Int. J. Production Economics 141 (2013) 286–294

Contents lists available at SciVerse ScienceDirect

Int. J. Production Economics

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

Two-machine flow shop scheduling of polyurethane foam production

Bertrand M.T. Lin*, Y.-Y. Lin, K.-T. Fang

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

ARTICLE INFO

Article history:
Received 23 June 2011
Accepted 1 August 2012
Available online 16 August 2012

Keywords:
Polyurethane foam
Flow shop
Precedence constraints
Branch-and-bound algorithm
Heuristic
Iterative local search

ABSTRACT

This paper studies a two-machine flow shop scheduling problem with a supporting precedence relation. The model originates from a real production context of a chemical factory that produces foam-rubber products. We extend the traditional two-machine flow shop by dividing the operations into two categories: supporting tasks and regular jobs. In the model, several different compositions of foam rubber can be mixed at the foam blooming stage, and products are processed at the manufacturing stage. Each job (product) on the second machine cannot start until its supporting tasks (parts) on the first machine are all finished and the second machine is not occupied. The objective is to find a schedule that minimizes the total job completion time. The studied problem is strongly NP-hard. In this paper, we propose a branch-and-bound algorithm incorporating a lower bound and two dominance rules. We also design a simple heuristic and an iterated local search (ILS) algorithm to derive approximate solutions. The performances of the proposed algorithms are examined through computational experiments.

1. Introduction

This research investigates a flow shop scheduling model inspired by a real production line of polyurethane (PU) foam at a manufacturing site in central Taiwan. Due to different chemical compositions, various types of foams, including general-PU foam, inert foam, viscoelastic (VE) foam and bamboo charcoal foam, can be produced by mixing different materials on a foam blooming machine. While only one foam blooming machine is available, a certain amount of each composition type can be processed at a time. When a composition is finished, the foam can be segmented or sliced into specific sizes for different final products on another machine. To synthesize a final product (job), different types of compositions could be required. Consider the following example production scenario. There are four products to be produced—multi-layer mattress, single-layer mattress, memory pillow and seat pad. Materials required for producing the above products include general PU foam, inert foam and bamboo charcoal foam. The combination of materials for the four products is shown below:

Product 1 Multi-layer mattress requires general-PU foam and inert foam;
Product 2 Single-layer mattress requires general PU foam;
Product 3 Memory pillow requires inert foam;
Product 4 (Seat pad) requires general PU foam and bamboo charcoal foam.

Fig. 1 depicts a production sequence (Product 2, Product 3, Product 1, Product 4). Note that Product 1 cannot be produced until the processing of general PU foam and inert foam is finished at the first stage.

The above foam production environment can be modeled as a two-machine flow shop. Johnson's seminal work (1954) has spurred extensive research works on flow shop scheduling with new manufacturing settings and different objective functions (El-Bouri et al., 2008; Fondrevelle et al., 2009; Haouari and Hidri, 2008; Haq et al., 2010; Yang, 2010). A flow shop consists of several machines arranged in series, and each stage consists of a single machine such that all jobs or products must visit the machines along the specified route. To minimize the time required for finishing all jobs in a two-machine flow shop, Johnson (1954) proposed an elegant algorithm that can solve the problem in polynomial time. In a two-machine flow shop, each job (product) has two operations to process on the machines subject to the specified route, i.e. all jobs need to visit the first machine and then the second machine. Moreover, each operation on the second machine cannot start until the corresponding operation on the first machine is finished and the second machine is not occupied. The problem studied in this research is an extension of flow shop scheduling in the following aspects: For a specific product, it requires one or more types of foams. Preparation of the required foams is performed on the first machine, while production of the final products is carried out

* Corresponding author. Tel.: +886 3 5729915; Fax: +886 3 5131472
E-mail address: bmtlin@mail.nctu.edu.tw (B.M.T. Lin).

0925-5273/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ijpe.2012.08.006

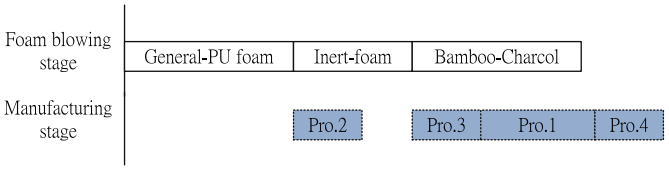

Fig. 1. Gantt chart of an example production schedule.

on the second machine. The production process of a product on the second machine cannot start until the second machine is available and all the required foams are prepared and ready for use. The production model in this research exhibits a clear difference from traditional two-machine flow shop scheduling. A multiple-to-multiple relation exists between machine-one operations and machine-two operations because a type of foam can support one or more products and vice versa. This feature exhibits a significant difference from the one-to-one relationship inherent in the traditional two-machine flow shop. In summary, in the studied model all operations are categorized into two types: *supporting tasks* and *regular jobs*. Machine one is dedicated to the supporting tasks and machine two to the regular jobs. A job can be processed only if the second machine is free and all of its supporting tasks have been done on the first machine. The model was also studied by Chen and Lee (2009) in the context of cross-docking to minimize the makespan of the jobs on the second machine. This paper will investigate the scheduling problem of minimizing the sum of job completion times on the second machine, in short, *the total job completion time*. This objective function reflects not only the service quality, indicating the average customer waiting time, but also the work-in-process inventory cost.

The rest of this paper is organized as follows. Section 2 presents formal statements of the problem definition as well as reviews related previous works. In Section 3, we will propose an integer linear programming model and some preliminary properties that will assist the development of solution algorithms. Section 4 is dedicated to the development of a lower bound and two dominance rules that are to be included in a branch-and-bound algorithm for solving the problem optimally. To derive production schedules in an acceptable time, two approximation algorithms, including a greedy heuristic and an iterated local search (ILS) algorithm, are designed in Section 5. A computational study on the proposed algorithms is given in Section 6. We give conclusions and suggest potential research directions in Section 7.

2. Problem statements and literature review

This section first gives formal statements of the studied problem. Notation and an example follow. Review on related works will also be presented.

The scheduling problem is formally defined as follows: There are two disjoint sets of operations $A=\{a_1,a_2,\ldots,a_m\}$ and $B=\{b_1,b_2,\ldots,b_n\}$ to process on two machines M_A and M_B, respectively. Processing times of $a_i\in A$ and $b_j\in B$ are denoted by p_i^a and p_j^b, respectively. The relation between the operations of sets A and B is specified by the supporting relation $\mathcal{R}\subseteq A\times B$ such that for $a_i\in A$ and $b_j\in B$, if $(a_i,b_j)\in\mathcal{R}$ then operation b_j cannot start on machine M_B unless operation a_i is complete on machine M_A. Hereafter, we call the elements of set A (*supporting*) tasks and the elements of set B (*regular*) jobs. Let α_i denote the subset of jobs supported by task i, and β_j the subset of tasks supporting job j. For example, if $\mathcal{R}=\{(a_1,b_1),(a_1,b_2),(a_2,b_1),(a_3,b_2)\}$ then $\alpha_1=\{1,2\},\alpha_2=\{1\},\alpha_3=\{2\}$, and $\beta_1=\{1,2\},\beta_2=\{1,3\}$.

The uniqueness of the studied problem lies in the virtual but mandatory role of supporting operations of set A. The setting of machine M_A and machine M_B can be treated as a flow shop. The sum of processing times of the tasks in β_j on machine M_A corresponds to the processing time of job b_j on the first machine in a traditional two-machine flow shop, where relation $\mathcal{R}$ is a one-to-one and onto function, or in other words, $|\alpha_i|=1$ for all tasks a_i and $|\beta_j|=1$ for all jobs b_j. This is depicted in Fig. 2. In the studied problem, the supporting tasks of a job are not always processed consecutively on machine M_A. Similarly, the jobs supported by a task are not required to be executed consecutively on machine M_B, either. Therefore, the structure of the problem setting is much more complicated. This paper investigates the objective function of the total completion time.

Throughout the paper, a sequence of tasks on machine M_A is denoted by $s=(s_1,\ldots,s_m)$, and a sequence of jobs on machine M_B by $S=(S_1,\ldots,S_n)$. In a particular schedule, C_i^A denotes the completion time of task a_i on M_A, and C_j^B the completion time of job b_j on M_B. Function $Z(s,S)$ gives the objective value under sequences s and S. Later, we will show that parameter s can be omitted for it can be determined once a job S is given.

To illustrate the problem definition, we consider the following instance. There are five tasks $A=\{a_1,a_2,a_3,a_4,a_5\}$ and four jobs $B=\{b_1,b_2,b_3,b_4\}$. The processing times are shown below.

tasks	a_1	a_2	a_3	a_4	a_5
p_i^A	6	3	2	5	9

jobs	b_1	b_2	b_3	b_4
p_j^B	10	3	1	7

The supporting relation is $\mathcal{R}=\{(a_1,b_1),(a_1,b_3),(a_2,b_1),(a_2,b_4),(a_3,b_2),(a_3,b_3),(a_4,b_3),(a_5,b_4)\}$, as depicted in Fig. 3(a). Given sequence $s=(3, 4, 1, 2, 5)$ on M_A and sequence $S=(2, 3, 1, 4)$ on M_B, we have the corresponding Gantt chart shown in Fig. 3(b). The completion times of the jobs on machine M_B are 5, 14, 26 and 33. Therefore, the total completion time is 78.

Since Johnson's paper (1954), flow shop scheduling has been widely studied in the literature. While the minimization of makespan in a two-machine flow shop ($F2||C_{\max}$) can be solved by Johnson's $O(n\log n)$-time algorithm, the minimization of total completion time ($F2||\sum C_j$) is nevertheless strongly NP-hard (Garey et al., 1976). With the strongly NP-hard $F2||\sum C_j$ problem as a special case, the problem we are considering is also computationally intractable. To the best of our knowledge, there are two models related to the production setting of this paper. First, Chen and Lee (2009) studied a cross-docking problem where a warehouse receives goods from various vendors and then repackages the goods for distributions to various destinations. The operations can be regarded as the two-machine flow shop setting investigated in this paper. They proved the problem of makespan minimization to be strongly NP-hard, proposed a branch-and-bound algorithm, and designed a heuristic algorithm. It is shown that the ratio between the heuristic solution and optimal solution is not greater than 3/2. The second related model is due to Lin et al. 2010 where all of the supporting tasks and the regular jobs are processed on a single machine. The model stems from streaming and scheduling of multi-media objects. They discussed the complexity status of three objective functions, namely $L_{\max}$, $\sum w_jC_j$, and $\sum w_jU_j$, on the single-machine setting and extended the existing complexity results of single-machine scheduling with precedence constraints.

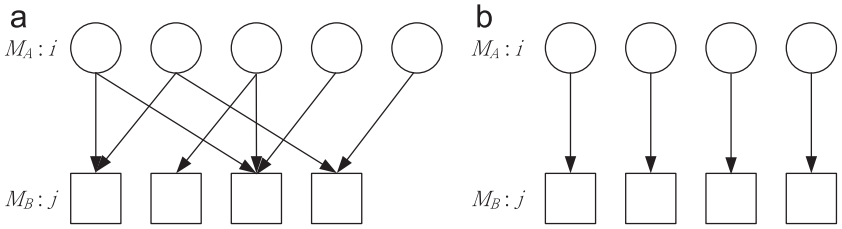

Fig. 2. Flow shop with a supporting relation and traditional flow shop. (a) Flow shop with a supporting relation. (b) Traditional flow shop.

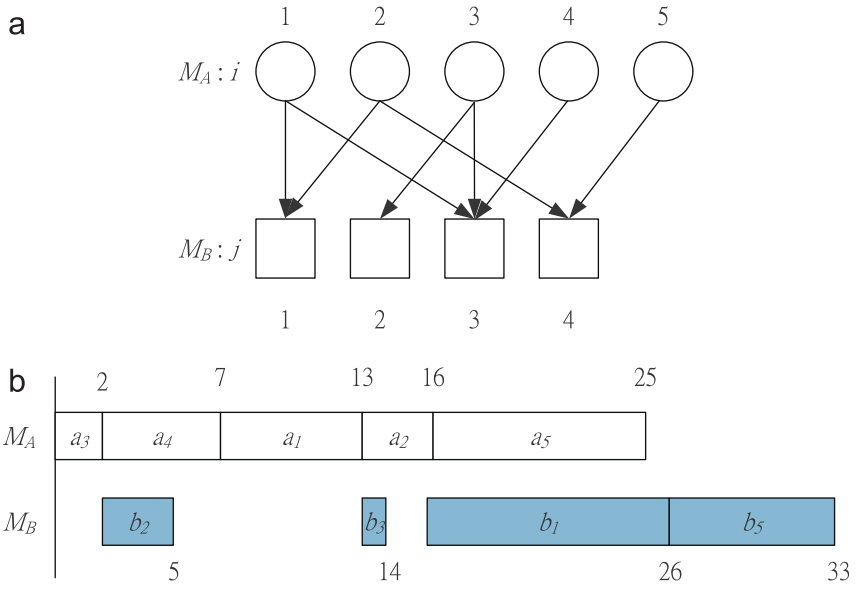

Fig. 3. Example for problem definition. (a) Example of supporting relation. (b) Gantt chart of the example schedule.

3. Mathematical model and preliminary properties

In this section, a mathematical programming model will be given to describe the problem. We will develop two properties that not only address complexity issues but also serve as the base of the development of exact and approximation algorithms.

Integer linear program. There are three approaches commonly adopted for formulating scheduling problems into integer (linear) programs; namely, positional, time-indexed, and sequencing (Ziaee and Sadjadi, 2007). In this paper, we adopt positional variables in the program. Let x_{ik} be a binary variable that equals 1 if task a_i is scheduled at position k on machine M_A and equals 0 otherwise. Let y_{jl} be a binary variable that equals 1 if job b_j is scheduled at position l on machine M_B and equals 0 otherwise.

(ILP)

$$\text{Minimize} \quad \sum_{1 \le j \le n} C^B_{[j]} \tag{1}$$

$$\text{subject to} \quad C^A_{[1]} = \sum_{i=1}^{m} x_{i1} p^a_i; \tag{2}$$

$$C^A_{[k]} = C^A_{[k-1]} + \sum_{i=1}^{m} x_{ik} p^a_i, \quad 2 \le k \le m; \tag{3}$$

$$C^B_{[l]} \ge C^B_{[l-1]} + \sum_{j=1}^{n} y_{jl} p^b_j, \quad 2 \le l \le n; \tag{4}$$

$$\sum_{l=1}^{n} y_{jl} C^B_{[l]} \ge \sum_{k=1}^{m} x_{ik} C^A_{[k]} + p^b_j \quad \forall (i,j) \in \mathcal{R}; \tag{5}$$

$$\sum_{i=1}^{m} x_{ik} = 1, \quad 1 \le k \le m; \tag{6}$$

$$\sum_{j=1}^{n} y_{jl} = 1, \quad 1 \le l \le n; \tag{7}$$

$$\sum_{k=1}^{m} x_{ik} = 1, \quad 1 \le i \le m; \tag{8}$$

$$\sum_{l=1}^{n} y_{jl} = 1, \quad 1 \le j \le n; \tag{9}$$

$$x_{ik}, y_{jl} \in \{0,1\}, \quad 1 \le i,\ k \le m,\ 1 \le j,\ l \le n. \tag{10}$$

Eq. (1) gives the objective function of minimizing the sum of job completion times. Constraints (2) enforce the processing of any a_i on machine M_A to start from time 0 onward. Constraints (3) dictate that a supporting task cannot be processed before its immediate predecessor is completed on machine M_A. Constraints (4) give similar restrictions for jobs on machine M_B. Constraints (5) require that job b_j cannot start on machine M_B before all of its supporting tasks are completed on machine M_A. Constraints (6) and (7) enforce that exactly one operation is allowed at any specific position on either machine. On the contrary, any task or any job can be allocated to exactly one position on their dedicated machines. This is reflected in constraints (8) and (9). Constraints (10) confine the decision variables to be either 0 or 1. The model uses m^2+n^2 decision variables and $O(mn)$ constraints.

Preliminary properties. In this section, we present the properties that will be used in the development of exact and approximation algorithms. The first result states that if a processing

sequence of jobs of set B is given a priori, then a sequence of supporting tasks of set A which attains the minimum total completion time can be determined in polynomial time.

Algorithm. *S-to-s*

Begin
Let $S=(S_1,\ldots,S_n)$ be a given job sequence, and $s=()$ an empty task sequence.
For $j=1$ **To** n
Begin
Set $s=s\oplus s(S_j)$, where $s(S_j)$ is an arbitrary permutation of the tasks of β_{S_j}, and $\oplus$ is a sequence concatenation operator.
For $j'=j+1$ **To** n
Begin
Set $\beta_{S_{j'}}=\beta_{S_{j'}}\backslash\beta_{S_j}$.
End
End
Return sequence s.
End

In the algorithm, the loop on index j iteratively augments sequence s by appending a subsequence of tasks supporting job j, and the loop on index j' removes the newly sequenced tasks from the supporting relation with the unscheduled jobs.

Theorem 1. *Given a sequence S of jobs of set B,* Algorithm *S-to-s optimally produces a sequence of tasks of set A in* $O(|\mathcal{R}|)$ *time.*

Proof. Let $s'\neq s$ be an optimal sequence of tasks on machine M_A subject to the given sequence S on machine M_B. Assume i is the smallest index such that $s_i\neq s'_i$. Then, in sequence s', there must be some position $l>i$ with $s_i=s_{l'}$. Move task $a_{s_{l'}}$ backward to the position immediately preceding task $a_{s'_i}$. It is clear that the completion time of any job will not increase after the move. Repeating the move, if necessary, we can come up with sequence s without increasing the total completion time. □

Theorem 1 states that when the sequence of jobs on machine M_B is settled, the sequence of tasks on machine M_A can be subsequently determined in a greedy manner. It is interesting to consider the reverse of the scenario. Given a processing sequence of tasks on M_A, it however remains hard to determine an optimal sequence of jobs on machine M_B.

Theorem 2. *Given a sequence s of tasks of set A, to determine an optimal sequence of jobs of set B is strongly NP-hard.*

Proof. We show the hardness of determining a optimal sequence of jobs of B via a reduction from the single-machine scheduling problem of minimizing total completion time subject to release dates, i.e. $1|r_i|\sum C_i$ (Garey and Johnson, 1979). Consider an instance of t jobs with processing times p_i and release dates r_i of the $1|r_i|\sum C_i$ problem. Re-index the jobs in non-decreasing order of release dates, i.e. $r_1\leq r_2\leq\cdots\leq r_t$. An instance of the scheduling problem is constructed by letting $m=n=t; p_1^a=r_1; p_i^a=r_i-r_{i-1}$ for $2\leq i\leq n$; $p_j^b=p_j$ for $1\leq j\leq n$. Given the sequence $s=(1,2,\ldots,t)$ of supporting tasks, we want to find an optimal sequence of jobs. Since this is equivalent to solving an instance of the problem $1|r_i|\sum C_i$ which is NP-hard, the problem considered is NP-hard as well. □

From the problem definition, to reach an optimal schedule the decision is to compose an optimal combination of machine-one sequence s and machine-two sequence S. The number of possible combinations is $O(m!n!)$, which explodes exceedingly fast when m and n grow large. With the above two properties, it is evident that the hardness of the studied problem is mainly due to the sequencing issue on the second machine. Therefore, the development of solution algorithms, either exact or approximation, can be collated as sequencing the jobs rather than examining the complicated combinations of task sequences and job sequences. That is, the solution space is reduced to $O(n!)$, although it is still exponential in terms of input length.

4. Branch-and-bound algorithm

The strong NP-hardness indicates that it is very unlikely to design a polynomial time algorithm for producing optimal solutions. Branch-and-bound algorithm is one of the exact methods widely adopted for tackling hard optimization problems. Effective lower bounds and dominance rules, used to prune off non-promising solutions, are crucial to the efficiency of branch-and-bound algorithms. Based upon the optimality properties addressed in the previous section, we develop a lower bound and two dominance rules. To provide a better starting point for the branch-and-bound algorithm, a heuristic is proposed to provide the initial incumbent values. As for the branching rule, we adopt the depth-first search strategy, which exhibits the advantages of low memory requirement and easy implementations. The unscheduled jobs are examined for further recursion in the order specified by Johnson's sequence.

Lower bound. By Theorem 1, the enumeration tree will be explored to enumerate all possible job sequences, i.e. each node corresponds to a subset of scheduled jobs. Given a partial job sequence, the algorithm in the search tree produces the corresponding partial sequence of supporting tasks using Algorithm *S-to-s*. Note that Algorithm *S-to-s* can be deployed to find a task sequence minimizing the makespan ($C_{\max}$) as well because the proof of Theorem 1 still works. The task sequence for minimizing makespan subject to a given job sequence is crucial to further discussion because it provides the validity of the development of dominance rules and lower bounds.

To minimize the total job completion time, the studied problem is reshaped into a single-machine scheduling problem with different release times (i.e. $1|r_j|\sum C_j$ problem). Ahmadi and Bagchi (1990) tackled this problem by exploiting the SRPT (Shortest Remaining Processing Time) rule to solve the preemptive relaxation in $O(n\log n)$ time. Adjustment is needed before deploying the same technique. To create an instance of problem $1|r_j|\sum C_j$, let p_j^b be the processing time of job J_j, and let the sum $\sum_{i\in\beta_j}p_i^a$ be the release date r_j of job J_j. It is worthy to emphasize again that the supporting tasks are shared. As a consequence, given a partial sequence S in the enumeration tree, the release date r_j of unscheduled job J_j is recalculated by excluding the supporting tasks that are already finished on machine M_A. Then, the adjusted release dates are used to implement the SRPT rule.

Herewith we use the same numerical example of Section 2. Assume that the partial sequence S consists of only job b_2. Then, $C_1^A=2$ and $C_1^B=5$. The sums of processing lengths of supporting tasks for the unscheduled jobs b_1,b_3,b_4 become 9, 11 and 9, respectively. The new release dates r_j are now 11, 13 and 11, respectively, by including C_1^A. Consequently, the values shown in Fig. 4 follow.

With $C_3^B(SRPT)=14, C_4^B(SRPT)=21, C_1^B(SRPT)=31$, the lower bound on the total completion time is calculated as $5+14+21+31=71$.

Dominance rules. Making good use of dominance rules can curtail a great amount of unnecessary nodes of an enumeration tree. Dominance rules tell that if certain conditions are met, then some branches can be eliminated without sacrificing the optimal solution.

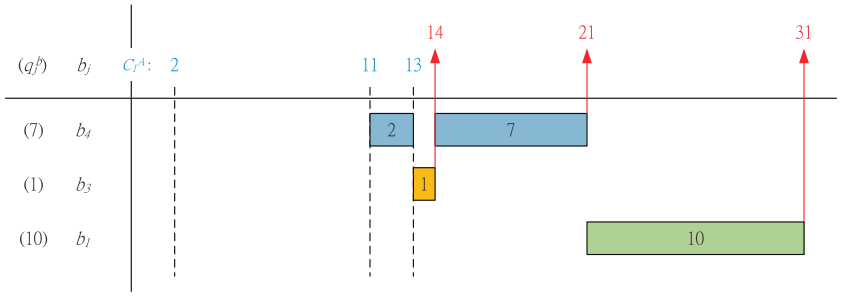

Fig. 4. Result of the SRPT rule.

D1: *Consider a partial sequence σ and two jobs b_x and b_y to be scheduled in the next two consecutive positions after σ. If $\beta_x \subseteq \beta_y$ and $p_x^b \le p_y^b$, then the node rooted at partial schedule $\sigma b_y b_x$ can be eliminated.*

Proof. Assume partial sequence σ and jobs b_y,b_x satisfy the conditions of D1. Let t be the starting time of job b_y on M_B subject to $\sigma b_y b_x$. Because $\beta_x \subseteq \beta_y$, job J_x is ready for processing on M_B when job b_y is finished. The sum of completion times of b_x and b_y in sequence $\sigma b_y b_x$ is thus $(t+p_y^b)+(t+p_y^b+p_x^b)$. Consider the sequence $\sigma b_x b_y$ obtained from swapping the positions of jobs b_y and b_x. The starting time t' of job b_x is not later than t, i.e. $t' \le t$. The starting time of job b_y is not greater than $\max\{t,t'+p_x^b\}$. The sum of completion times of b_x and b_y in $\sigma b_x b_y$ is not greater than $(t'+p_x^b)+(\max\{t,t'+p_x^b\}+p_y^b)$. The premises $t' \le t$ and $p_x^b \le p_y^b$ boil down to $(t'+p_x^b)+(\max\{t,t'+p_x^b\}+p_y^b) \le (t+p_y^b)+(t+p_y^b+p_x^b)$. Moreover, the completion time of $\sigma b_x b_y$ is $\max\{t,t'+p_x^b\}+p_y^b$, which is not greater than the completion time $t+p_y^b+p_x^b$ of partial schedule $\sigma b_y b_x$. In other words, partial schedule $\sigma b_x b_y$ has a shorter completion time and a smaller total completion time than partial schedule $\sigma b_y b_x$. □

Note that the two jobs of concern in **D1** must be adjacent, otherwise the rule may not work. The second dominance rule follows from the calculation of the cost already incurred by the scheduled jobs and an estimate of the cost of the remaining jobs. Della Croce et al. (1996) deployed the basic concept of dynamic programming approach to formulate a dominance rule, which states that for two different partial schedules S and S' of the same subset B' of jobs, if S has a smaller total completion time and a shorter makespan, then schedule S' can be pruned. To realize this concept in the studied problem, we consider the following inequality:

$$Z(S)-Z(S')+(n-|B'|)(C_{\max}^{S}-C_{\max}^{S'}) \le 0, \tag{11}$$

where $C_{\max}^{S}$ (respectively, $C_{\max}^{S'}$) denotes the makespan of S (respectively, S'). If partial schedule S has a smaller objective value and completes the jobs earlier, then Eq. (11) is satisfied and the partial schedule S' is fathomed. Note that in the case where partial schedule S completes the jobs later (i.e. makespan is larger), if the advantage in the total completion time $(Z(S)-Z(S'))$ overrides the inferior influence over the unscheduled jobs $((n-|B'|)(C_{\max}^{S}-C_{\max}^{S'}))$, then we still can curtail the branching toward partial schedule S'. The discussion leads to the second dominance rule.

D2: *If inequality (11) is satisfied, then the subtree rooted at partial schedule S' can be eliminated.*

Branch-and-bound algorithm. With the lower bound and two dominance rules addressed above, we design a branch-and-bound algorithm for deriving optimal solutions to the studied problem. Depth-first search is employed as the branching strategy for the branch-and-bound algorithm. Such a strategy merits simplicity in implementation and avoids exceeding demand for computer memory. The branch-and-bound algorithm starts with an initial solution derived by a greedy heuristic algorithm designed in the next section to have a better starting point. During the course of exploration of the search tree, rule **D1** is applied to determine if a branch can be dropped or not. When the answer is negative, the lower bound of the partial solution will be calculated. If the incumbent value is less than or equal to the lower bound, then we eliminate the node. If the node passes the two tests, rule **D2** is examined. The exploration proceeds until all nodes are visited or pruned. If the algorithm can complete the exploration within a given limit on the execution time, the optimal solution will be found and reported. On the other hand, if the algorithm does not complete its execution when the specified time limit is elapsed, then it will abort with failure and report the best incumbent value found thus far.

5. Heuristic algorithms

Although the branch-and-bound approach is designed to produce optimal solutions, an exceedingly long running time can be demanded when the problem size grows large. For large-scale problems, optimal solutions cannot be obtained within reasonable time. Therefore, seeking effective approximate solutions is an alternative for decision makers in practical applications. In this section, we will develop a heuristic as well as a meta-heuristic to produce quality schedules in an acceptable time. The heuristic method dispatches the jobs in a greedy manner. The algorithm is outlined as follows.

HEURISTIC G

Step 1 : Let $A'=\emptyset$ be the set of scheduled supporting tasks. Calculate the total processing time $p_j=p_j^b+\sum_{i\in\beta_j}p_i^a$ for each job b_j, where $\beta_j=\beta_j\backslash A'$.

Step 2 : If there are jobs that will not induce idle times on machine M_B, then select from these jobs the one that attains the minimum makespan when dispatched; otherwise, select the job that has the minimum makespan when dispatched. Call this job b_{j^*}. Set $A'=A'\cup\beta_{j^*}$ and $\beta_k=\beta_k\backslash\beta_{j^*}$ for all unscheduled jobs b_k. Repeat the process until all jobs are scheduled.

Step 3 : Stop and report the solution.

Note that when selecting the next job to schedule, if there are more than one job inducing the smallest makespan, we select the one that has the largest number of supporting tasks. This can be easily justified. If all of the supporting tasks on machine M_A are finished, then the remaining jobs on machine M_B can be processed by the SPT rule directly.

We next design an *iterated local search* (*ILS*) algorithm to improve the solution derived by the greedy Heuristic G. Iterated

local search is a simple and powerful meta-heuristic that iteratively applies local search to improve the solutions. ILS consists in three main components: *Local Search*, *Perturbation* and *Acceptance Criterion*. Local Search finds local optimum by an iterative improvement method. The Perturbation phase is to avoid being trapped at a local optimal solution so as to explore more regions of the solution space. The Acceptance Criterion gives the conditions of the acceptance of new solutions. With regard to the literature, the basic structure of ILS is given in Stutzle (1998) and can be found also in Lourenco et al. (2001). Stutzle (1998) and Dong et al. (2009) applied ILS to solve the permutation flow shop problem and compared it with other heuristics for performance assessment. ILS is also applied to resource-constrained project scheduling (Ballestin and Trautmann, 2008), vehicle routing (Prins et al., 2009), scheduling of machines and automated guided vehicles (Deroussi et al., 2007), just to name a few. Previous works showed that ILS runs fast and can convey impressive solution quality. The construct of ILS can be outlined as follows:

```
Procedure Iterated Local Search
Generate initial solution S0.
S = LocalSearch(S0).
repeat
  S′ = Perturbation(S).
  S″ = LocalSearch(S′).
  S = AcceptanceCriterion(S,S″).
until termination conditions are met.
```

Detail specifications for each stage of the iterated local search are elaborated in the following.

Initial solution: The initial solution is generated by Heuristic G.

Local search: One of the major ingredients of local search algorithms is concerned about the neighborhood structures. The approaches commonly adopted for defining the neighborhood of a solution include, although not limited to, (1) swap-moves that swap two neighboring positions i and $i+1$, (2) exchange-moves that swap two arbitrary positions i and j, and (3) insertion-moves that select two positions i and j and insert the job at the i-th position to the j-th position (Stutzle, 1998). This paper uses insertion-moves, which are realized by two `for` loops to ensure that every pair of positions is considered. During the iterative process, if a better solution or sequence is encountered, the local search procedure will start the insertion-moves again from scratch.

Perturbation: Due to the fact that the solution yielded by a local search algorithm is not necessarily global optimal, the perturbation phase of solution modifications is a very important mechanism to allow for the possibility of exploring other regions of the solution space. It can enable escape from local optimum. Ruiz and Stutzle (2007) applies insertion neighborhood of perturbation, which is commonly regarded as being a very good choice for the permutation flow shop problem. Tasgetiren et al. (2011) tested perturbation values ranging from 1 to 20, and their experiments showed that swap-moves or insertion-moves generated better results with perturbation values equal to 1 or 2. In the studied problem, the supporting relation in some sense is a kind of precedence constraint, from tasks to jobs. If we adopt a "macro" operation, say 2-opt moves, on S, then the structure of sequence s will be drastically changed. With such macro operations, diversity could be achieved, but in the meantime fluctuation would also emerge. Therefore, we adopt exchange-moves in the perturbation phase. Applying an exchange-move of two randomly selected job positions i and j to the schedule $S=(1,\ldots,i,\ldots,j,\ldots,n)$ yields the new schedule $S'=(1,\ldots,j,\ldots,i,\ldots,n)$. After the perturbation, sequence S' is used as the seed for invoking local search. After that, a new solution S'' emerges.

Acceptance criterion: As for acceptance criterion, we consider the simple one that has been widely adopted by previous ILS works: If the objective value of new schedule S'' is better than that of the original schedule S, then we accept sequence S'' and replace the current schedule S with it.

In the implementation of ILS for the studied problem, the termination condition is defined by an upper limit on the number of iterations exercised. The limit is set to be 200 iterations.

6. Computational study

Previous sections presented a branch and bound algorithm for finding exact solutions, and a greedy heuristic and a meta-heuristic ILS algorithm for approximate solutions. This section is dedicated to computational experiments for comparison and analysis of the performances of the proposed algorithms. The programs were coded in C++ and executed on a personal computer with an Intel Pentium(R) 4 CPU (3.4 GHz) with 0.99G RAM running Microsoft Windows XP Professional. Function "*srand*((*unsigned*)*time*(*NULL*))" was used to generate random numbers. Processing times were randomly drawn from the uniform discrete interval [1, 100]. For each pair of task a_i and job b_j, if a random number drawn from interval [0, 1] is smaller than parameter 0.5, then task a_i supports job b_j, i.e. $(a_i,b_j)\in\mathcal{R}$. The parameters m and n representing the numbers of operations on the two machines were taken into account simultaneously for determining problem sizes. For each combination of n and m, we generated and tested 10 independent instances. In the following, the numerical results from the experiments are analyzed in two parts. The first part is for exact solutions and the second part is for approximate solutions.

In the first part concerning optimal solutions, two branch-and-bound algorithms were examined. The first one (B&B) is equipped with the lower bound derived from the SRPT solution of the $1|r_j|\sum C_j$ problem, and the second one (B&B_D) further incorporates two dominance rules D1 and D2. Heuristic G was deployed to generate initial solutions for the branch-and-bound algorithms. A limit of 30 min was given to the execution of the branch-and-bound algorithms. If the algorithms could not completely examine the enumeration tree, then they aborted with failures. As some of the instances were not successfully solved, we kept track the detail statistics of only those optimally solved. The performance indices investigated in the experiment include the number of instances optimally solved (#Opt), the average number of nodes explored (Nodes), and the average elapsed running time (Time). The problem size n ranges from 10 to 40 with an increment of 5. Table 1 summarizes the performance results of the two branch-and-bound algorithms with the sizes of $m=\lceil 0.25n\rceil$, $m=\lceil 0.5n\rceil$ and $n=m$. The numerical values for each problem size n and m are averaged over the solved instances from each 10 problem instances.

We can readily see from the statistics that the execution time and the number of visited nodes are significantly reduced if the two dominance rules are incorporated. For example, for solving the setting of $n=25$ and $m=\lceil 0.25n\rceil$, the average running time of algorithm B&B is 82.09 s, and that of algorithm B&B_D is only 3.57 s. The results also suggest that the values of m plays a crucial

Table 1
Computational results of branch-and-bound algorithms.

n	#Opt	B&B						#Opt	B&B_D					
		Time			Node				Time			Node		
		Min	Avg	Max	Min	Avg	Max		Min	Avg	Max	Min	Avg	Max
$m=0.25n$														
10	10	0.00	0.00	0.00	10	209	552	10	0.00	0.00	0.00	10	94	252
15	10	0.00	0.06	0.23	43	15,236	56,580	10	0.00	0.11	0.03	40	3023	11,454
20	10	0.02	0.58	3.34	1919	100,047	586,493	10	0.00	0.08	0.23	675	13,329	39,885
25	10	0.77	82.09	623.70	80,595	9,289,494	69,918,835	10	0.23	3.57	20.55	27,610	451,817	2,656,050
30	7	6.36	326.27	719.64	513,969	27,366,574	60,553,746	9	1.03	37.26	195.66	92,865	3,267,712	17,019,896
35	3	147.59	387.96	704.34	8,177,593	21,860,926	40,219,927	8	15.20	202.93	505.78	1,030,298	13,734,189	35,065,211
40	1	1327.81	1327.81	1327.81	57,060,164	57,060,164	57,060,164	4	102.56	501.39	876.92	5,226,700	27,285,385	49,296,815
$m=0.5n$														
10	10	0.00	0.00	0.02	37	616	2190	10	0.00	0.00	0.00	37	289	947
15	10	0.06	0.63	3.52	14,567	161,138	920,569	10	0.03	0.12	0.31	9264	29,908	78,183
20	10	1.27	22.97	97.97	146,826	2,923,146	12,515,288	10	0.25	1.76	4.33	33,639	233,734	572,985
25	8	62.50	524.13	1246.00	5,109,862	4,594,456	112,392,401	10	9.74	48.19	151.77	828,691	4,394,067	13,999,552
30	4	244.91	977.89	1400.33	14,997,852	21,267,628	88,587,485	7	35.09	205.82	427.63	2,339,135	13,765,936	28,691,204
35	1	1355.69	1355.69	1355.69	58,578,183	58,578,183	58,578,183	1	223.13	223.13	223.13	11,152,226	11,152,226	11,152,226
$m=n$														
10	10	0.00	0.01	0.03	357	3565	11,182	10	0.00	0.00	0.02	246	1339	2768
15	10	0.50	5.96	18.44	76,983	997,053	3,104,254	10	0.14	0.57	1.05	24,796	102,470	187,893
20	10	21.66	300.23	706.33	2,334,570	36,860,095	89,919,773	10	2.94	13.59	32.52	367,149	1,750,577	4,221,092
25	1	621.48	621.48	621.48	37,474,591	37,474,591	37,474,591	5	96.41	312.39	628.59	6,276,946	21,190,718	42,573,752

role in determining the level of hardness of the problem. As the number of tasks m grows, more entries would emerge in the supporting relation and the problem thus becomes relatively hard to solve. While all instances of 25 jobs with $m=\lceil 0.25n\rceil$ and $m=\lceil 0.5n\rceil$ were solved by algorithm B&B_D, only 5 of the 10 instances with $m=n$ were solved. As shown in the literature, the best exact algorithms of the classical $F2\|\sum C_j$ problem can solve instances of around 40 jobs (Della Croce et al., 1996, 2002; Hoogeveen et al., 2006; Lin and Wu, 2005). The problem setting studied in this paper is much more complicated than $F2\|\sum C_j$. Therefore, the proposed properties are quite useful in curtailing unnecessary branching.

The second part of the experiment was conducted for the appraisal of two approximation algorithms. We used the same test instances and the solution values derived by the branch-and-bound as in Table 1 for $m=\lceil 0.25n\rceil$. The test instances of the settings $m=\lceil 0.5n\rceil$ and $m=\lceil n\rceil$ were not used because less exact solutions were successfully reported. The greedy method Heuristic G was invoked first and then the ILS algorithm was applied for further improvement. Solution values of the problems with n ranging from 10 to 40 are shown in Table 2. The underlined entries are the incumbent values reported when the branch-and-bound algorithm aborted upon the limit of 30 min. The rows entitled "avg" contain the average objective values over each 10 instances. The error ratios in percentages are calculated as

$$\text{Avg\_dev}(\%)=\frac{(\text{Appx}-\text{B\&B\_D})\times 100\%}{\text{Appx}},$$

where Appx and B&B_D respectively denote the solution values given by the approximation approaches and the branch-and-bound algorithms with two dominance rules. The error ratios of ILS are around 0.1%. Examining the numerical results, we can find that ILS even provides better solutions than the branch-and-bound algorithms that failed to completely explore the enumeration tree. Consider the last instance of ($n=40$, $m=10$) as an example, the best solution value ever found before the termination of the two branch-and-bound algorithms is 37,191, while that given by ILS is 36,051. In other words, ILS is not only efficient (using less execution time) but sometimes more effective (giving better solutions) than exact methods within a time limit of 30 min. Another observation is about the execution of branch-and-bound algorithms. Consider the second instance of ($n=40$, $m=10$). Although algorithm B&B_D encountered the optimal solution value 39,713, it did not possess any information to terminate the execution until the time limit was reached.

Next we examine all approaches, including CPLEX implementation of the proposed integer linear programming model, with 10 test instances of ($n=200$, $m=50$). The results are shown in Table 3. For each instance, we first deployed the greedy heuristic G and then improve the solution by ILS. The solutions generated by heuristic G and ILS were used as the initial incumbent values for the branch-and-bound algorithm. The corresponding columns are $B\&B_G$ and $B\&B_I$. A time limit of 30 min was also imposed on the branch-and-bound algorithm and the CPLEX implementation. The column entitled ILS contains the solution reported by two execution runs of the ILS algorithm such that for a given instance, ILS was invoked to provide an initial solution for the branch-and-bound, and the incumbent solution kept at the termination of the branch-and-bound algorithm was used as the seed solution for further improvement by deploying ILS for another 200 iterations. Such a design is due to the fact that the running time of ILS is less than 1 min which is relatively negligible than the execution time (30 min) of CPLEX implementation and branch-and-bound algorithm. The results are shown in the column entitled $B\&B_I+ILS$. From Table 3, it is evident that the branch-and-bound algorithms did not find solutions better than the initial values given by the greedy heuristic or the ILS algorithm. Better solutions can be found when the execution of the ILS algorithm was further exercised. The CPLEX solutions are even more inferior. The average CPLEX solution value is 1,491,376, while the average $B\&B_I+ILS$ solution value is only 1,042,097.

From the above discussion, we may conclude that the proposed lower bound and dominance rules really help to reduce the efforts required for locating optimal solutions. For small-scale instances, ILS can produce optimal solutions or approximate solutions with negligible errors. When the instances contain more jobs, ILS still exhibits impressive performances in comparison with the exact approaches. The results suggest that if no stronger

Table 2
Solution quality for instances with 40 or less jobs.

Instance	$n=10, m=3$				$n=15, m=5$				$n=20, m=5$				$n=25, m=7$			
	B&B	B&B_D	Greedy	ILS	B&B	B&B_D	Greedy	ILS	B&B	B&B_D	Greedy	ILS	B&B	B&B_D	Greedy	ILS
1	2592	2592	2592	2592	5488	5488	5561	5488	8723	8723	8855	8723	13,628	13,628	14,570	13,628
2	3334	3334	3578	3334	5593	5593	6578	5593	9263	9263	9308	9263	13,398	13,398	14,065	13,398
3	2161	2161	2161	2161	4155	4155	5391	4155	9867	9867	10,645	9867	18,019	18,019	18,626	18,019
4	2525	2525	2944	2525	3107	3107	3343	3107	10363	10,363	10,391	10,363	14,327	14,327	16,061	14,327
5	2088	2088	2495	2088	5790	5790	6411	5790	8303	8303	8779	8341	13,529	13,529	13,740	13,529
6	1945	1945	2196	1945	4251	4251	4877	4251	9482	9482	9497	9482	14,865	14,865	14,865	14,865
7	1565	1565	1565	1565	4510	4510	4514	4510	7029	7029	7600	7029	13,504	13,504	13,808	13,573
8	2228	2228	2505	2228	5532	5532	5772	5532	7591	7591	7627	7591	15,593	15,593	17,265	15,593
9	2458	2458	2458	2458	4253	4253	4296	4253	9649	9649	9649	9649	13,860	13,860	15,403	13,914
10	2136	2136	2220	2136	5936	5936	5936	5936	7503	7503	7866	7503	12,880	12,880	13,658	12,880
Avg_obj	2303.2	2303.2	2471.4	2303.2	4861.5	4861.5	5267.9	4861.5	8777.3	8777.3	9021.7	8781.1	14,360.3	14,360	15,206	14,373
Avg_dev (%)			6.8	0.0			7.7	0.0			2.7	0.0			5.6	0.1

Instance	$n=30, m=8$				$n=35, m=9$				$n=40, m=10$			
	B&B	B&B_D	Greedy	ILS	B&B	B&B_D	Greedy	ILS	B&B	B&B_D	Greedy	ILS
1	20,369	20,369	21,237	20,369	28,890	28,888	30,784	28,888	35,532	35,398	35,532	35,532
2	19,555	19,555	20,319	19,555	25,894	25,894	26,746	25,971	39,713	39,713	40,050	39,713
3	18,772	18,772	20,797	18,772	22,639	22,379	23,708	22,482	38,273	38,273	40,505	37,814
4	20,086	20,069	20,086	20,069	21,284	21,160	21,563	21,284	39,819	39,819	41,023	39,819
5	18,113	18,113	18,119	18,113	30,380	30,302	32,575	30,338	36,939	36,055	38,701	36,055
6	21,873	21,873	24,129	21,873	31,744	31,744	34,824	31,744	38,275	36,767	38,532	36,527
7	22,724	22,724	23,087	22,832	26,373	26,373	26,728	26,373	35,681	35,681	36,174	34,768
8	15,799	15,799	16,080	15,799	24,588	24,588	24,699	24,588	37,456	36,724	39,997	36,464
9	16,397	16,397	17,034	16,397	29,020	29,020	29,700	29,020	40,824	40,389	42,272	39,060
10	18,227	18,227	19,437	18,227	28,687	28,452	31,151	28,452	37,191	37,191	37,363	36,051
Avg	19,601	19,092	20,027	19,104	27,096	27,192	28,498	27,221	39,819	37,746	38,826.5	37,779.8
Avg_dev(%)			5	0			4.6	0.1			2.8	0.1

Table 3
Solution values of 10 large instances.

Instance	$n=200$, $m=50$				
	B&B$_G$	B&B$_I$	Greedy	B&B$_I$+ILS	CPLEX
1	1,064,893	1,063,035	1,064,893	1,060,902	1,475,739
2	995,598	994,345	995,598	990,737	1,422,659
3	1,020,419	1,008,944	1,020,419	1,008,944	1,457,987
4	1,043,034	1,039,199	1,043,034	1,039,199	1,483,254
5	1,194,550	1,179,502	1,194,550	1,179,502	1,649,472
6	1,113,713	1,105,535	1,113,713	1,103,055	1,576,792
7	1,009,519	1,008,472	1,009,519	1,007,271	1,437,228
8	974,819	959,338	974,819	959,338	1,393,268
9	1,020,943	1,012,749	1,020,943	994,125	1,470,480
10	1,087,319	1,077,901	1,087,319	1,077,901	1,546,883
Avg	1,052,481	1,044,902	1,052,481	1,042,097	1,491,376

properties can be developed for the exact approaches, then the designed approximation approaches would be more appropriate for dealing with the studied problem.

7. Concluding remarks

From a real production context of foam-related products, we formulated a two-machine flow shop scheduling problem with supporting precedences. The objective in this research is to minimize the total job completion time. The unique feature of the production model is the multiple-to-multiple relation between machine-one and machine-two operations. We developed an integer linear programming model to interpret the studied problem in a mathematical way. Two preliminary properties about sequencing of tasks and jobs on either machines were proposed. The properties simplify the solution structures and facilitate the development of solution algorithms. A lower bound and two dominance rules were presented to design branch-and-bound algorithms for producing optimal solutions. A greedy heuristic and an ILS algorithm were designed to produce approximate solutions. Through computational experiments, we studied the performance of the proposed properties and algorithms. The branch-and-bound algorithm incorporating the lower bound and two dominance rules can solve the test instances with up to 40 jobs and 10 supporting tasks. The numerical results also suggest that ILS is quite efficient and effective when applied to the test instances with up to 200 jobs. The ILS algorithm resulted in average deviations less than 0.109% from the optimum of the small-scale problems.

For future research, design and analysis of approximation algorithms for the studied problem could be an interesting topic. Settling the complexity of some further restricted cases are also of research interest. For example, the three special cases (1) $\forall i \in A(p_i^a = 1)$; (2) $\forall j \in B(p_j^b = 1)$ and (3) $\forall i \in A(p_i^a = p)$, $\forall j \in B(p_j^b = q)$ may exhibit theoretical challenges. Moreover, we may include another type of precedence among the regular jobs to reflect more practical applications. For example, streaming and scheduling of multi-media presentations especially demands specific subsequences of playbacks (jobs, in this paper).

Acknowledgements

The authors are grateful to the anonymous reviewers for their constructive comments which are very helpful in improving the paper. This research was partially supported by the National Science Council of Taiwan under Grant NSC-100-2410-H-009 -015 -MY2.

References

Ahmadi, R.H., Bagchi, U., 1990. Improved lower bounds for minimizing the sum of completion times of n jobs over m machines in a flow shop. European Journal of Operational Research 44, 331–336.
Ballesti'n, F., Trautmann, N., 2008. An iterated-local-search heuristic for the resource-constrained weighted earliness-tardiness project scheduling problem. International Journal of Production Research 46 (22), 6231–6249.
Chen, F., Lee, C.Y., 2009. Minimizing the makespan in a two-machine cross-docking flow shop problem. Discrete Optimization 193, 59–72.
Della Croce, F., Ghirardi, M., Tadei, R., 2002. An improved branch-and-bound algorithm for the two machine total completion time flow shop problem. European Journal of Operational Research 139, 293–301.
Della Croce, F., Narayan, V., Tadei, R., 1996. The two-machine total completion time flow shop problem. European Journal of Operational Research 90, 227–237.
Deroussi, L., Gourgand, M., Tchernev, N., 2007. A simple metaheuristic approach to the simultaneous scheduling of machines and automated guided vehicles. International Journal of Production Research 46 (8), 2143–2164.
Dong, X., Huang, H., Chen, P., 2009. An iterated local search algorithm for the permutation flow shop problem with total flowtime criterion. Computers and Operations Research 36 (5), 1664–1669.
El-Bouri, A., Balakrishnan, S., Popplewell, N., 2008. Cooperative dispatching for minimizing mean flowtime in a dynamic flowshop. International Journal of Production Economics 113 (2), 819–833.
Fondrevelle, J., Oulamara, A., Portmann, M.-C., Allahverdi, A., 2009. Permutation flow shops with exact time lags to minimise maximum lateness. International Journal of Production Research 47 (23), 6759–6775.
Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freedman, San Francisco, CA.
Garey, M.R., Johnson, D.S., Sethi, R., 1976. The complexity of flow shop and jobshop scheduling. Mathematics of Operations Research 1 (2), 117–129.
Haq, A.N., Ramanan, T.R., Shashikant, K.S., Sridharan, R., 2010. A hybrid neural network-genetic algorithm approach for permutation flow shop scheduling. International Journal of Production Research 48 (14), 4217–4231.
Haouari, M., Hidri, L., 2008. On the hybrid flowshop scheduling problem. International Journal of Production Economics 113 (1), 495–497.
Hoogeveen, H., van Norden, L., van de Velde, S., 2006. Lower bounds for minimizing total completion time in a two-machine flow shop. Journal of Scheduling 9, 559–568.
Johnson, S.M., 1954. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1, 61–68.
Lin, B.M.T., Kononov, A.V., Fang, K.T., Single-machine scheduling with supporting precedence constraints. Technical Report NSC 97-2923-H-009-001-MY3. National Chiao University, Taiwan, 2010.
Lin, B.M.T., Wu, J.M., 2005. A simple lower bound for total completion time minimization in a two-machine flowshop. Asia-Pacific Journal of Operational Research 22 (3), 1–17.
Lourenco, H., Martin, O., Stutzle, T., 2001. A gentle introduction to iterated local search. In: Proceedings of MIC2001, the 4th Metaheuristics International Conference, Porto, Portugal.
Prins, C., Labadi, N., Reghioui, M., 2009. Tour splitting algorithms for vehicle routing problems. International Journal of Production Research 47 (2), 507–535.
Ruiz, R., Stutzle, T., 2006. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177 (3), 2033–2049.
Stutzle, T., 1998. Applying Iterated Local Search to the Permutation Flow Shop Problem. Technical Report, AIDA-98-04, FG Intellektik, TU Darmstadt.
Tasgetiren, M.F., Pan, Q.K., Suganthan, P.N., Chen, A.H.L., 2011. A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. Information Sciences 181, 3459–3475.
Yang, J., 2010. A new complexity proof for the two-stage hybrid flow shop scheduling problem with dedicated machines. International Journal of Production Research 48 (5), 1531–1538.
Ziaee, M., Sadjadi, S.J., 2007. Mixed binary integer programming formulations for the flow shop scheduling problems. A case study: ISD projects scheduling. Applied Mathematics and Computation 185 (1), 218–228.