Discrete Optimization 9 (2012) 29–39

Contents lists available at SciVerse ScienceDirect

Discrete Optimization

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

Total completion time minimization in two-machine flow shop scheduling problems with a fixed job sequence

F.J. Hwang [a], M.Y. Kovalyov [b], B.M.T. Lin [a,*]

[a] Institute of Information Management, Department of Information and Finance Management, National Chiao Tung University, Taiwan

[b] United Institute of Informatics Problems, National Academy of Sciences of Belarus, Belarus

ARTICLE INFO

Article history:
Received 9 January 2010
Received in revised form 11 November 2011
Accepted 14 November 2011
Available online 6 December 2011

Keywords:
Two-machine flow shop
Total completion time
Fixed sequence
Dynamic programming

ABSTRACT

This paper addresses scheduling n jobs in a two-machine flow shop to minimize the total completion time, subject to the condition that the jobs are processed in the same given sequence on both machines. A new concept of optimal schedule block is introduced, and polynomial time dynamic programming algorithms employing this concept are derived for two specific problems. In the first problem, the machine-2 processing time of a job is a step increasing function of its waiting time between the machines, and a decision about machine-1 idle time insertion has to be made. This problem is solved in $O(n^2)$ time. In the second problem, the jobs are processed in batches and each batch is preceded by a machine-dependent setup time. An $O(n^5)$ algorithm is developed to find an optimal batching decision.

1. Introduction

This paper studies two scheduling problems in a two-machine flow shop, subject to the condition that the jobs are processed in the same fixed sequence on both machines. A new *block approach* is proposed and deployed to determine their solutions in polynomial time within the general dynamic programming framework.

1.1. State of the art

In machine scheduling, a schedule specifies job starting and completion times on the machines. In some cases, it is completely characterized by the job sequence, and in the other cases, extra information is needed to fully specify it. A complicated schedule structure is typical for two major types of scheduling: scheduling with inserted machine idle time [1] and batch scheduling [2]. A commonly used approach to tackle NP-hard problems of these types is to derive an optimal polynomial time algorithm for the problem in which jobs are processed in the same fixed sequence. Then this algorithm can be used in a local search method to evaluate candidate job sequences for the general problem. There also exist studies, in which a specific optimal job sequence is established analytically. In real-life applications, a job sequence can be given as an input data of the problem. Shafransky and Strusevich [3] studied several open shop scheduling problems, subject to a given sequence of jobs on one machine, and mentioned that a pre-assigned sequence of jobs could be retained on one of the machines in the manufacturing process owing to technological or managerial decisions. A fixed job sequence can also appear due to the application of the First-Come–First-Served principle, which is commonly regarded fair by customers.

* Corresponding author. Tel.: +886 3 5131472; fax: +886 3 5723792.
E-mail addresses: fengjanghwang@gmail.com (F.J. Hwang), kovalyov_my@newman.bas-net.by (M.Y. Kovalyov), bmtlin@mail.nctu.edu.tw, bmtlin@gmail.com (B.M.T. Lin).

1572-5286/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.disopt.2011.11.001

Algorithms with machine idle time insertion for a fixed job sequence, also called timing algorithms [4], were applied for a single machine [5–11], parallel machines [12] and a flow shop [13,4]. The problem studied by Lin and Cheng [13] belongs to the area of scheduling with time-dependent processing times, as the machine-2 processing time depends on job's waiting time between the machines. To avoid extra processing on machine 2, idle times can be inserted on machine 1. Scheduling with time-dependent processing times was surveyed by Cheng et al. [14].

Batching decisions for a fixed job sequence were studied by Cheng et al. [15] and Ng and Kovalyov [16] for the makespan minimization in a flow shop. Considering the fabrication and assembly of components in a two-machine flow shop, Cheng and Wang [17] studied the problem of batching the jobs sequenced according to Johnson's rule or the agreeable processing time condition for makespan minimization. As for the assembly-type flowshop batching problem with a fixed job sequence, Lin et al. [18] investigated the makespan minimization for the three-machine case. Later, Hwang and Lin [19] addressed the four regular objective functions of total completion time, maximum lateness, total tardiness, and number of tardy jobs for the general $(m+1)$-machine case. Lin and Cheng [20] also considered the objective of maximum lateness, weighted number of tardy jobs or total weighted completion time for a fixed job sequence with centralized or decentralized batching decisions in concurrent open shops.

In this paper, we consider two problems of total completion time minimization in a two-machine flow shop, subject to a fixed job sequence. Though there exist results for the makespan ($C_{\max}$) minimization in these settings, there is no result for the minimization of total completion time, which is one of the most popular and practically relevant criteria in the scheduling literature.

1.2. Formulation of problems and block approach

Denote the processing times of job j on machine 1 and machine 2 by p_{1j} and p_{2j}, respectively. The first problem includes a step increasing job processing time on machine 2, which depends on the job waiting time between the machines. We call this waiting time a *time lag* between the two operations of a job. For job j, it is denoted by ℓ_j and, given a schedule, it is equal to the difference between the starting time of this job on machine 2 and its completion time on machine 1. A specified time delay t_j is also associated with each job j. If $\ell_j \leq t_j$, then $p_{2j} = \alpha_j$; otherwise $p_{2j} = \alpha_j + \beta_j$. Besides the given job sequence, a schedule should specify machine-1 idle times, which are used to avoid an increase of the job processing times on machine 2, and the time lags between the two operations of each job. Similar to Lin and Cheng [13], who studied the makespan minimization problem in the same setting, our first problem is denoted by $F2|\mathit{fixed\_seq}, p_{2j} \in \{\alpha_j, \alpha_j + \beta_j\}|\sum C_j$, where $F2$ stands for a two-machine flow shop, *fixed_seq* for a fixed job sequence, and $\sum C_j$ for the total completion time minimization criterion. The problem is motivated by the manufacturing operations in which temperature regime plays a role.

The second problem is to batch jobs in a predefined sequence. The batches are consistent in the sense that their formations are the same on both machines. Job completion times are determined according to the batch availability model such that jobs of the same batch complete on the same machine at the same time, when processing of the last job in this batch has been finished. The processing time of a batch B on machine l is equal to the sum of the machine setup time, s_l, and job processing times, $\sum_{j\in B} p_{lj}$. The setups are non-anticipatory, i.e., a setup on machine 2 can start only after the corresponding batch is completed on machine 1 and machine 2 is unoccupied. Following Ng and Kovalyov [16], we denote this problem by $F2|\mathit{fixed\_seq}, \mathit{sum\text{-}batch}, \mathit{consi}, s_l|\sum C_j$, where *sum-batch* stands for the batch processing time calculation formula, *consi* for consistent batches, and s_l for non-anticipatory machine-dependent setup times. Motivation for this problem comes from manufacturing environments in which items move between facilities in containers such as pallets, boxes or carts.

In the two problems under study, the difficulty in the design of a polynomial time dynamic programming algorithm stems from the potential conflict between the makespan and the total completion time. A subschedule which minimizes the total completion time for the first r jobs may retain a relatively large makespan which worsens the total completion time for the remaining $n-r$ jobs. To resolve the difficulty that hinders the principle of optimality, a mechanism for fixing the processing lengths of subschedules is required. A common approach is to introduce temporal parameters into the dynamic programs. This however gives rise to pseudo-polynomial running times. This study develops a two-phase dynamic programming framework without including any temporal parameters as state variables. We present a concept that a feasible schedule can be divided into several specific subschedules, each of which is characterized by its *shape*. On the Gantt chart, the shape of a subschedule in a two-machine flow shop is determined by the positions of its end on machine 1, its start on machine 2 and its end on machine 2, relative to its start on machine 1. Fig. 1 demonstrates that the shape of a subschedule consisting of jobs $\{i, \ldots, j\}$ is settled by the three-tuple (L_1, L_2, L_3). A specific subschedule is called as a *block* and associated with a *state* in the phase-1 dynamic program. A specific state could be associated with more than one block. These blocks in the same state may have different sums of job completion times, while they must retain an identical shape. Based upon this background idea, the concept of *optimal* (*schedule*) *block* is introduced. The phase-1 dynamic program produces for each state an *optimal block*, which is a block attaining the minimum total completion time among all blocks associated with the same state. Then an optimal schedule can be built by concatenating several optimal blocks in the phase-2 dynamic program, which is formulated with backward recursion to establish the validity of the principle of optimality in the studied problems.

For the two studied problems, we utilize idle times to divide a feasible schedule into several blocks. Any two consecutive blocks for problems $F2|\mathit{fixed\_seq}, p_{2j} \in \{\alpha_j, \alpha_j + \beta_j\}|\sum C_j$ and $F2|\mathit{fixed\_seq}, \mathit{sum\text{-}batch}, \mathit{consi}, s_l|\sum C_j$ are separated by machine-1 and machine-2 idle times, respectively.

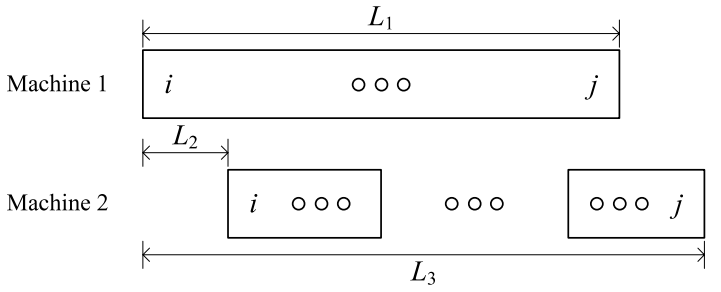

Fig. 1. Illustration for the shape of a subschedule in a two-machine flow shop.

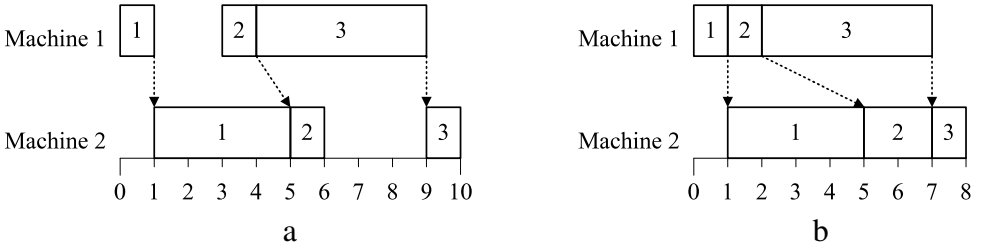

Fig. 2. Two example schedules.

In Sections 2 and 3, block structures are defined for each problem and then employed in the development of polynomial time dynamic programming algorithms. Without the fixed sequence assumption, the two studied problems are both NP-hard, as will be explained in the corresponding sections. Section 4 contains a summary of the results and suggestions for future research.

In the sequel, we assume without loss of generality that the fixed job sequence is $\langle 1, 2, \ldots, n\rangle$.

2. Problem $F2|\textit{fixed\_seq}, p_{2j} \in \{\alpha_j, \alpha_j + \beta_j\}|\sum C_j$

In many manufacturing environments, it is realistic to consider jobs with time-dependent processing times rather than static and fixed processing times. Lin and Cheng [13] proved the strong NP-hardness of the $F2|p_{2j} \in \{\alpha_j, \alpha_j + \beta_j\}|C_{\max}$ problem, and designed an $O(n^2)$ time solution algorithm for the case with a given job sequence. Minimizing the total completion time without the assumption of a fixed job sequence is strongly NP-hard because the classical $F2 \parallel \sum C_j$ problem corresponds to the special case with $\beta_j = 0$ or $t_j = \infty$ for all jobs.

The challenge of problem $F2|\textit{fixed\_seq}, p_{2j} \in \{\alpha_j, \alpha_j + \beta_j\}|\sum C_j$ is how to make an optimal decision about the machine-1 idle times to avoid extra processing penalty β_j on machine 2. Assigning jobs in the given sequence one by one at the earliest times on the machines and inserting an idle time in front of job j on machine 1 if the time lag ℓ_j exceeds t_j cannot guarantee an optimal schedule. Consider the following instance with three jobs: $(p_{11}, t_1, \alpha_1, \beta_1) = (1, 1, 4, 1)$, $(p_{12}, t_2, \alpha_2, \beta_2) = (1, 1, 1, 1)$, and $(p_{13}, t_3, \alpha_3, \beta_3) = (5, 1, 1, 1)$. The total completion time of the schedule obtained by the above-mentioned method is 21 (Fig. 2(a)) while the optimal objective value is 20 (Fig. 2(b)).

For this studied problem, we treat a subschedule in which no machine-1 idle time exists between any two consecutive jobs as a block. An optimal schedule can be either a single block consisting of all jobs $\{1, \ldots, n\}$, or a concatenation of several blocks such that there is a machine-1 idle time immediately preceding each block except for the first one. A block is specified as a *maximal*, by inclusion, subsequence of jobs which are consecutively processed without any idle time between them on machine 1, and characterized by the time lag between the two operations of the first job of this subsequence. The remaining jobs of a block start on machine 2 at the earliest possible times. It is clear that the time lag for job 1 of the first block in an optimal schedule is equal to zero, i.e. $\ell_1 = 0$.

We associate a block consisting of jobs $\{i, \ldots, j\}$ with the two-tuple (i, j), and denote its makespan by $T(i, j)$. The first block $(1, j)$, $1 \le j \le n$, is illustrated in Fig. 3(a). For each block (i, j) with $2 \le i \le j \le n$, the time lag ℓ_i has to be determined.

Lemma 1. *Given that schedules comprise at least two blocks for problem $F2|\textit{fixed\_seq}, p_{2j} \in \{\alpha_j, \alpha_j + \beta_j\}|\sum C_j$, there exists an optimal schedule, in which the time lag ℓ_i between the two operations of job i in each block (i, j) is exactly t_i for $2 \le i \le j \le n$.*

Proof. Assume an optimal schedule where there are block(s) not satisfying the claimed property. Consider the block (i, j) with the smallest $i \ge 2$. Let I_i be the idle time preceding the block (i, j) on machine 1. If $\ell_i > t_i$, then $p_{2i} = \alpha_i + \beta_i$. Shift job i on machine 1 to the left such that the idle time I_i on machine 1 is eliminated. The movement does not increase p_{2i}, and the optimality of the schedule remains. In the new schedule, block (i, j) vanishes. On the other hand, if $\ell_i < t_i$, then $p_{2i} = \alpha_i$.

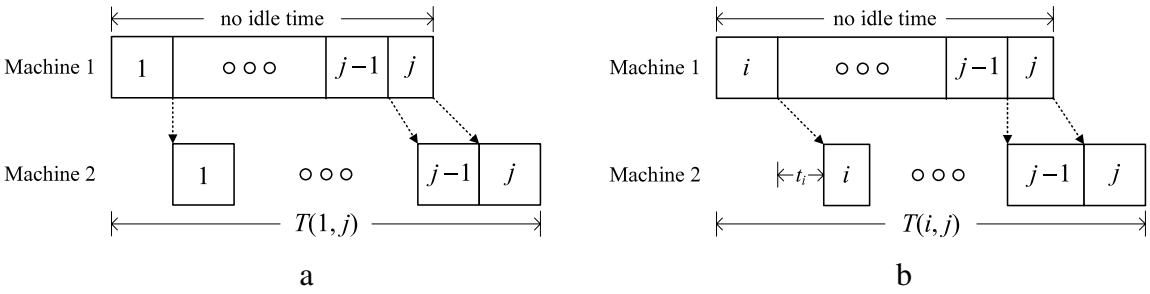

Fig. 3. Structure of (a) the first block $(1, j)$ for $1 \le j \le n$, and (b) block (i, j) for $2 \le i \le j \le n$.

Shift job i on machine 1 to the left by $\min\{I_i, t_i - \ell_i\}$ time units so that either the idle time I_i is eliminated or the time lag becomes t_i. Similarly, the movement does not increase p_{2i}, and the optimality of the schedule remains. In the new schedule, either block (i, j) vanishes or $\ell_i = t_i$ holds. Repeating the described shifting procedure, whenever necessary, can produce an optimal schedule as specified in the lemma. □

By Lemma 1, we assume that $\ell_i = t_i$ for any block (i, j), $2 \le i \le j \le n$. The structure of block (i, j) for $2 \le i \le j \le n$ is shown in Fig. 3(b).

For problem $F2|fixed\_seq, p_{2j} \in \{\alpha_j, \alpha_j + \beta_j\}|\sum C_j$, there exists an optimal schedule which is a single block $(1, n)$ or a concatenation of r blocks, $(1, j_1), (j_1 + 1, j_2), \ldots, (j_{r-1} + 1, n)$. Note that each two-tuple (i, j) is associated with only one block, i.e., each block itself is an optimal block. All blocks (i, j) and their corresponding makespan values $T(i, j)$, $1 \le i \le j \le n$, can be built by the following pre-processing procedure. To facilitate further presentation, denote $p_{1[i:j]} = \sum_{h=i}^{j} p_{1h}$. We have $T(1, 1) = p_{11} + \alpha_1$, and $T(i, i) = p_{1i} + t_i + \alpha_i$, $2 \le i \le n$. For $1 \le i < j \le n$,

$$T(i, j) = \begin{cases} T(i, j-1) + \alpha_j + \beta_j, & \text{if } T(i, j-1) > p_{1[i:j]} + t_j; \\ \max\{T(i, j-1), p_{1[i:j]}\} + \alpha_j, & \text{otherwise.} \end{cases}$$

In the second phase, we design a backward dynamic programming algorithm, denoted as DP-Idle-Time, in which a single block is added as the prefix of a partial schedule of jobs $\{i, \ldots, n\}$. Such a partial schedule is said to be in state i. Algorithm DP-Idle-Time recursively calculates the value $g(i)$, which is the minimum total completion time of the partial schedule(s) in state i, $1 \le i \le n$. Fig. 4 illustrates the recursion scenario of algorithm DP-Idle-Time. A dummy job $n + 1$ with $p_{1,n+1} = t_{n+1} = \alpha_{n+1} = \beta_{n+1} = 0$ is needed for the boundary condition in algorithm DP-Idle-Time. Denote inequality

$$T(i, i' - 1) > p_{1[i:i']} + t_{i'} \tag{1}$$

by condition A.

Algorithm DP-Idle-Time

Initialization:

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

Recursion: (Refer to Fig. 4)

For each i from n down to 1 do

$$g(i) = \min_{i+1 \le i' \le n+1} \begin{cases} g(i') + (n - i' + 1)(T(i, i' - 1) - p_{1i'} - t_{i'}) + \sum_{h=i}^{i'-1} T(i, h), & \text{if condition A;} \\ \infty, & \text{otherwise.} \end{cases} \tag{2}$$

Goal: Calculate $g(1)$.

The optimal objective value is $g(1)$ and the corresponding optimal schedule can be retrieved by backtracking.

The algorithm is justified as follows. For this problem, we aim to make an optimal decision about which machine-1 operations are to be deferred so that the total completion time is minimized. A crude enumeration requires evaluation of 2^{n-1} combinations of decisions on whether to defer each of the jobs $\{2, \ldots, n\}$. All candidate solutions can be constructed by the proposed block concatenation. We avoid full enumeration by using algorithm DP-Idle-Time, which in each state i eliminates $n - i$ candidate partial schedules that are impossible to be extended to constitute an optimal complete schedule. Since the backward recursion is performed, the processing length of the partial schedule in state i with value $g(i)$ does not affect the remaining unscheduled jobs $\{1, \ldots, i - 1\}$ and the optimality of a partial schedule can be kept. In Eq. (2), the optimal values of all possible 'previous' states i' are examined and used to derive the solution of the current state i. As

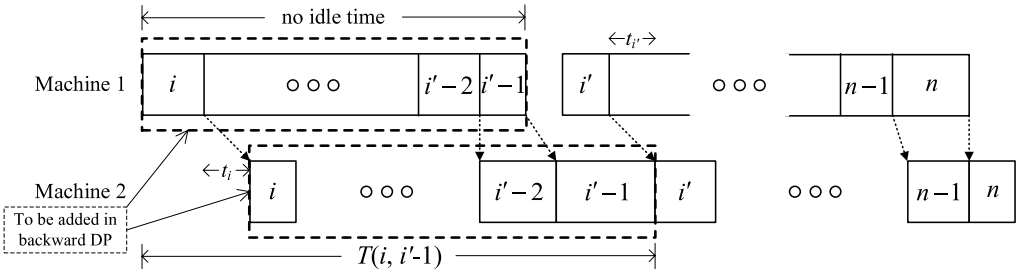

Fig. 4. Illustration for algorithm DP-Idle-Time.

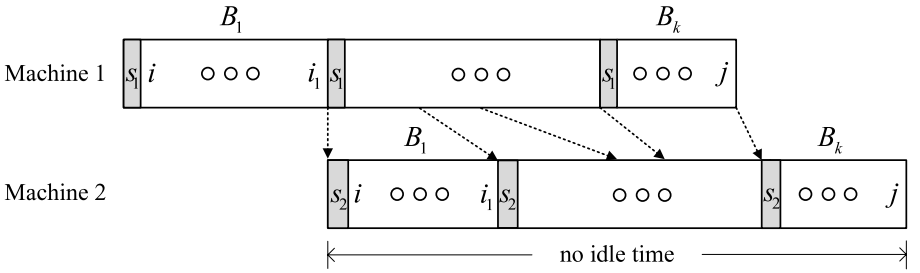

Fig. 5. Structure of block (i, i_1, j, k).

illustrated in Fig. 4, condition A is employed to guarantee that a partial schedule in state i can be constructed by articulating block $(i, i'-1)$ with the partial schedule in state i' which possesses value $g(i')$. The 'min' operation for i' from $i+1$ up to $n+1$ in Eq. (2) determines the value $g(i)$, and a partial schedule in state i with value $g(i)$ *dominates* all other partial schedules in the same state. This claim can be proved by induction on i. Denote an arbitrary partial schedule in state i and its total completion time by σ_i and $\nu(\sigma_i)$, respectively. We obviously have the induction base $g(n+1) \leq \nu(\sigma_{n+1})$. The induction hypothesis is that given $j \leq n+1$ the inequality $g(h) \leq \nu(\sigma_h)$ holds for all $h \in \{j, \ldots, n+1\}$. Then we prove that $g(j-1) \leq \nu(\sigma_{j-1})$. For some j' with $j \leq j' \leq n+1$, we assume that there exists a partial schedule σ_{j-1} with a leading block $(j-1, j'-1)$ such that $\nu(\sigma_{j-1}) < g(j-1)$. Then we have $\nu(\sigma_{j-1}) = \sum_{h=j-1}^{j'-1} T(j-1, h) + (n-j'+1)(T(j-1, j'-1) - p_{1j'} - t_{j'}) + X$, where X is the total completion time of the partial schedule in state j' from which σ_{j-1} is built. From the induction hypothesis $X \geq g(j')$, it follows that $\nu(\sigma_{j-1}) \geq \sum_{h=j-1}^{j'-1} T(j-1, h) + (n-j'+1)(T(j-1, j'-1) - p_{1j'} - t_{j'}) + g(j') \geq g(j-1)$, where the last inequality is due to Eq. (2). A contradiction arises. We establish $g(h) \leq \nu(\sigma_h)$ for $1 \leq h \leq n$ by induction, and the claim is proved. Hence, the principle of optimality holds and algorithm DP-Idle-Time can generate an optimal solution. An optimal schedule is the schedule in state $i = 1$ with value $g(1)$, which is either a concatenation of the first block $(1, i'-1)$ and a partial schedule in state i' with value $g(i')$ for $2 \leq i' \leq n$, or the single block $(1, n)$.

As for the required running time, it is not hard to see that computing $T(i, j)$ for $1 \leq i \leq j \leq n$ requires $O(n^2)$ time, and computing the value $g(i)$ requires $O(n)$ time for each i. Hence, the running time of algorithm DP-Idle-Time is $O(n^2)$. Our discussion is summarized in the following theorem. A numerical example is given in Appendix A for illustration.

Theorem 1. *Problem $F2|fixed\_seq, p_{2j} \in \{\alpha_j, \alpha_j + \beta_j\}| \sum C_j$ can be solved in $O(n^2)$ time.*

3. Problem $F2|$*fixed_seq*, *sum-batch*, *consi*, $s_l|\sum C_j$

The $F2|sum\text{-}batch, consi, s_l|\sum C_j$ problem is also strongly NP-hard because it is equivalent to the classical problem $F2 \parallel \sum C_j$ if setup times are equal to zero. The $F2|sum\text{-}batch, consi, s_l|C_{\max}$ problem was studied by Cheng et al. [15], Glass et al. [21] and Kovalyov et al. [22]. NP-hardness proofs, polynomial time algorithms for special cases and heuristics were presented in these studies. For the case of a fixed job sequence, Cheng et al. [15] developed an $O(n^3)$ algorithm, and Ng and Kovalyov [16] designed an $O(n^{5m-7})$ dynamic programming algorithm if there are m ($m \geq 2$) machines.

For problem $F2|fixed\_seq, sum\text{-}batch, consi, s_l|\sum C_j$, we define a block as a subschedule associated with a state (i, i_1, j, k), where i and j are the first and the last jobs of the block, respectively, k is the number of batches in the block, and i_1 is the last job of the first batch in the block. It is assumed that the first batch of the block is *critical* in the sense that, for this block, no idle time exists on machine 2 between the completion of the first batch on machine 1 and the completion of the last batch on machine 2. The structure of a block (i, i_1, j, k) is shown in Fig. 5.

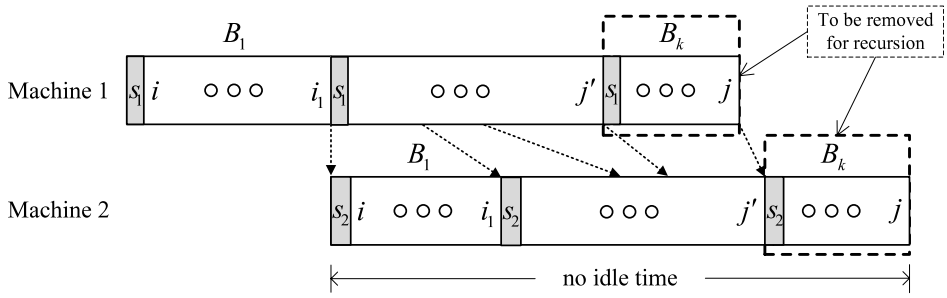

Fig. 6. Illustration for calculation of $f(i, i_1, j, k)$.

Notice that a state can be associated with one or more blocks. Let $f(i, i_1, j, k)$ denote the minimum total completion time of jobs in a block among all blocks associated with the same state (i, i_1, j, k). A block with value $f(i, i_1, j, k)$ is an optimal block in state (i, i_1, j, k). Denote $p_{l[i:j]} = \sum_{h=i}^{j} p_{lh}$ for $l = 1, 2$, and denote relation

$$(k-1)s_1 + p_{1[i_1+1:j]} \le (k-1)s_2 + p_{2[i:j']} \tag{3}$$

by condition B. The phase-1 dynamic program calculates values $f(i, i_1, j, k)$ for $1 \le i \le i_1 \le j \le n$ and $1 \le k \le j - i_1 + 1$ by forward recursion.
Forward recursion for $f(i, i_1, j, k)$
Initialization:

$$f(i, i_1, j, k) = \begin{cases} (i_1 - i + 1)(s_1 + p_{1[i:i_1]} + s_2 + p_{2[i:i_1]}), & \text{if } j = i_1, \text{ and } k = 1; \\ \infty, & \text{otherwise.} \end{cases}$$

Recursion: (Refer to Fig. 6)
For each feasible i, i_1, j, k satisfying $1 \le i \le i_1 \le n-1$, $i_1 + 1 \le j \le n$, $2 \le k \le j - i_1 + 1$ do

$$f(i, i_1, j, k) = \min_{i_1+k-2 \le j' \le j-1} \begin{cases} f(i, i_1, j', k-1) + (j - j')(s_1 + p_{1[i:i_1]} + ks_2 + p_{2[i:j]}), & \text{if condition B;} \\ \infty, & \text{otherwise.} \end{cases} \tag{4}$$

In the second phase, we construct an optimal schedule by concatenating several optimal blocks. In the following backward dynamic programming algorithm, named as DP-Batch, an optimal block is added as the prefix of a partial schedule of jobs $\{i, \ldots, n\}$ in state (i, i_1), where i and i_1 are respectively the first and the last jobs of the first batch in the partial schedule. The corresponding minimum total completion time is denoted by $g(i, i_1)$. Algorithm DP-Batch recursively calculates values $g(i, i_1)$, $1 \le i \le i_1 \le n$. A dummy job $n+1$ with $p_{1,n+1} = \infty$ and $p_{2,n+1} = 0$ is used for the boundary condition. Denote relations

$$i_1 + 1 \le i' \le n+1, i' \le i'_1 \le n + \left\lfloor \frac{i'}{n+1} \right\rfloor, 2 - \left\lfloor \frac{i_1+1}{i'} \right\rfloor \le k \le i' - i_1 \tag{5}$$

by range C, and denote inequality

$$ks_1 + p_{1[i_1+1:i'_1]} > ks_2 + p_{2[i:i'-1]} \tag{6}$$

by condition D.
Algorithm DP-Batch
Initialization:

$$g(i, i_1) = \begin{cases} 0, & \text{if } i = i_1 = n+1; \\ \infty, & \text{otherwise.} \end{cases}$$

Recursion: (Refer to Fig. 7)
For each feasible i, i_1 satisfying $1 \le i \le i_1 \le n$ do

$$g(i, i_1) = \min_{\mathrm{C}} \begin{cases} g(i', i'_1) + (n - i' + 1)(ks_1 + p_{1[i:i'-1]}) + f(i, i_1, i'-1, k), & \text{if condition D;} \\ \infty, & \text{otherwise.} \end{cases} \tag{7}$$

Goal: Find $\min_{1 \le i_1 \le n} g(1, i_1)$.

The optimal objective value is $\min_{1 \le i_1 \le n} g(1, i_1)$ and the corresponding optimal schedule can be obtained by backtracking.

The algorithm is justified as follows. In this problem, a crude enumeration needs to evaluate 2^{n-1} combinations of decisions on whether to have a machine-dependent setup time inserted in prior to each of the jobs $\{2, \ldots, n\}$. We can

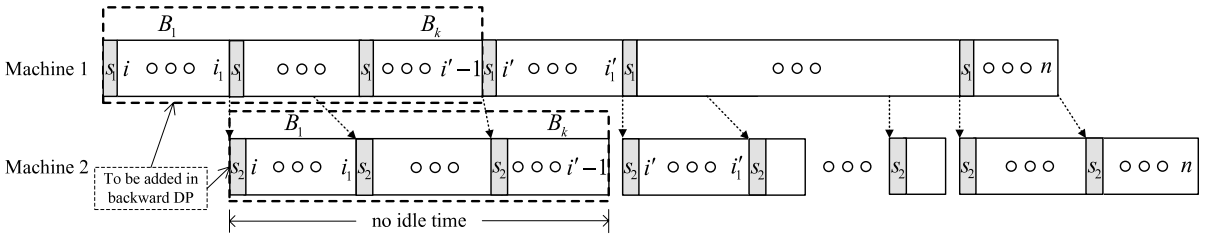

Fig. 7. Illustration for algorithm DP-Batch.

generate all candidate solutions by the presented block concatenation. In the phase-1 dynamic program, Eq. (4) examines all possible 'previous' states $(i, i_1, j', k-1)$ confined within range $i_1 + k - 2 \le j' \le j - 1$ to determine the solution of the current state (i, i_1, j, k). As demonstrated in Fig. 6, condition B is used to guarantee that a block in state (i, i_1, j, k) can be built by merging an optimal block $(i, i_1, j', k-1)$ with the single batch consisting of jobs $\{j'+1, \ldots, j\}$. Note that all blocks in the same state (i, i_1, j, k) retain an identical shape. As defined in Eq. (4), the optimal block (i, i_1, j, k), i.e. block (i, i_1, j, k) with value $f(i, i_1, j, k)$, *dominates* all other blocks in the same state in the sense that it has the minimum sum of job completion times among those of all blocks in this state. In the second phase, we develop algorithm DP-Batch by Eq. (7). With range C, all possible leading optimal blocks $(i, i_1, i'-1, k)$ and 'previous' states (i', i'_1) from which the current state (i, i_1) can be derived are considered. As shown in Fig. 7, condition D guarantees that a partial schedule in state (i, i_1) can be constructed by articulating the optimal block $(i, i_1, i'-1, k)$ with the partial schedule in state (i', i'_1) which has the value $g(i', i'_1)$. The value $g(i, i_1)$ is obtained by performing the 'min' operation in Eq. (7), and a partial schedule in state (i, i_1) with value $g(i, i_1)$ *dominates* all other partial schedules in the same state. This claim can be proved by induction. Denote an arbitrary partial schedule in state (i, i_1) and its total completion time by σ_{i,i_1} and $\nu(\sigma_{i,i_1})$, respectively. Obviously, the induction base $g(n+1, n+1) \le \nu(\sigma_{n+1,n+1})$ holds. Assume, as the induction hypothesis, that given $j \le n$ we have $g(h, h_1) \le \nu(\sigma_{h,h_1})$ for all h and h_1 with $j \le h \le h_1 \le n + \lfloor \frac{h}{n+1} \rfloor$. Then we prove that $g(j-1, j_1) \le \nu(\sigma_{j-1,j_1})$ for $j - 1 \le j_1 \le n$. Suppose that for some j' and j'_1 with $j_1 + 1 \le j' \le j'_1 \le n + \lfloor \frac{j'}{n+1} \rfloor$ there exists a partial schedule σ_{j-1,j_1} which is a concatenation of the optimal block $(j-1, j_1, j'-1, k)$ and a partial schedule in state (j', j'_1) such that $\nu(\sigma_{j-1,j_1}) < g(j-1, j_1)$. Then we have $\nu(\sigma_{j-1,j_1}) = f(j-1, j_1, j'-1, k) + (n - j' + 1)(ks_1 + p_{1[j-1:j'-1]}) + X$, where X is the total completion time of the partial schedule in state (j', j'_1) from which σ_{j-1,j_1} is built. By the induction hypothesis $X \ge g(j', j'_1)$, we can obtain that $\nu(\sigma_{j-1,j_1}) \ge f(j-1, j_1, j'-1, k) + (n - j' + 1)(ks_1 + p_{1[j-1:j'-1]}) + g(j', j'_1) \ge g(j-1, j_1)$, where the last inequality follows from Eq. (7). A contradiction arises. By induction, $g(h, h_1) \le \nu(\sigma_{h,h_1})$ for $1 \le h \le h_1 \le n$ is thus established. The principle of optimality holds and algorithm DP-Batch can produce an optimal schedule.

The two-phase dynamic programming algorithm consists of the construction of optimal blocks and the concatenation of optimal blocks. Since there are $O(n^4)$ states (i, i_1, j, k), each of which requires $O(n)$ operations, all the optimal blocks can be constructed in $O(n^5)$ time. Since there are $O(n^2)$ states (i, i_1), each of which requires $O(n^3)$ operations, the values $g(i, i_1)$ for $1 \le i \le i_1 \le n$ can be obtained in $O(n^5)$ time, too. Finally, the calculation of $\min_{1 \le i_1 \le n} g(1, i_1)$ requires $O(n)$ time. Hence, the running time of algorithm DP-Batch is $O(n^5)$ and the following theorem follows. A numerical example is provided in Appendix B for illustration.

Theorem 2. *Problem* $F2|fixed\_seq, sum\text{-}batch, consi, s_l|\sum C_j$ *can be solved in* $O(n^5)$ *time.*

4. Concluding remarks

This paper studied two scheduling problems in two-machine flow shops, in which the jobs are processed in a predetermined sequence on both machines and the objective is to minimize the total completion time. A new concept of optimal block was suggested and utilized to develop polynomial time dynamic programming algorithms. Problems $F2|fixed\_seq, p_{2j} \in \{\alpha_j, \alpha_j + \beta_j\}|\sum C_j$ and $F2|fixed\_seq, sum\text{-}batch, consi, s_l|\sum C_j$ were solved in $O(n^2)$ and $O(n^5)$ times, respectively.

Possible extensions of our research include the minimization of due date related performance metric, i.e. the maximum lateness, total tardiness or number of tardy jobs, and more general flow shop environments. For example, the general m-machine flow shop model can be of further interest. Further research on the second studied problem could be conducted by relaxing the restriction of consistent batches since there could exist an optimal schedule in which the batches are not consistent on two machines. Consider an instance with the following processing times and setup times: $p_{11} = p_{21} = 1$, $p_{12} = p_{22} = 2$, $s_1 = 5$, and $s_2 = 1$. In the optimal schedule, jobs 1 and 2 are grouped into the same batch on machine 1 and then processed with individual batches on machine 2. The max-batch model, in which the processing time of a batch is defined as the maximum processing time of the jobs contained in this batch, rather than the sum-batch model can also be considered in the second problem.

Acknowledgments

The authors are grateful to the anonymous reviewers for their constructive comments that have improved earlier versions of this paper. This research was supported in part by the National Science Council of Taiwan under the grants NSC-98-2410-H-009-011-MY2 and NSC-98-2912-I-009-002.

Appendix A. Example of algorithm DP-Idle-Time

Example 1. Consider the following instance with $n = 3 : (p_{11}, t_1, \alpha_1, \beta_1) = (1, 1, 3, 1)$, $(p_{12}, t_2, \alpha_2, \beta_2) = (1, 1, 1, 2)$, and $(p_{13}, t_3, \alpha_3, \beta_3) = (5, 1, 1, 1)$. The two-phase algorithm is demonstrated as follows.

Phase 1. (*Calculation of* $T(i, j)$)

$T(1, 1) = 1 + 3 = 4$;
$T(2, 2) = 1 + 1 + 1 = 3$;
$T(3, 3) = 5 + 1 + 1 = 7$;
$T(1, 2) = 4 + 1 + 2 = 7$;
$T(1, 3) = \max\{7, 7\} + 1 = 8$;
$T(2, 3) = \max\{3, 6\} + 1 = 7$.

Phase 2. (*Algorithm DP-Idle-Time*)

Boundary condition: $p_{1,4} = t_4 = \alpha_4 = \beta_4 = 0$.
Initialization: $g(4) = 0$; $g(1) = g(2) = g(3) = \infty$.
Recursion:
$i = 3$
$i' = 4$
$T(3, 3) = 7 > p_{1[3:4]} + t_4 = 5$
$z_1 = g(4) + 0 + T(3, 3) = 7$;
$g(3) = \min\{z_1\} = 7$.

$i = 2$
$i' = 3$
$T(2, 2) = 3 < p_{1[2:3]} + t_3 = 7$
$z_1 = \infty$;
$i' = 4$
$T(2, 3) = 7 > p_{1[2:4]} + t_4 = 6$
$z_2 = g(4) + 0 + \sum_{h=2}^{3} T(2, h) = 3 + 7 = 10$;
$g(2) = \min\{z_1, z_2\} = 10$.

$i = 1$
$i' = 2$
$T(1, 1) = 4 > p_{1[1:2]} + t_2 = 3$
$z_1 = g(2) + (4 - 2)(T(1, 1) - p_{12} - t_2) + T(1, 1) = 10 + 2(4 - 1 - 1) + 4 = 18$;
$i' = 3$
$T(1, 2) = 7 < p_{1[1:3]} + t_3 = 8$
$z_2 = \infty$;
$i' = 4$
$T(1, 3) = 8 > p_{1[1:4]} + t_4 = 7$
$z_3 = g(4) + 0 + \sum_{h=1}^{3} T(1, h) = 4 + 7 + 8 = 19$;
$g(1) = \min\{z_1, z_2, z_3\} = 18$.

Goal: $g(1) = 18$.

The corresponding optimal schedule can be constructed by backtracking the recursion, as illustrated in Fig. A.1.

Appendix B. Example of algorithm DP-Batch

Example 2. Consider the following instance with $n = 3 : (p_{11}, p_{21}) = (1, 3)$, $(p_{12}, p_{22}) = (2, 1)$, $(p_{13}, p_{23}) = (5, 1)$, and $(s_1, s_2) = (1, 2)$. The two-phase dynamic programming algorithm is demonstrated as follows.

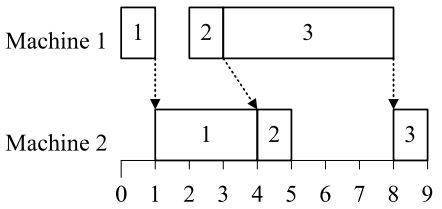

Fig. A.1. Optimal schedule with $g(1) = 18$.

Phase 1. (*Forward recursion for* $f(i, i_1, j, k)$)

Initialization:

$f(1, 1, 1, 1) = (1 - 1 + 1)(s_1 + p_{11} + s_2 + p_{21}) = 7$;
$f(1, 2, 2, 1) = (2 - 1 + 1)(s_1 + p_{1[1:2]} + s_2 + p_{2[1:2]}) = 20$;
$f(1, 3, 3, 1) = (3 - 1 + 1)(s_1 + p_{1[1:3]} + s_2 + p_{2[1:3]}) = 48$;
$f(2, 2, 2, 1) = (2 - 2 + 1)(s_1 + p_{12} + s_2 + p_{22}) = 6$;
$f(2, 3, 3, 1) = (3 - 2 + 1)(s_1 + p_{1[2:3]} + s_2 + p_{2[2:3]}) = 24$;
$f(3, 3, 3, 1) = (3 - 3 + 1)(s_1 + p_{13} + s_2 + p_{23}) = 9$;
For other values of i, i_1, j, k, we denote $f(i, i_1, j, k) = \infty$.

Recursion:

$(i, i_1, j, k) = (1, 1, 2, 2)$
$j' = 1$
$(2 - 1)s_1 + p_{12} = 3 < (2 - 1)s_2 + p_{21} = 5$
$z_1 = f(1, 1, 1, 1) + (2 - 1)(s_1 + p_{11} + 2s_2 + p_{2[1:2]}) = 7 + (1 + 1 + 4 + 4) = 17.$
$f(1, 1, 2, 2) = \min\{z_1\} = 17.$

$(i, i_1, j, k) = (1, 1, 3, 2)$
$j' = 1$
$(2 - 1)s_1 + p_{1[2:3]} = 8 > (2 - 1)s_2 + p_{21} = 5$
$z_1 = \infty.$
$j' = 2$
$(2 - 1)s_1 + p_{1[2:3]} = 8 > (2 - 1)s_2 + p_{2[1:2]} = 6$
$z_2 = \infty.$
$f(1, 1, 3, 2) = \min\{z_1, z_2\} = \infty.$

$(i, i_1, j, k) = (1, 1, 3, 3)$
$j' = 2$
$(3 - 1)s_1 + p_{1[2:3]} = 9 > (3 - 1)s_2 + p_{2[1:2]} = 8$
$z_1 = \infty.$
$f(1, 1, 3, 3) = \min\{z_1\} = \infty.$

$(i, i_1, j, k) = (1, 2, 3, 2)$
$j' = 2$
$(2 - 1)s_1 + p_{13} = 6 = (2 - 1)s_2 + p_{2[1:2]} = 6$
$z_1 = f(1, 2, 2, 1) + (3 - 2)(s_1 + p_{1[1:2]} + 2s_2 + p_{2[1:3]}) = 20 + (1 + 3 + 4 + 5) = 33.$
$f(1, 2, 3, 2) = \min\{z_1\} = 33.$

$(i, i_1, j, k) = (2, 2, 3, 2)$
$j' = 2$
$(2 - 1)s_1 + p_{13} = 6 > (2 - 1)s_2 + p_{22} = 3$
$z_1 = \infty.$
$f(2, 2, 3, 2) = \min\{z_1\} = \infty.$

Phase 2. (*Algorithm DP-Batch*)

Boundary condition: $p_{14} = \infty$; $p_{24} = 0$.
Initialization:
$g(4, 4) = 0$;
For other values of i, i_1, we denote $g(i, i_1) = \infty$.

Recursion:
$(i, i_1) = (3, 3)$
  $(i', i'_1, k) = (4, 4, 1)$
    $s_1 + p_{14} = \infty > s_2 + p_{23} = 3$
      $z_1 = g(4, 4) + 0 + f(3, 3, 3, 1) = 9.$
  $g(3, 3) = \min\{z_1\} = 9.$

$(i, i_1) = (2, 3)$
  $(i', i'_1, k) = (4, 4, 1)$
    $s_1 + p_{14} = \infty > s_2 + p_{2[2:3]} = 4$
      $z_1 = g(4, 4) + 0 + f(2, 3, 3, 1) = 24.$
  $g(2, 3) = \min\{z_1\} = 24.$

$(i, i_1) = (2, 2)$
  $(i', i'_1, k) = (4, 4, 2)$
    $2s_1 + p_{1[3:4]} = \infty > 2s_2 + p_{2[2:3]} = 6$
      $z_1 = g(4, 4) + 0 + f(2, 2, 3, 2) = \infty.$
  $(i', i'_1, k) = (3, 3, 1)$
    $s_1 + p_{13} = 6 > s_2 + p_{22} = 3$
      $z_2 = g(3, 3) + (3 - 3 + 1)(s_1 + p_{12}) + f(2, 2, 2, 1) = 9 + 3 + 6 = 18.$
  $g(2, 2) = \min\{z_1, z_2\} = 18.$

$(i, i_1) = (1, 3)$
  $(i', i'_1, k) = (4, 4, 1)$
    $s_1 + p_{14} = \infty > s_2 + p_{2[1:3]} = 7$
      $z_1 = g(4, 4) + 0 + f(1, 3, 3, 1) = 48.$
  $g(1, 3) = \min\{z_1\} = 48.$

$(i, i_1) = (1, 2)$
  $(i', i'_1, k) = (4, 4, 2)$
    $2s_1 + p_{1[3:4]} = \infty > 2s_2 + p_{2[1:3]} = 9$
      $z_1 = g(4, 4) + 0 + f(1, 2, 3, 2) = 33.$
  $(i', i'_1, k) = (3, 3, 1)$
    $s_1 + p_{13} = 6 = s_2 + p_{2[1:2]} = 6$
      $z_2 = \infty.$
  $g(1, 2) = \min\{z_1, z_2\} = 33.$

$(i, i_1) = (1, 1)$
  $(i', i'_1, k) = (4, 4, 3)$
    $3s_1 + p_{1[2:4]} = \infty > 3s_2 + p_{2[1:3]} = 11$
      $z_1 = g(4, 4) + 0 + f(1, 1, 3, 3) = \infty;$
  $(i', i'_1, k) = (4, 4, 2)$
    $2s_1 + p_{1[2:4]} = \infty > 2s_2 + p_{2[1:3]} = 9$
      $z_2 = g(4, 4) + 0 + f(1, 1, 3, 2) = \infty;$
  $(i', i'_1, k) = (3, 3, 2)$
    $2s_1 + p_{1[2:3]} = 9 > 2s_2 + p_{2[1:2]} = 8$
      $z_3 = g(3, 3) + (3 - 3 + 1)(2s_1 + p_{1[1:2]}) + f(1, 1, 2, 2) = 9 + 5 + 17 = 31;$
  $(i', i'_1, k) = (2, 3, 1)$
    $s_1 + p_{1[2:3]} = 8 > s_2 + p_{21} = 5$
      $z_4 = g(2, 3) + (3 - 2 + 1)(s_1 + p_{11}) + f(1, 1, 1, 1) = 24 + 4 + 7 = 35;$
  $(i', i'_1, k) = (2, 2, 1)$
    $s_1 + p_{12} = 3 < s_2 + p_{21} = 5$
      $z_5 = \infty;$
  $g(1, 1) = \min\{z_1, z_2, z_3, z_4, z_5\} = 31.$

Goal: $\min_{1 \le i_1 \le 3} g(1, i_1) = g(1, 1) = 31.$

The corresponding optimal schedule can be constructed by backtracking the recursion, as illustrated in Fig. B.1.

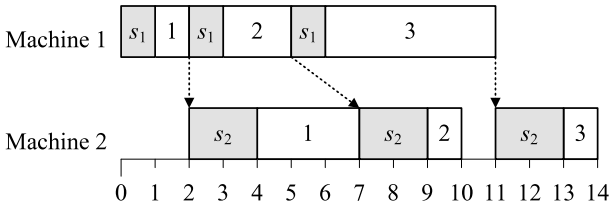

Fig. B.1. Optimal schedule with $g(1, 1) = 31$.

References

[1] J.J. Kanet, V. Sridharan, Scheduling with inserted idle time: problem taxonomy and literature review, Operations Research 48 (1) (2000) 99–110.
[2] C.N. Potts, M.Y. Kovalyov, Scheduling with batching: a review, European Journal of Operational Research 120 (2) (2000) 228–249.
[3] Y.M. Shafransky, V.A. Strusevich, The open shop scheduling problem with a given sequence of jobs on one machine, Naval Research Logistics 45 (7) (1998) 705–731.
[4] Y. Hendel, F. Sourd, An improved earliness-tardiness timing algorithm, Computers & Operations Research 34 (10) (2007) 2931–2938.
[5] M.R. Garey, R.E. Tarjan, G.T. Wilfong, One processor scheduling with symmetrical earliness and tardiness penalties, Mathematics of Operations Research 13 (2) (1988) 330–348.
[6] J.S. Davis, J.J. Kanet, Single-machine scheduling with early and tardy completion costs, Naval Research Logistics 40 (1) (1993) 85–101.
[7] W. Szwarc, S.K. Mukhopadhyay, Optimal timing schedules in earliness-tardiness single machine sequencing, Naval Research Logistics 42 (7) (1995) 1109–1114.
[8] F. Sourd, Optimal timing of a sequence of tasks with general completion costs, European Journal of Operational Research 165 (1) (2005) 82–96.
[9] Y. Pan, L. Shi, Dual constrained single machine sequencing to minimize total weighted completion time, IEEE Transactions on Automation Science and Engineering 2 (4) (2005) 344–357.
[10] E.C. Colina, R.C. Quinino, An algorithm for insertion of idle time in the single-machine scheduling problem with convex cost functions, Computers & Operations Research 32 (9) (2005) 2285–2296.
[11] J. Bauman, J. Józefowska, Minimizing the earliness-tardiness costs on a single machine, Computers & Operations Research 33 (11) (2006) 3219–3230.
[12] F. Della Croce, M. Trubian, Optimal idle time insertion in early-tardy parallel machines scheduling with precedence constraints, Production Planning & Control 13 (2) (2002) 133–142.
[13] B.M.T. Lin, T.C.E. Cheng, Two-machine flowshop scheduling with conditional deteriorating second operations, International Transactions in Operational Research 13 (2) (2006) 91–98.
[14] T.C.E. Cheng, Q. Ding, B.M.T. Lin, A concise survey of scheduling with time-dependent processing times, European Journal of Operational Research 152 (1) (2004) 1–13.
[15] T.C.E. Cheng, B.M.T. Lin, A. Toker, Makespan minimization in the two-machine flowshop batch scheduling problem, Naval Research Logistics 47 (2) (2000) 128–144.
[16] C.T. Ng, M.Y. Kovalyov, Batching and scheduling in a multi-machine flow shop, Journal of Scheduling 10 (6) (2007) 353–364.
[17] T.C.E. Cheng, G. Wang, Scheduling the fabrication and assembly of components in a two-machine flowshop, IIE Transactions 31 (2) (1999) 135–143.
[18] B.M.T. Lin, T.C.E. Cheng, A.S.C Chou, Scheduling in an assembly-type production chain with batch transfer, Omega 35 (2) (2007) 143–151.
[19] F.J. Hwang, B.M.T. Lin, Two-stage assembly-type flowshop batching problem subject to a fixed job sequence, Journal of the Operational Research Society (2011), in press (doi:10.1057/jors.2011.90).
[20] B.M.T. Lin, T.C.E. Cheng, Scheduling with centralized and decentralized batching policies in concurrent open shops, Naval Research Logistics 58 (1) (2011) 17–27.
[21] C.A. Glass, C.N. Potts, A.V. Strusevich, Scheduling batches with sequential job processing for two-machine flow and open shops, INFORMS Journal on Computing 13 (2) (2001) 120–137.
[22] M.Y. Kovalyov, C.N. Potts, V.A. Strusevich, Batching decisions for assembly production systems, European Journal of Operational Research 157 (3) (2004) 620–642.