0.32
15	0.22	0.24	0.25	0.31	0.34	0.38	0.37	0.42	0.43	0.47	0.49	0.51	0.50	0.62	0.60	0.63	0.61	0.75	0.73	0.71	0.97
20	0.40	0.43	0.46	0.50	0.58	0.63	0.67	0.81	0.57	0.62	0.67	0.68	0.72	0.75	0.76	0.80	0.89	0.96	0.97	1.08	1.04
25	0.42	0.47	0.58	0.66	0.71	0.63	0.81	0.89	0.87	0.89	1.02	1.04	1.02	1.06	1.19	1.10	1.35	1.47	1.36	1.59	1.43
30	0.67	0.64	0.74	0.74	0.89	0.96	1.00	1.14	1.21	1.19	1.38	1.44	1.60	1.86	1.54	1.61	1.84	1.82	1.90	2.24	2.04
35	0.68	0.90	0.92	1.27	1.06	1.15	1.45	1.41	1.62	1.83	1.78	1.86	2.19	2.13	2.36	2.22	2.31	2.54	2.74	2.72	2.65
40	0.83	1.11	1.33	1.36	1.46	1.68	1.61	1.69	1.78	2.32	2.16	2.35	2.53	2.73	2.74	2.79	3.10	2.93	3.25	3.46	3.39
45	1.60	1.48	1.58	1.82	2.13	2.27	2.11	2.22	2.24	2.67	2.47	2.92	3.04	3.32	3.34	3.11	3.78	4.09	4.42	4.12	4.20
50	1.21	1.70	1.78	1.90	2.14	2.26	2.28	2.68	2.73	2.79	3.30	3.52	3.40	3.85	4.15	4.45	4.69	4.39	5.39	5.46	5.76
55	1.92	1.83	2.14	2.54	2.45	2.57	3.28	3.00	3.64	3.73	4.17	4.15	4.28	4.49	4.69	4.87	5.59	5.56	5.79	5.93	5.82
60	2.05	2.28	2.63	3.04	3.16	3.32	3.43	3.90	4.31	3.79	4.26	4.70	5.52	5.18	5.78	5.97	6.40	5.99	6.81	6.76	7.20
65	2.36	2.65	3.08	3.30	3.68	4.00	4.04	4.25	4.28	5.35	5.31	5.39	5.17	5.81	7.24	7.21	6.85	8.06	7.49	8.29	7.78
70	2.37	2.98	3.20	3.33	3.88	4.33	4.66	4.57	5.34	5.16	6.07	5.64	6.29	7.14	6.98	7.18	8.24	7.81	8.43	9.31	9.83
75	2.57	2.92	3.67	4.10	4.33	4.82	5.11	5.52	5.71	6.47	7.04	7.20	7.73	7.39	8.09	8.50	8.77	10.14	10.22	10.82	9.78
80	3.06	3.50	4.10	4.23	5.11	5.10	4.96	6.24	6.08	7.18	7.28	8.82	8.87	8.73	8.51	9.13	10.25	10.92	12.72	11.22	11.38
85	3.59	4.25	4.17	5.21	5.26	5.88	6.56	7.19	7.20	7.84	8.67	9.00	8.49	9.42	10.41	10.38	10.20	11.44	13.09	12.00	14.07
90	4.45	4.50	5.35	5.00	6.21	7.42	6.98	6.71	7.47	9.15	9.00	9.30	9.20	11.72	10.46	11.47	11.52	12.46	13.18	13.23	13.60
95	4.62	5.45	6.32	5.60	6.07	7.15	7.08	8.34	8.88	9.56	10.61	11.16	12.20	11.20	12.64	14.39	13.30	15.78	13.54	14.57	17.19
100	4.80	5.11	5.67	7.70	7.43	8.29	7.94	8.80	9.35	10.56	11.42	11.78	12.70	13.26	13.67	14.54	14.19	14.48	17.82	15.80	20.19

3.4 Computational experiments

To demonstrate the practical performance of the developed dynamic programming framework, we conducted brief computational experiments for the objective function $C_{\max}$. The numerical experiments were implemented in Mathematica 10.1 on a laptop computer equipped with an Intel Core i5-4310M CPU, 8GB RAM and Windows 7 64-bit operating system. Regarding the experimental setting, the processing time a_j or b_j for each job $J_j \in \mathcal{J}$ was generated as an uniformly distributed random number within the interval [0, 20]. The processing times of the flexible operations were produced by a random partition of a given amount $\sum_{h=1}^{n} c_h$ into n values such that $c_j \geq 0$, $j \in \{1, \ldots, n\}$. The first experiment was performed for $10 \leq n \leq 100$ (with an interval of 5) and $100 \leq \sum_{h=1}^{n} c_h \leq 300$ (with an interval of 10). For each combination of n and $\sum_{h=1}^{n} c_h$, we generate 25 random test instances. The average run times of 25 instances for all combinations are given in Table 1, and the regression result is illustrated in Fig. 8, where the dots denote the experimental data of test instances and the associated best-fit regression polynomial reflects the computational complexity $O(n^2 \sum_{h=1}^{n} c_h)$.

The second experiment was concerned with large-scale instances, where $50 \leq n \leq 200$ (with an interval of 50) and $500 \leq \sum_{h=1}^{n} c_h \leq 1000$ (with an interval of 500). Table 2 summarizes the average and maximum run times of 25 random test instances. Given $n = 50$, even the maximum run time is no more than 26 s for the case with $\sum_{h=1}^{n} c_h = 1000$.

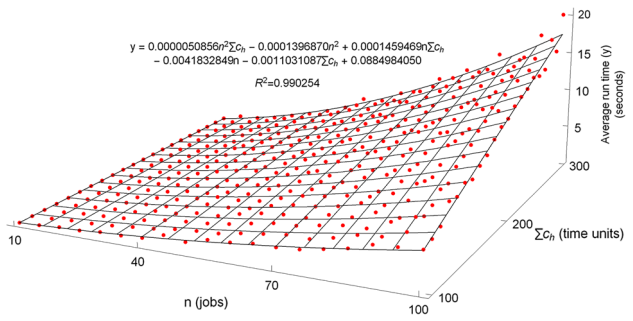

Fig. 8 Regression surface of the average run time and $(n, \sum_{h=1}^{n} c_h)$

Table 2 Average and maximum run times (in seconds) of 25 instances for $50 \leq n \leq 200$ and $500 \leq \sum_{h=1}^{n} c_h \leq 1000$

n	$\sum_{h=1}^{n} c_h = 500$		$\sum_{h=1}^{n} c_h = 1000$	
	Average	Maximum	Average	Maximum
50	9.81	13.03	20.49	25.09
100	30.92	44.09	71.67	105.35
150	67.34	107.56	160.44	198.54
200	117.83	168.03	295.79	399.22

Table 3 Complexity results of the studied fixed-job-sequence problems

Performance metric	Complexity	Run time	Remark
$C_{\max}$	Ordinary NP-hard	$O(n^2 \sum_{h=1}^{n} c_h)$	Theorem 2
$L_{\max}$	Ordinary NP-hard	$O(n^2 (\sum_{h=1}^{n} c_h)^2)$	Theorem 3
$\sum C_j$	Open	$O(n^2 (\sum_{h=1}^{n} c_h)^2)$	Theorem 4
$\sum w_j C_j$	Ordinary NP-hard	$O(n^2 (\sum_{h=1}^{n} c_h)^2)$	Theorem 4
$\sum w_j T_j$	Ordinary NP-hard	$O(n^2 (\sum_{h=1}^{n} c_h)^3)$	Theorem 5
$\sum w_j U_j$	Ordinary NP-hard	$O(n^2 (\sum_{h=1}^{n} c_h)^3)$	Theorem 6

If $n = 200$ is considered, then the average run time is less than 118 s for $\sum_{h=1}^{n} c_h = 500$ and 296 s for $\sum_{h=1}^{n} c_h = 1000$.

4 Conclusions

This paper investigated the problem of scheduling three-operation jobs in a two-machine flowshop subject to a given job processing sequence for five standard performance metrics, viz. $C_{\max}$, $L_{\max}$, $\sum w_j C_j$, $\sum w_j T_j$, and $\sum w_j U_j$. We showed that the studied problem is ordinary NP-hard even if the processing times of the preassigned operations are all zero. Although the extremely restricted case remains NP-hard, the general setting is pseudo-polynomially solvable with the development of dynamic programs. The detailed complexity results of the studied fixed-job-sequence problem is provided in Table 3. It shall be highlighted that this study is mainly aimed at contributing to the three-operation flowshop scheduling problem from the perspective on solution technique development.

For future research, we could first clarify the complexity status of the total completion time minimization problem, the polynomial solvability of which is not ruled out. The following two observations suggest that a polynomial-time dynamic programming algorithm design similar to Hwang et al. (2012, 2014) may not be applicable to the $\sum C_j$ problem:

(1) In the development of a dynamic program with backward (or forward) recursion for the open problem, the shape of the block (or partial schedule) cannot be fixed with a constant number of state variables the ranges of which are polynomial in the number of jobs, i.e., the machine-A or machine-B processing span of a block (or partial schedule) cannot be realized without specifying exactly which flexible operations are on machine A or B. Thus, it is inevitable to include a temporal state variable S to enumerate the machine-A processing span in the recursive function.
(2) If the aforementioned difficulty can be overcome by some technique, then we will be able to avoid the temporal state

variable ℓ with the same approach. It thus will make the studied problem for the makespan minimization polynomially solvable, which contradicts the NP-hardness result in Theorem 1.

Therefore, the development of an algorithm other than block-based dynamic program is suggested for investigating the polynomial solvability. It would be essential to delve into the structural properties of schedules with respect to the objective function $\sum C_j$. As for the conjecture about NP-hardness, an unconventional proving scheme could be necessary for the flexibility in job manipulation could be limited by the fixed-job-sequence assumption. Another direction is to devise fully polynomial time approximation schemes for the five considered performance metrics. A further extension is to investigate the generalized fixed-sequence scenario where the requirement for the flexible operation being immediately preceded or followed by the corresponding preassigned operations is relaxed. This relaxation may render much more interleaving combinations of operations on either machine, and make the problem more challenging. The consideration of preemption, particularly for the flexible operations could also be an interesting extension.

Acknowledgements The authors are grateful to the editors and the anonymous reviewers for their constructive comments that have improved earlier versions of this paper. Lin was partially supported by the Ministry of Science and Technology of Taiwan under the Grant MOST 104-2410-H-009-029-MY2.

References

Crama, Y., & Gultekin, H. (2010). Throughput optimization in two-machine flowshops with flexible operations. *Journal of Scheduling*, *13*(3), 227–243.

Gupta, J. N., Koulamas, C. P., Kyparisis, G. J., Potts, C. N., & Strusevich, V. A. (2004). Scheduling three-operation jobs in a two-machine flow shop to minimize makespan. *Annals of Operations Research*, *129*(1), 171–185.

Gultekin, H. (2012). Scheduling in flowshops with flexible operations: Throughput optimization and benefits of flexibility. *International Journal of Production Economics*, *140*(2), 900–911.

Hwang, F. J., Kovalyov, M. Y., & Lin, B. M. T. (2012). Total completion time minimization in two-machine flow shop scheduling problems with a fixed job sequence. *Discrete Optimization*, *9*(1), 29–39.

Hwang, F. J., Kovalyov, M. Y., & Lin, B. M. T. (2014). Scheduling for fabrication and assembly in a two-machine flowshop with a fixed job sequence. *Annals of Operations Research*, *217*(1), 263–279.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, *1*(1), 61–68.

Lin, B. M. T., & Hwang, F. J. (2011). Total completion time minimization in a 2-stage differentiation flowshop with fixed sequences per job type. *Information Processing Letters*, *111*(5), 208–212.

Lin, B. M. T., Hwang, F. J., & Kononov, A. V. (2016). Relocation scheduling subject to fixed processing sequences. *Journal of Scheduling*, *19*(2), 153–163.

Shafransky, Y. M., & Strusevich, V. A. (1998). The open shop scheduling problem with a given sequence of jobs on one machine. *Naval Research Logistics*, *45*(7), 705–731.

Uruk, Z., Gultekin, H., & Akturk, M. S. (2013). Two-machine flowshop scheduling with flexible operations and controllable processing times. *Computers & Operations Research*, *40*(2), 639–653.