Engineering Optimization
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/geno20

A branch-and-bound algorithm for makespan minimization in differentiation flow shops

Yen-Cheng Liu[a], Kuei-Tang Fang[a] & Bertrand Lin[a]
[a] Institute of Information ManagementNational Chiao Tung UniversityHsinchuTaiwanRepublic of China
Published online: 03 Dec 2012.

To cite this article: Yen-Cheng Liu, Kuei-Tang Fang & Bertrand Lin (2013) A branch-and-bound algorithm for makespan minimization in differentiation flow shops, Engineering Optimization, 45:12, 1397-1408, DOI: 10.1080/0305215X.2012.737783

To link to this article: http://dx.doi.org/10.1080/0305215X.2012.737783

Engineering Optimization, 2013
Vol. 45, No. 12, 1397–1408, http://dx.doi.org/10.1080/0305215X.2012.737783

A branch-and-bound algorithm for makespan minimization in differentiation flow shops

Yen-Cheng Liu, Kuei-Tang Fang and Bertrand M.T. Lin*

Institute of Information Management, National Chiao Tung University, Hsinchu, Taiwan, Republic of China

(*Received 8 April 2012; final version received 24 September 2012*)

This article considers a differentiation flow-shop model, where the jobs are divided into various categories, each of which consists of two stages of operations. All products should be processed first on the single common machine at stage 1. At the second stage, each individual product proceeds to a dedicated machine according to its type. The problem of makespan minimization under the setting with two product types is known to be strongly NP hard. This article considers an arbitrary number of job types by developing a lower bound and two dominance rules, based upon which branch-and-bound algorithms are designed. Computational experiments are carried out to examine the performance of the proposed properties. The statistics show that the proposed properties can substantially reduce the computing efforts required for finding optimal solutions.

Keywords: differentiation flow shop; makespan; branch-and-bound

1. Introduction

This article considers a scheduling problem in the context of a two-stage differentiation flow shop, which is a variant generalized from the traditional flow-shop scheduling, in which the jobs are to be processed on a set of machines that are arranged in a serial order and all jobs should flow through all of these machines in the specified machining route. Flow-shop settings have practical significance because many real-world applications can be formulated as flow shops. Since Johnson's seminal work (1954) on flow-shop scheduling, considerable research has been done on this topic. Many new models extended from the traditional flow shops have been proposed and studied in the literature (Dudek, Panwalkar, and Smith 1992; Gupta and Stafford 2006; Lee, Cheng, and Lin 1993; Linn and Zhang 1999; Rahimi-Vahed *et al*. 2008; Reisman, Kumar, and Motwani 1997; Tran and Ng 2012). Another direction concerning flow-shop research is scheduling with variable job processing times (Behnamian, Ghomi, and Zandieh 2011; Cheng, Ding, and Lin 2004; Gawiejnowicz 2008; Kononov and Gawiejnowicz 2001; Nowicki and Zdrzalka 1988).

This article addresses a scheduling problem in a differentiation flow shop, inspired by the concept of delayed customization, which is one of the major approaches adopted to achieve mass customization (Da Silveira, Borenstein, and Fogliatto 2001). In the differentiation setting, all of the jobs share a common critical machine at the primary stage, and then each individual product

*Corresponding author. Email: bmtlin@mail.nctu.edu.tw

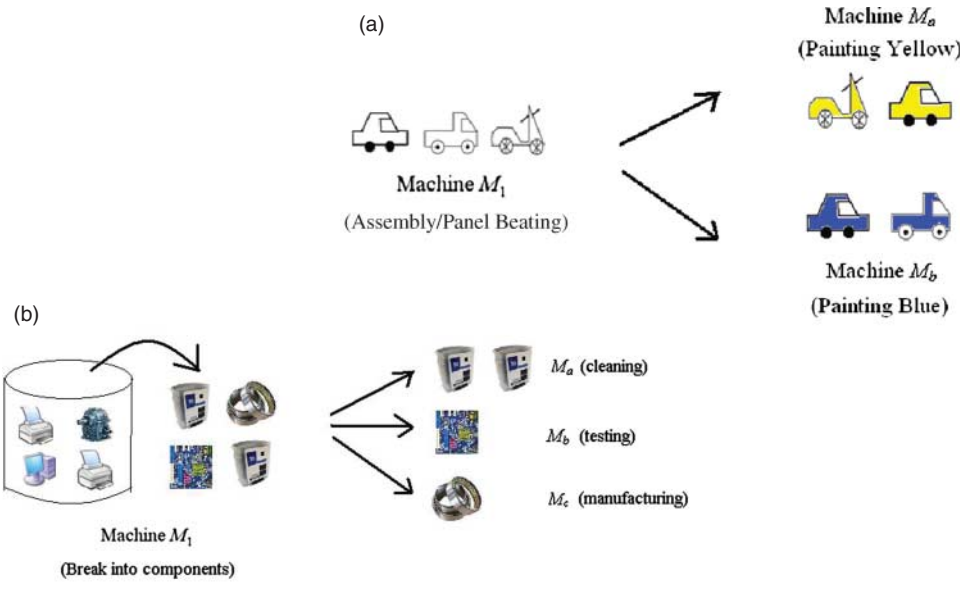

Figure 1. Applications of differentiation flow shops: (a) differentiation flow shop of a car manufacturing process; (b) differentiation flow shop of a recycling process.

proceeds to a dedicated machine at the successive stage. The differentiation flow shop arises from various practical applications. Figure 1(a) shows that in a car manufacturing plant or a garage, assembly or panel beating operations are carried out on a common stage 1 station (machine) no matter which kinds of cars are to be processed. After the processing at stage 1, the cars will flow toward a painting process at the second stage where several painting machines are installed for different colours. Therefore, the colour specification of a car determines the stage 2 machine to which this car should be routed and operated on. Another application shown in Figure 1(b) demonstrates a green supply chain. In the recycling plants, a recycled item is disassembled into several parts on the stage 1 station (machine), and then the parts flow forward to the second-stage stations (machines), depending on the types of the parts.

Differentiation flow shops have been studied in different contexts with different objective functions (Herrmann and Lee 1992; Cheng, Lin, and Tian 2009). This article discusses the minimization of makespan in a differentiation flow shop which has been proven to be NP hard, even if there are only two dedicated machines at the second stage (Herrmann and Lee 1992). This article develops a lower bound and two dominance rules to reduce the time branch-and-bound algorithms require for producing optimal solutions.

The article is organized as follows. Section 2 gives a formal definition of the studied problem and reviews related research works. Section 3 is dedicated to the development of two dominance rules and a lower bound for the design of branch-and-bound algorithms. In Section 4, a computational study is conducted to examine the performance of the proposed properties. Section 5 gives concluding remarks and suggests several potential research directions.

2. Problem statements and literature review

This section formally describes and defines the studied differentiation flow-shop problem. The notation used throughout this article will be introduced first. The flow shop is denoted by $F(1, m)$,

indicating a flow shop consisting of a single common machine at stage 1 and m independent dedicated machines at stage 2.

In problem $F(1, m)$, there are m disjoint sets of jobs $N_1 = \{J_1, \ldots, J_{n_1}\}$, $N_2 = \{J_{n_1+1}, \ldots, J_{n_1+n_2}\}, \ldots$, and $N_m = \{J_{n_1+n_2+\ldots+n_{m-1}+1}, J_{n_1+n_2+\ldots+n_{m-1}+2}, \ldots, J_{n_1+n_2+\ldots+n_{m-1}+n_m}\}$ to process. Each set stands for a specific type of jobs. Variables N_i and n_i denote the set of type i jobs and the number of jobs in N_i (*i.e.* $n_i = |N_i|$), respectively. All jobs must be processed in two stages. At the first stage, there is only one machine M_1 eligible for processing all jobs. Each job J_j, no matter which type it belongs to, must be first processed on this machine and takes a processing time $p_{j,1}$. Its completion time on this machine is denoted by $C_{j,1}$. At the second stage, there are m dedicated machines $M_{2,1}, M_{2,2}, \ldots, M_{2,m}$. If a job J_j belongs to type k, then it will be processed on the machine $M_{2,k}$ and requires a processing time $p_{j,2,k}$. Its completion time on the second machine is denoted by $C_{j,2,k}$. Jobs of a particular type are independent from those of another type at the second stage. But all jobs compete for the processing resource at the first stage. As it can be established by a job-interchange argument that there exist optimal permutation schedules, *i.e.* all machines have the same job processing sequence, this article will consider only permutation schedules. The objective of this article is to find a permutation schedule whose makespan ($C_{\max}$), or the maximum completion time, is minimized under the assumption that no pre-emption is allowed on any machine.

2.1. *Notation*

$N_1 = \{J_1, J_2, \ldots, J_{n_1}\}$: type 1 jobs
$N_2 = \{J_{n_1+1}, J_{n_1+2}, \ldots, J_{n_1+n_2}\}$: type 2 jobs
$\vdots$
$N_1 = \{J_{n_1+n_2+\ldots+n_{m-1}+1}, J_{n_1+n_2+\ldots+n_{m-1}+2}, \ldots, J_{n_1+n_2+\ldots+n_{m-1}+n_m}\}$: type m jobs
$N = N_1 \cup N_2 \cup \ldots \cup N_m$: the set of all jobs
$n = |N| = n_1 + n_2 + \ldots + n_m$
M_1: the stage 1 machine for processing all types of jobs
$M_{2,k}$: the stage 2 machine dedicated to the type k jobs
$P_{j,1}$: the processing time of job J_j on the stage 1 machine M_1
$P_{j,2,k}$: the processing time of type k job J_j on the stage 2 machine $M_{2,k}$
$C_{j,1}$: the completion time of job J_j on the stage 1 machine M_1
$C_{j,2,k}$: the completion time of type k job J_j on the stage 2 machine $M_{2,k}$
S: a particular processing sequence of all jobs on machine M_1.

2.2. *Numerical example*

Consider a set of five jobs $\{J_1, J_2, J_3, J_4, J_5\}$ in which jobs J_1 and J_2 belong to type 1, and jobs J_3, J_4 and J_5 belong to type 2. The processing times of the jobs are shown in Table 1. Given a processing sequence $S = (J_1, J_3, J_5, J_2, J_4)$, the Gantt chart is depicted in Figure 2.

Table 1. Example of five jobs in two types.

Jobs	J_1	J_2	J_3	J_4	J_5
Type	1	1	2	2	2
$p_{i,1}$	4	6	2	7	8
$p_{i,2,k}$	3	2	10	5	2

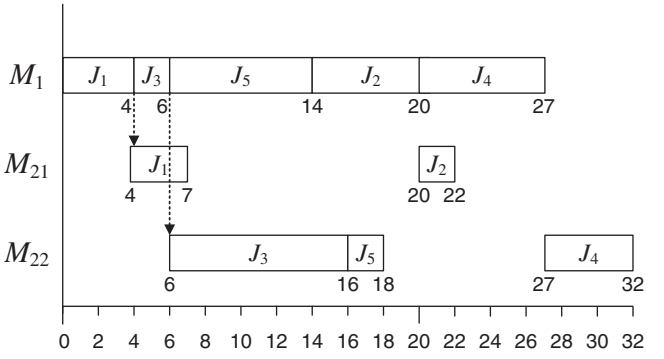

Figure 2. Gantt chart of the example schedule.

The first study on the setting of differentiation flow shops could be attributed to Herrmann and Lee (1992). They proved the $F(1, 2)$ problem of makespan minimization to be strongly NP hard. A polynomial time algorithm was developed to solve the problem subject to the assumption that the processing sequence of each job type is known *a priori*. The algorithm is based upon a transformation to the single-machine scheduling problem of minimizing the maximum lateness, which can be solved by the earliest due date (EDD) rule (Jackson 1955). They also designed a branch-and-bound algorithm to solve the problem to optimality. Instances containing 15 jobs were solved within 1 second. An independent study by Kyparisis and Koulamas (2000) proposed a polynomial-time algorithm for the $F(1, m)$ problem subject to the assumption that jobs of the same type should be processed consecutively on the stage 1 machine. They developed an $O(m(\log n + \log m))$ algorithm, where m and n are the number of machines and the number of jobs, respectively. Mosheiov and Yovel (2004) improved the complexity to $O(n \log n)$ if $m \leq n$. Cheng and Kovalyov (1998) considered the problem where whenever the processing of the jobs transfers from one type to another a set-up time on the stage 1 machine a setup time is incurred. They developed a dynamic programming algorithm for this problem. Considering the $F(1, 2)$ model with the objective function of the weighted sum of stage 2 machine completion times, Cheng, Lin, and Tian (2009) gave a strong NP-hardness proof and proposed a heuristic algorithm and analysed its performance ratio, the ratio between the approximation solution value and the optimal one, to be $4/3$. Lin and Hwang (2011) developed a polynomial-time dynamic programming algorithm to minimize the total completion time, subject to the condition that the processing sequence of each job type is known in advance.

In this article, the case with an arbitrary number m of job types is addressed. As mentioned above, Herrmann and Lee (1992) developed a branch-and-bound algorithm for $F(1, 2)$ to minimize the makespan. This study will extend the manufacturing setting to an arbitrary number of dedicated machines launched at stage 2. In addition, the branch-and-bound algorithm proposed in this article can solve instances of more jobs.

3. Branch-and-bound algorithm

Once a problem is proven to be NP hard, it is very unlikely that a polynomial algorithm could be designed for finding optimal solutions to this problem. Branch-and-bound is a solution approach that implicitly enumerates all of the feasible solutions by constructing and exploring a tree structure of the solution space. This section proposes a branch-and-bound algorithm by developing two dominance rules and a lower bound to reduce the search efforts required for probing the solution space of the hard $F(1, m)$ problem.

Branch-and-bound methods use a tree structure to enumerate all of the possible solutions. The depth-first search (DFS) approach is adopted in this article to construct the enumeration tree. DFS facilitates easy implementations in a recursive way and requires less memory space than a breadth-first search. The following two dominance properties are first developed to eliminate unnecessary branching.

Consider two jobs J_i and J_j of the same type that are processed consecutively in some optimal schedule. Assume that J_i precedes J_j on the stage 1 machine in the optimal schedule but J_j precedes J_i in Johnson's rule. Swap the positions of jobs J_i and J_j. The processing of any job of other types on any machine is not altered. Since J_j precedes J_i in Johnson's rule, swapping their positions will not increase the processing span on machine $M_{2,k}$. Consider type 2 jobs J_5 and J_3 in the schedule $(J_1, J_5, J_3, J_2, J_4)$ of the numerical instance of Figure 2 as an example. Swapping their positions results in the schedule $(J_1, J_3, J_5, J_2, J_4)$, as shown in Figure 2. The starting times of all operations of type 1 jobs remain unchanged, and the processing span on machine $M_{2,2}$ does not increase. The above arguments lead to the first property.

LEMMA 1 *For any job type N_k ($1 \leq k \leq m$), if jobs J_i and $J_j \in N_k$ are scheduled consecutive and J_i precedes J_j in accordance with Johnson's rule, then there is an optimal schedule in which J_i precedes J_j.*

The following property does not require the two jobs under consideration to be consecutive to each other. This relaxation will increase the pruning power of dominance properties.

LEMMA 2 *For any two jobs J_i and $J_j \in N_k$ ($1 \leq k \leq m$), if $p_{i,1} \leq p_{j,1}$ and $p_{j,2,k} \leq p_{i,2,k}$, then there is an optimal solution in which job J_i precedes job J_j.*

Proof Assume that job J_j precedes job J_i in some optimal schedule. Perform operation-based interchanges. The machine 1 operations of the two jobs are swapped, while keeping their stage 2 operations unaltered. It is clear that the completion times of any other jobs will not increase. Then, swap the stage 2 operations of the two jobs in the derived schedule. Similarly, the completion times of all jobs will not increase. ■

Lower bounds or upper bounds on solution values are critical to the efficiency of branch-and-bound algorithms. As the $F(1, m)$ problem is for minimization, a lower bound of each tree node corresponding to a partial schedule will be computed. The lower bound consists of the cost already incurred by the assigned jobs and an underestimate of the cost that will be incurred by the remaining unscheduled jobs. If the derived lower bound value at a node is larger than or equal to the best (incumbent) solution value found thus far, then the enumeration process can skip the subtree rooted at this node without sacrificing the optimal solutions. It also means that some nodes and their subtrees will be pruned off and the optimality of the solution is still guaranteed, if the elapsed running time allows. For this reason, the following discussion is dedicated to the development of a lower bound through several properties of an instance transformation.

The development of the lower bound is adapted from the concept of data rearrangement proposed in Lin and Wu (2005) and the polynomial-time algorithm of Herrmann and Lee (1992). The data rearrangement technique of Lin and Wu (2005), proposed for solving two-machine flow-shop scheduling of total completion time minimization, is applied to each type of job for constructing a special instance that admits a linear ordering of jobs an optimal sequence. The m sequences are then merged into a single sequence of all jobs by generalizing the polynomial-time algorithm of Herrmann and Lee (1992) for m=2. The objective value of the derived sequence is guaranteed to be a lower bound on the optimal value of the original problem.

The discussion will be illustrated through a numerical example, followed by formal proofs. First, the transformation disaggregates the processing times of each job and then rearranges the

Table 2. (a) Original and (b) derived type 1 jobs.

Type 1 jobs	J_1	J_2	J_3	J_4	J_5
(a) Original					
$p_{i,1}$	4	7	2	6	8
$p_{i,2,1}$	3	5	10	2	2
(b) Derived					
$p_{(i),1}$	2	4	6	7	8
$p_{(i),2,1}$	10	5	3	2	2

job processing times in order to create a new instance for which the optimal solution value is not greater than the original optimal solution value. The first step of the transformation procedure is to focus solely on each job type, collect all the stage 1 processing times of that type of jobs, and then sort them in non-decreasing order. After that, these processing times are used as the stage 1 processing times of the jobs of this type. Similarly, all the stage 2 processing times in each job type are collected and sorted in non-increasing order. The processing times are then reassigned to match the stage 1 processing times. After the above steps, the optimal makespan solution of this new instance is smaller than or equal to the original one and the optimal sequence of this new instance can be easily constructed from the sequencing order in each type. An instance and a derived instance are shown in Table 2. Among the new jobs, the first job has the shortest stage 1 processing time and the longest stage 2 processing time, while the last job has the longest stage 1 processing time and the shortest stage 2 processing time. In general, for any two jobs, say J_i and J_j, of the same type k, $p_{i,1} \leq p_{j,1}$ if and only if $p_{i,2,k} \geq p_{j,2,k}$.

A formal proof of the rearrangement process and the variables used in the proof are given below.

$p_{(1),1}$ denotes the smallest value in the stage 1 processing times among type 1 jobs, *i.e.* $p_{(1),1} = \min\{p_{1,1}, p_{2,1}, \ldots, p_{n_1,1}\}$; $p_{(n_1,+1),1}$ denotes the smallest value in the stage 1 processing times among type 2 jobs, . . ., and $p_{(n_1+n_2+\ldots+n_{m-1}+1),1}$ denotes the smallest value in the stage 1 processing times among type m jobs.

$p_{(2),1}, p_{(n_1+2),1}, p_{(n_1+n_2+2),1}, \ldots, p_{(n_1+n_2+\ldots+n_{m-1}+2),1}$ are the second smallest values in the stage 2 processing times of type 1, type 2, . . ., type m jobs, respectively.

$\vdots$

$p_{(n_1),1}, p_{(n_1+n_2),1}, p_{(n_1+n_2+n_3),1}, \ldots, p_{(n_1+n_2+\ldots+n_m),1}$ are the largest values in the stage 1 processing times of type 1, type 2, . . ., type m jobs, respectively.

Therefore, the following inequalities follow:

$$p_{(1),1} \leq p_{(2),1} \leq \cdots \leq p_{(n_1),1}$$
$$p_{(n_1+1),1} \leq p_{(n_1+2),1} \leq \cdots \leq p_{(n_1+n_2),1}$$
$$\vdots$$
$$p_{(n_1+n_2+\ldots+n_{m-1}+1),1} \leq p_{(n_1+n_2+\ldots+n_{m-1}+2),1} \leq \cdots \leq p_{(n_1+n_2+\ldots+n_m),1}$$

Similarly,

$p_{(1),2,1}, p_{(n_1+1),2,2}, p_{(n_1+n_2+1),2,3}, \ldots, p_{(n_1+n_2+\ldots+n_{m-1}+1),2,m}$ are the smallest values in the stage 2 processing times of type 1, type 2, . . ., type m jobs, respectively.

$p_{(2),2,1}, p_{(n_1+1),2,2}, p_{(n_1+n_2+1),2,3}, \ldots, p_{(n_1+n_2+\ldots+n_{m-1}+2),2,m}$ are the second smallest values in the stage 2 processing times of jobs type 1, type 2, . . ., type m jobs, respectively.

$\vdots$

$p_{(n_1),2,1}, p_{(n_1+n_2),2,2}, p_{(n_1+n_2+n_3),2,3}, \ldots, p_{(n_1+n_2+\ldots+n_m),2,m}$ are the largest values in the stage 2 processing times of type 1, type 2, ..., type m jobs, respectively.

The arrangement leads to the following inequalities:

$$p_{(1),2,1} \leq p_{(2),2,1} \leq \cdots \leq p_{(n_1),2,1}$$
$$p_{(n_1+1),2,2} \leq p_{(n_1+2),2,2} \leq \cdots \leq p_{(n_1+n_2),2,2}$$
$$\vdots$$
$$p_{(n_1+n_2+\ldots+n_{m-1}+2),2,m} \leq p_{(n_1+n_2+\ldots+n_{m-1}+2),2,m} \leq \cdots \leq p_{(n_1+n_2+\ldots+n_m),2,m}$$

With the above parameter values, new jobs of each type are then defined. Set N'_k contains the type k jobs derived after the operation rearrangement such that for type k jobs J'_i and J'_j if $i < j$ then the stage 1 (respectively, stage 2) processing time of job J'_i is smaller than (respectively, larger than) or equal to that of job J'_j. For example, the job indexed ith in $N'_1 = \{J'_1, J'_2, \ldots, J'_{n_1}\}$ has processing times $p_{(i),1}$ and $p_{(n_1-i+1),2,1}$. The whole instance is the union of all derived subsets, $N' = N'_1 \cup N'_2 \cup \ldots \cup N'_m$.

LEMMA 3 *There is an optimal schedule of the instance $N_1 \cup N_2 \cup \ldots \cup N'_k \cup \ldots \cup N_m$ in which the jobs of N'_k are sequenced in the order $(J'_{n_1+n_2+\ldots+n_{k-1}+1}, J'_{n_1+n_2+\ldots+n_{k-1}+2}, \ldots, J'_{n_1+n_2+\ldots+n_k})$.*

Proof For any two adjacent jobs of N'_k such that the former has a shorter stage 1 processing time and a longer stage 2 processing time, swap their positions. According to Lemma 2, this will not increase the makespan and thus the job sequence of N'_k will be part of an optimal schedule. ■

LEMMA 4 *The optimal solution value of the instance $N_1 \cup N_2 \cup \ldots \cup N'_k \cup \ldots \cup N_m$ is smaller than or equal to that of the original instance $N_1 \cup N_2 \cup \ldots \cup N_k \cup \ldots \cup N_m$, i.e. $Z^*(N_1 \cup N_2 \cup \ldots \cup N'_k \cup \ldots \cup N_m) \leq Z^*(N)$, where $Z^*(.)$ is a function giving the optimal solution value of the given job set.*

Proof Consider that if there is an optimal solution of the original instance $N_1 \cup N_2 \cup \ldots \cup N_k \cup \ldots \cup N_m$ in which the type k job J_i with a larger stage 1 machine processing time $p_{i,1}$ precedes the job J_j with a shorter stage 1 machine processing time $p_{j,1}$. Interchange the stage 1 operations of the two jobs, (*i.e.* job J'_i with the stage 1 processing time $p_{j,1}$ and job J'_j with the stage 1 processing time $p_{i,1}$); the makespan will not increase. Please refer to Figure 3.

From the above analysis, the same operation-interchange argument is applied to all jobs of type k: $J_{n_1+n_2+\ldots+n_{k-1}+1}, J_{n_1+n_2+\ldots+n_{k-1}+2}, \ldots, J_{n_1+n_2+\ldots+n_k}$. This leads to a job sequence $J'_{n_1+n_2+\ldots+n_{k-1}+1}, J'_{n_1+n_2+\ldots+n_{k-1}+2}, \ldots, J'_{n_1+n_2+\ldots+n_k}$ with stage 1 machine processing times $p_{(n_1+n_2+\ldots+n_{k-1}+1),1}, p_{(n_1+n_2+\ldots+n_{k-1}+2),1}, \ldots, p_{(n_1+n_2+\ldots+n_k),1}$, derived from which the optimal solution will not be greater than the optimal solution of the original instance. Therefore, if there are any two adjacent jobs J'_i and J'_j such that job J'_i with a shorter stage 2 processing time $p_{i,2,1}$ precedes job

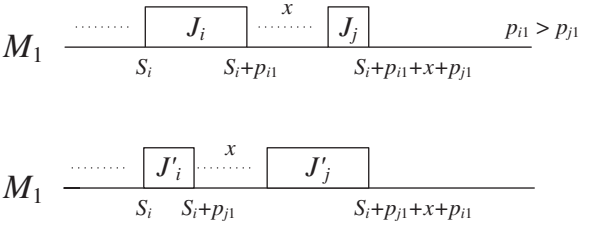

Figure 3. Operation interchange.

J'_j with a longer stage 2 processing time $p_{j,2,1}$, then interchange the stage 2 operations of the two jobs to equip job J''_i with the stage 2 processing time $p_{j,2,1}$ and job J''_j with the stage 2 processing time $p_{i,2,1}$. The makespan will not increase, either.

Repeating the same technique to the jobs $J'_{n_1+n_2+\ldots+n_{k-1}+1}, J'_{n_1+n_2+\ldots+n_{k-1}+2}, \ldots, J'_{n_1+n_2+\ldots+n_k}$ the sub-sequence $J''_{n_1+n_2+\ldots+n_{k+1}}, J''_{n_1+n_2+\ldots+n_{k-1}+2}, \ldots, J''_{n_1+n_2+\ldots+n_k}$ with stage 1 processing times $p_{(n_1+n_2+\ldots+n_{k-1}+1),1}, p_{(n_1+n_2+\ldots+n_{k-1}+2),1}, \ldots, p_{(n_1+n_2+\ldots+n_k),1}$ and stage 2 processing times $p_{(n_1+n_2+\ldots+n_{k-1}+1),2,k}, p_{(n_1+n_2+\ldots+n_{k-1}+2),2,k}, \ldots, p_{(n_1+n_2+\ldots+n_k),2,k}$ will emerge. These jobs constitute the sets N'_k whose optimal solution will not be larger than the optimal solution value of the original instance, either. Consequently, inequality $Z^*(N_1 \cup N_2 \cup \ldots \cup N'_k \cup \ldots \cup N_m) \leq Z^*(N)$ holds, and the lemma follows. ■

THEOREM 1 *In F(1, m) of makespan minimization, the optimal solution value of the instance* $N' = N'_1 \cup N'_2 \cup \ldots \cup N'_m$ *is a lower bound on the optimal solution value of the instance* $N = N_1 \cup N_2 \cup \ldots \cup N_m$.

Proof According to Lemma 2 and Lemma 4, if only the type k jobs are considered and the operation interchanges are performed, when necessary, as in Lemma 2, the optimal solution of the new instance will not be greater than that of the original instance, *i.e.* $Z^*(N_1 \cup N_2 \cup \ldots \cup N'_k \cup \ldots \cup N_m) \leq Z^*(N)$. Each type of job is iteratively handled by the operation-interchange arrangements as in Lemma 2. Then, the new instance is exactly $N' = N'_1 \cup N'_2 \cup \ldots \cup N'_m$, and the optimal solution for which will not be greater than that of the original instance. ■

While Theorem 1 indicates the relation between the optimal solution values of the original instance and the derived instance, the issue concerning how to calculate the optimal solution value $Z^*(N_1 \cup N_2 \cup \ldots \cup N'_k \cup \ldots \cup N_m)$ needs to be addressed. Given the new instance, there is an optimal solution in which the job processing sequence for each type of jobs abides by the increasing order of job indices. Therefore, the question concerning deriving $Z^*(N_1 \cup N_2 \cup \ldots \cup N'_k \cup \ldots \cup N_m)$ reduces to determining an optimal interleaving sequence on the stage 1 machine from the m sequences, one for each type of job. By a reduction to the single-machine maximum lateness problem, Herrmann and Lee (1992) developed an $O(n \log n)$ algorithm for optimally interleaving two sequences of two types of job. The solution approach can be generalized to deal with m types of job as follows.

Given m sequences $\sigma_1, \sigma_2, \ldots, \sigma_m$ for the m types of job, for each J_i of type k, define $t_{(i),k} = p_{(i),2,k} + \sum_{J_u \in A_i} p_{(u),2,k}$, where A_i consists of job J_i and its successor jobs in sequence σ_k. An optimal interleaving algorithm is to sequence all jobs in non-increasing order of the derived $t_{(i),k}$ values. Table 3 shows an example with two types of job such that the jobs of each type are sequenced by their indices.

With the $t_{(i),k}$ values of all jobs, deploying the optimal interleaving algorithm yields the optimal schedule $(J_6, J_1, J_7, J_2, J_3, J_8, J_4, J_9, J_5, J_{10})$ in which all jobs are ordered in non-increasing order

Table 3. Example of optimal interleaving.

Type 1 jobs	J_1	J_2	J_3	J_4	J_5
$p_{(i),1}$	2	4	6	7	8
$p_{(i),2,1}$	10	5	3	2	2
$t_{(i),1}$	22	12	7	4	2
Type 2 jobs	J_6	J_7	J_8	J_9	J_{10}
$p_{(i),2}$	3	4	5	6	7
$p_{(i),2,2}$	12	8	2	2	1
$t_{(i),2}$	25	13	5	3	1

of the $t_{(i),k}$ values. It can be seen that the derived schedule will not violate the given sub-sequences $(J_1, J_2, J_3, J_4, J_5)$ and $(J_6, J_7, J_8, J_9, J_{10})$ of type 1 and type 2 jobs. Using this interleaved sequence, the minimum makespan can be obtained and it serves as a lower bound for the original instance.

Before closing the discussion on lower bounds, the computing time required for each node needs to be analysed. The computing time required for deriving lower bounds is crucial to the efficiency of branch-and-bound algorithms,as this computation is invoked for each node in the enumeration tree. Deriving set N' from set N requires $O(n_k \log n_k)$ time for sorting the machine 1 operations and machine 2 operations of type k jobs. The required time can be reduced to $O(n_k)$ because the sorting process can be done only once by a preprocessing procedure when the whole instance is given. That is, it is not necessary to invoke the sorting process at each tree node. Therefore, set N' can be obtained in $O(n_k) = O(n)$ time. Given the m sequences, the optimal interleaved sequence can be obtained in $O(n \log n)$ time, which is required by sorting the t values. Therefore, computing the lower bound of each tree node takes $O(n \log n)$ time.

4. Computational study

This section presents the computational experiment conducted to study the performance of the proposed algorithm. The codes were written in C++ (Visual Studio 2008) and the platform for the study is a personal computer with a P5 2.8 MHz CPU and 2 GB RAM.

In the experimental settings, the two models $F(1, 3)$ and $F(1, 5)$ were considered to examine the efficiency of the proposed algorithm. Two modes of the proposed algorithm are tested. The first one, denoted by LB, deploys the lower bound introduced in the previous section. The second one, LB+DR, incorporates both the lower bound and the two dominance rules.

In the experiment, all job processing times were generated at random from the uniform interval [1, 100]. The limit of execution time for each instance was set as 1800 seconds. If the algorithm could not finish the exploration of a given instance within the time limit, the algorithm would abort and report a failure for this instance. For each scenario, 20 independent instances were generated and then solved by the branch-and-bound algorithm. For each 20 instances, the statistics of interest include the maximum execution time, the minimum execution time, the average execution time, the maximum number of visited nodes, the minimum number of visited nodes and the average number of visited nodes. The average statistics excludes the results associated with the outlier instances that were not successfully solved within the time limit.

The computational results concerning the $F(1, 3)$ model are summarized in Table 4. The right-most column entitled #Solved contains the number of instances successfully solved within the given time limit of 1800 seconds. Preliminary experiments indicated that a depth-first-search strategy without a lower bound and dominance rules can solve instances of up to 13 or 14 jobs. It is interesting to note the effects introduced by the proposed lower bound. The average elapsed running time of the branch-and-bound algorithm for coping with instances with 15 jobs is less than 1 second. Only two of the tested instances were not successfully solved within the time limit. The statistics indicate that the lower bound can actually enhance efficiency. As for the average number of nodes visited in solving the instances with nine jobs, the algorithm equipped with the lower bound visits only 8521 nodes on average. Even the maximum number of nodes visited in all sessions is only 58,205. The results show the power of the lower bound in pruning off the non-promising nodes of the enumeration tree.

The next part is to examine the effects brought forth by the dominance rules. In Table 4, almost all of the average execution times under the column LB+DR are negligible. In the case of $3 \times 20 = 60$ jobs and $3 \times 25 = 75$ jobs, the maximum time, average time, maximum number of nodes and average number of nodes in the LB+DR mode are much smaller than those in the mode with the lower bound only. One instance required an exceedingly long running time such that the

Table 4. Computational results of the $F(1, 3)$ setting.

		Time			Node			
$m \times n$		Max.	Min.	Avg.	Max.	Min.	Avg.	# Solved
$3 \times 3 = 9$	LB	0.08	0.00	0.01	58,205	45	8,521.50	20
	LB+DR	0.03	0.00	0.01	27,615	45	4,169.50	20
$3 \times 5 = 15$	LB	1.28	0.00	0.13	735,484	15	74,302.33	18
	LB+DR	0.27	0.00	0.03	149,744	15	14,660.72	18
$3 \times 10 = 30$	LB	1.91	0.00	0.11	652,202	30	38,507.94	18
	LB+DR	0.02	0.00	0.00	2,449	30	755.89	18
$3 \times 15 = 45$	LB	0.05	0.00	0.01	8,641	45	2,012.53	19
	LB+DR	0.05	0.00	0.01	8,523	45	1,966.26	19
$3 \times 20 = 60$	LB	0.14	0.00	0.04	15,509	60	3,493.28	18
	LB+DR	9.27	0.00	0.52	1,442,403	60	79,211.84	19
		(0.13)	(0.00)	(0.04)	(15,387)	(60)	(3,479.00)	
$3 \times 25 = 75$	LB	3.47	0.00	0.27	534,551	75	35,875.00	17
	LB+DR	558.80	0.00	28.88	66,889,828	75	3,463,878.45	20
		(0.44)	(0.00)	(0.09)	(36,705)	(75)	(6,579.82)	

Note: LB = lower bound; DR = dominance rules.

Table 5. Computational results of the $F(1, 5)$ setting.

		Time			Node			
$m \times n$		Max.	Min.	Avg.	Max.	Min.	Avg.	# Solved
	LB	0.00	0.00	0.00	47	5	24.40	
$5 \times 1 = 5$	LB+DR	0.00	0.00	0.00	47	5	24.40	20
	LB	0.02	0.00	0.00	5,463	10	398.50	20
$5 \times 2 = 10$	LB+DR	0.00	0.00	0.00	4,241	10	312.55	20
	LB	0.59	0.00	0.03	298,900	15	15,887.21	20
$5 \times 3 = 15$	LB+DR	0.30	0.00	0.02	142,348	15	7,640.42	19
	LB	2.39	0.00	0.23	474,608	50	45,204.45	19
$5 \times 10 = 50$	LB+DR	0.05	0.00	0.01	5,447	50	1,893.70	20
	LB	0.27	0.00	0.14	10,550	100	5,270.21	20
$5 \times 20 = 100$	LB+DR	2.97	0.00	0.28	261,714	100	18,075.40	19
		(0.27)	(0.00)	(0.13)	(10,310)	(100)	(5,252.32)	20

Note: LB = lower bound; DR = dominance rules.

algorithm with the lower bound failed to solve it. When the dominance rules were incorporated, the running time was still long but fell within 30 minutes. If these specific instances are excluded from the LB+DR part, then the improvement attributed to the incorporation of dominance rules is evident. Take the case of $3 \times 25 = 75$ jobs as an example. The dominance rules provide synergy effects with the lower bound. There were originally three outliers to the LB mode. When the dominance rules were deployed, all instances were successfully solved and the average time reduced to only 28 seconds. With regard to the maximum required running time, the LB mode took more than 30 minutes to solve the worst case instance, but the LB+DR mode took only 558 seconds (about 9 minutes) to obtain the optimal solution. The entries enclosed with parenthesis, for example the last line in the row of $3 \times 20 = 60$, are the statistics for the instances solved by both LB and LB+DR. These entries can further highlight the improvements made through the deployment of dominance rules. The results of the $F(1, 5)$ setting are shown in Table 5. Similarly, the algorithm equipped with the lower bound successfully solved most instances within seconds. In the case with m=5, the number of jobs on each machine increased to 20, such that the total number of jobs was 100. There were no outliers not solved by the LB mode. This instance was solved when the dominance rule was incorporated.

Table 6. Computational results using LB+DR mode in the $F(1, 5)$ setting.

$m \times n$	Avg. time	Avg. node	# Solved
$5 \times 20 = 100$	0.28	18,075.40	20
$5 \times 100 = 500$	53.44	187,400.05	19
$5 \times 120 = 600$	138.66	522,227.40	20
$5 \times 160 = 800$	322.17	611,292.35	20
$5 \times 200 = 1000$	432.96	357,108.39	18
$5 \times 220 = 1100$	622.63	352,353.13	15
$5 \times 240 = 1200$	5.38	1,200.00	9

Note: LB = lower bound; DR = dominance rules.

The last part of the experiments was set to increase the number of jobs in the $F(1, 5)$ setting in order to examine the scalability of the branch-and-bound LB+DR algorithm. Table 6 summarizes the average elapsed execution time and the average number of visited nodes for each $n \times m$ combination. The LB+DR algorithm solved all test instances with 800 or fewer jobs in 5 minutes. When the total number of jobs increased to 1100, 15 of the 20 instances were successfully solved in 6 minutes, and the optimal solutions of five instances were not reported in the time limit of 30 minutes. The execution for most of the instances with 1200 jobs aborted with failure, but nine instances were successfully solved in a few seconds.

As mentioned, the previous algorithm for $F(1, 2)$ can solve instances up to 15 jobs. The numerical statistics conveyed through the computational study clearly showed the curtailing efficiency of the proposed lower bound and the two dominance rules. With regard to real applications, a scale of 1000 jobs is large. Therefore, the LB+DR algorithm is also of practical significance.

5. Conclusions

This study has considered the $F(1, m)$ problem, which is an extension of the traditional flowshop scheduling problem. The significance of the $F(1, m)$ model lies in the potential real-world applications in delayed differentiation as well as the theoretical challenges in characterizing the solution structures. In this model, stage 1 has a common machine shared by all types of job, and stage 2 consists of different types of dedicated machines. All of the jobs are processed on the stage 1 machine and then proceed to specific stage 2 dedicated machines according to their product types. The objective function considered in this article is makespan minimization. This problem has been proven to be NP hard in the literature. This article developed a branch-and-bound algorithm equipped with a lower bound and two dominance rules to obtain optimal solutions in a more efficient way by avoiding non-promising job permutations. Computational experiments showed the performance of the proposed lower bound and dominance rules in curtailing unnecessary branching.

For future research, it could be interesting and challenging to propose approximation algorithms and analyse their performance ratios. Another potential topic is to tackle the same scheduling problem with the objective function of the total completion time, *i.e.*, $F(1, m) \| \sum C_i$ which can be solved by polynomial time dynamic programming algorithms under the assumption that the sequence of each type of jobs is known *a priori* (Lin and Hwang 2011). The dynamic programing algorithms are, however, impractical to be embedded in branch-and-bound algorithms. On the other hand, the classical $F2| \sum C_i$ problem is strongly NP hard. The development of exact and heuristic algorithms for this problem for analysis on the solution structures will be necessary.

Acknowledgement

This research was partially supported by the National Science Council of Taiwan under grant NSC-NSC 97-2923-H-009-001-MY3.

References

Behnamian, J., S. M. T. F. Ghomi, and M. Zandieh. 2011. "Hybrid solving algorithm for complex machine scheduling problem." *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, 6–9 December, Singapore. New York: IEEE Press, 794–798.

Cheng, T. C. E., and M. Y. Kovalyov. 1998. "An Exact Algorithm for Batching and Scheduling Two Part Types in a Mixed Shop: A Technical Note." *International Journal of Production Economics*, 55, 53–56.

Cheng, T. C. E., Q. Ding, and B. M. T. Lin. 2004. "A Concise Survey of Scheduling with Time-Dependent Processing Times." *European Journal of Operational Research*, 152, 1–13.

Cheng, T. C. E., B. M. T. Lin, and Y. Tian. 2009. "Scheduling of a Two-Stage Differentiation Flow Shop to Minimize Weighted Sum Machine Completion Times." *Computers & Operations Research*, 36, 3031–3040.

Da Silveira, G., D. Borenstein, and F. S. Fogliatto. 2001. "Mass Customization: Literature Review and Research Directions." *International Journal of Production Economics*, 72, 1–13.

Dudek R. A., S. S. Panwalkar, and M. L. Smith. 1992. "The Lessons of Flow Shop Scheduling Research." *Operations Research*, 40, 7–13.

Gawiejnowicz, S. 2008. *Time-Dependent Scheduling*. Berlin: Springer.

Gupta, J. N. D., and E. F. Stafford. 2006. "Flow Shop Scheduling Research after Five decades." *European Journal of Operational Research*, 169, 699–711.

Herrmann, J. W., and C. Y. Lee. 1992. *Three-Machine Look-Ahead Scheduling Problems*. Research Report No. 92–93, Department of Industrial Engineering, University of Florida.

Jackson, J. R. 1955. *Scheduling a Production Line to Minimize Maximum Lateness*. Research Report 43, Management Science Research Report, University of California.

Johnson, S. M. 1954. "Optimal Two- and Three-Stage Production Schedules with Setup Times Included." *Naval Research Logistics Quarterly*,1, 61–67.

Kononov, A., and S. Gawiejnowicz. 2001. "NP-Hard Cases in Scheduling Deteriorating Jobs on Dedicated Machines." *Journal of the Operational Research Society*, 52, 708–717.

Kyparisis, G. J., and C. Koulamas. 2000. "Flow Shopand Open Shop Scheduling with a Critical Machine and Two Operations per Job." *European Journal of Operational Research*, 127, 120–125.

Lee, C. Y., T. C. E. Cheng, and B. M. T. Lin. 1993. "Minimizing the Makespan in Three-Machine Assembly Type Flow ShopProblem." *Management Science*, 39, 616–625.

Lin, B. M. T., and F. J. Hwang. 2011. "Total Completion Time Minimization in a 2-Stage Differentiation Flowshop with Fixed Sequences per Job Type." *Information Processing Letters*, 111, 208–212.

Lin, B.M.T., and J. M. Wu. 2005. "A simple lower bound for two machine flowshop scheduling to minimize total completion time." *Asia Pacific Journal of Operational Research*, 22, 391–408.

Linn, R., and W. Zhang. 1999. "Hybrid Flow Shop Scheduling: A Survey." *Computers and Industrial Engineering*, 37, 57–61.

Mosheiov, G., and U. Yovel. 2004. "Comments on Flow Shop and Open Shop Scheduling with a Critical Machine and Two Operations per Job." *European Journal of Operational Research*, 157, 257–261.

Nowicki, E., and S. Zdrzalka. 1988. "A Two-Machine Flowshop Scheduling Problem with Controllable Job Processing Times." *European Journal of Operational Research*, 34, 208–220.

Rahimi-Vahed, A. R., B. Javadi, M. Rabbani, and R. Tavakkoli-Moghaddam. 2008. "A Multi-Objective Scatter Search for A Bi-criteria No-Wait Flow Shop Scheduling Problem." *Engineering Optimization* 40, 331–334.

Reisman, A., A. Kumar, and J. Motwani. 1997. "Flow Shop Scheduling/Sequencing Research 1952–1994: A Statistical Review of the Literature." *IEEE Transactions on Engineering Management*, 44, 316–329.

Tran, T. H., and K. M. Ng. 2012. "A Hybrid Water Flow Algorithm for Multi-Objective Flexible Flow Shop Scheduling Problems." *Engineering Optimization*, forthcoming.