An improved branch-and-bound algorithm for the preemptive open shop total completion time scheduling problem

Ching-Fang Liaw*

Department of Industrial Engineering and Management, Chaoyang University of Technology, 168, Jifeng East Road, Wufeng District, Taichung, 41349, Taiwan

(Received February 2013; revised May 2013; accepted September 2013)

This article examines the problem of scheduling preemptive open shops to minimize the total completion time. The problem is known to be NP-hard. An efficient constructive heuristic is developed for solving large sized problems. A new lower bound scheme based on the solution of another special type of preemptive open shop problem is presented. Both of the proposed upper and lower bounds are incorporated into a branch-and-bound algorithm to solve medium sized problems. Computational results are reported. The branch-and-bound algorithm can handle problems of up to 14 jobs and 14 machines in size within a reasonable amount of time. The solution obtained by the heuristic has an average deviation of less than 0.14% from the optimal value, while the initial lower bound has an average deviation of less than 2.12% from the optimal value. Moreover, the heuristic finds approved optimal solutions for over 70% of the problem instances completely solved.

Keywords: scheduling; open shop; branch-and-bound; heuristic; completion time

1. Introduction

In this paper, we examine the total completion time preemptive open shop scheduling problem. The characteristics of the problem are as follows. We are given \( n \) independent jobs and \( m \) different machines. Each job \( i, i = 1, 2, \ldots, n \), consists of \( m \) operations, \( O_j, j = 1, 2, \ldots, m \), where operation \( O_j \) of job \( i \) has to be processed on machine \( j \) for \( p_{ij} \) time units. Each machine can process at most one operation at a time, and no two operations of the same job can be processed at the same time. The operations of the same job can be processed in any order. Preemption is allowed, that is, the processing of any operation can arbitrarily often be interrupted and resumed at a later time. Each job \( i, i = 1, 2, \ldots, n \), is available at time zero. The objective is to find a feasible schedule such that the total completion time \( \sum_{i=1}^{n} C_i \) is minimized, where \( C_i \) is the completion time of job \( i \). Following the three-field notation of Graham et al. [7], we denote this problem as \( O_{\infty}[\text{prmp}] \sum C_i \). The problem is NP-hard even for the case \( m = 2 \).[4]

The open shop scheduling model has received considerable research attention (see references in [2,16]) because it occurs in many real-world scheduling environments. In many situations, particularly testing and maintenance, the order in which the operations are processed is immaterial. The teacher-class timetabling is another example where the (preemptive) open shop model is a natural basic formulation. An open shop is similar to a job shop except that the operations of a job can be processed in any order. Relaxing the ordering constraints from a job shop, an open shop has a loose problem structure. This, however, implies that the number of feasible schedules increases tremendously. Thus, the problem of finding optimal open shop schedules by (implicit) enumeration is even more difficult than it is for job shops.

Most variations of open shop scheduling problems are known to be NP-hard. Polynomial time algorithms only exist for a few special cases. A large number of studies have been done on the non-preemptive scheduling of open shops. When preemption is allowed, Gonzalez and Sahni [6] proposed a polynomial time algorithm for the problem \( O_{\infty}[\text{prmp}] \sum C_i \). Improved versions of that algorithm were given by Gonzalez [5]. Zhan, Zhong, and Zhu [19] studied the same problem and presented a network-flow-based algorithm. Sedeño-Noda, Alcaide, and González-Martí [17] introduced a max-flow parametric algorithm to solve the \( O_{\infty}[\text{prmp}] \sum C_i \) problem with time-windows. Kis, Werra, and Kubicki [9,18] examined a multi-machine generalization of the \( O_{\infty}[\text{prmp}] \sum C_i \) problem, and identified some polynomially solvable cases.

Cho and Sanhi [3] presented a linear programming formulation for the problem \( O_{\infty}[\text{prmp}, r_i, d_t] \sum C_i \) where each job \( i \) has a release time \( r_i \) and due time \( d_t \). They also gave polynomial algorithms for special cases of this problem. Lawler, Lenstra, and Rinnooy Kan [10] developed a linear time algorithm (LTA) for the problem \( O_{\infty}[\text{prmp}] \sum U_i \). They also proved that the problem \( O_{\infty}[\text{prmp}] \sum U_i \) is NP-hard. Liu and Bultin [14] showed that both the problems \( O_{\infty}[\text{prmp}] \sum C_i \) and \( O_{\infty}[\text{prmp}, d_t] \sum C_i \) are NP-hard. Liu and Bultin [15] also

*Email: cfliaow@cyut.edu.tw
© 2013 Chinese Institute of Industrial Engineers
studied the preemptive ordered open shop and proposed polynomial time algorithms for special cases of this problem. Liaw [11] presented a tabu search algorithm for solving the problem $O_1|\text{prmp}| \sum T_i$. Liaw [12] also developed a dynamic programming-based algorithm for the problem $O_2|\text{prmp}| \sum w_i C_i$. Recently, Brassel and Hennes [1] presented disjunctive graph and block-matrices models for analyzing the problem $O_m|\text{prmp}| \sum C_i$.

Liu [13] developed a branch-and-bound algorithm for the problem $O_m|\text{prmp}| \sum C_i$. To the author's knowledge, this is the only branch-and-bound algorithm in the literature for the problem under consideration. However, computational result for this algorithm is presented only for the two-machine cases. In this paper, we improve both the upper and lower bounds proposed in Liu [13]. Furthermore, these bounds are incorporated into a branch-and-bound algorithm to solve the general $m$-machine problems. The rest of the paper is organized as follows. In Section 2, we present a mixed integer programming (MIP) formulation for the problem under consideration. In Section 3, an efficient upper bound which can be used as a heuristic for solving large-sized problem is proposed. A new lower bound is given in Section 4. In Section 5, a branch-and-bound algorithm that implicitly searches over all possible job completion sequences is described. Computational results are provided in Section 6 followed by some concluding remarks in Section 7.

2. A mixed integer programming formulation

Define $C_0$ to be zero and $x_{ijk}$ to be the amount of job $i$ processed on machine $j$ in the $k$th interval $[C_{k-1}, C_k)$, for $j=1,2,\ldots,m$; $i=1,2,\ldots,n$ and $k \geq 1$. By definition, $x_{ijk} = 0$ for all $j$ and $i < k$. Define $Z_{ik} = 1$ if job $i$ is in the $k$th position of the job completion sequence, and $Z_{ik} = 0$ otherwise. The $O_m|\text{prmp}| \sum C_i$ problem can be formulated as the following MIP problem.

**MIP:** min $\sum_{i=1}^{n} C_i$

s.t. $\sum_{j=1}^{m} x_{ijk} \leq C_k - C_{k-1}, \quad \forall j, k$ (1)

$\sum_{j=1}^{m} x_{ijk} \leq C_k - C_{k-1}, \quad \forall i, k$ (2)

$\sum_{k=1}^{n} x_{ijk} = p_{ij}, \quad \forall i, j$ (3)

$x_{ijk} + M \left( \sum_{l=1}^{k-1} Z_{il} - 1 \right) \leq 0, \quad \forall i,j,k$ (4)

$\sum_{i=1}^{n} Z_{il} = 1, \quad \forall i$ (5)

$\sum_{i=1}^{n} Z_{il} = 1, \quad \forall l$ (6)

$x_{ijk} \geq 0, \quad \forall i,j,k$ (7)

$Z_{il} = 0 \text{ or } 1, \quad \forall i,l$ (8)

Constraints (1) and (2) require that for each interval $[C_{k-1}, C_k)$, the amount of processing time assigned to each machine $j$ and each job $i$, respectively, be no more than the interval length $C_k - C_{k-1}$. Constraint (3) requires that each operation be completed. Constraint (4) guarantees that each job will not be assigned to any interval located after its completion time, where $M$ is a big number. Constraint (5) requires that each job take exactly one position in the job completion sequence, while constraint (6) requires each position in the job completion sequence take exactly one job. Finally, constraints (7) and (8) are the non-negativity and binary constraints.

If a job completion sequence is known in advance, that is, a feasible set of variables $\{Z_{il}\}$ is given, we can obtain the optimal job completion times for this sequence by solving a linear program. Without loss of generality, we assume that jobs are completed in the sequence $1,2,\ldots,n$, that is, $C_i \leq C_{i+1}$ for $i=1,2,\ldots,n-1$. The linear program that produces the optimal job completion times when jobs are completed in the sequence $1,2,\ldots,n$ is as follows:

$\text{LP1: Min } \sum_{k=1}^{n} C_k$

s.t. $\sum_{j=1}^{m} x_{ijk} \leq C_k - C_{k-1}, \quad \forall j, k$ (9)

$\sum_{j=1}^{m} x_{ijk} \leq C_k - C_{k-1}, \quad \forall k, i$ (10)

$\sum_{k=1}^{n} x_{ijk} = p_{ij}, \quad \forall j, i$ (11)

$C_i, x_{ijk} \geq 0, \quad \forall j, i, k \geq 1$ (12)

The solution to LP1 gives the completion time for each job and the amount of processing time of each operation $O_{ij}$ processed in each interval $[C_{k-1}, C_k)$, $k=1,2,\ldots,n$. With this information, we can apply the algorithm of Gonzalez and Sahni [6] to each of the $n$ intervals to obtain the corresponding schedule. As shown in Liu [15], the linear program LP1 simplifies to the
following $O(n)$ LTA when $m=2$. Given a partial job completion sequence with the first $k-1$ positions fixed, there are intervals during which machine 1 is idle but machine 2 is busy, and other intervals during which the reverse is true; these intervals have lengths $U_k$ and $V_k$, respectively.

2.1 Algorithm LTA

Step 1: Set $C_0 = U_1 = V_1 = 0$ and $k = 1$.
Step 2: While ($k < n$) do

$$C_k = C_{k-1} + \max(0, p_{k1} - U_k) + \max(0, p_{k2} - V_k)$$

$$U_{k+1} = \max(0, U_k - p_{k1}) + \max(0, p_{k2} - V_k)$$

$$V_{k+1} = \max(0, V_k - p_{k2}) + \max(0, p_{k1} - U_k)$$

To solve the problem, we must enumerate (at least implicitly) all possible job completion sequences. That is, we must examine $n!$ sequences in the worst case. One advantage of this search strategy is that the number of schedules examined is independent of the number of machines.

3. A new upper bound

Liu [13] presented an upper bound for the $O_m|p_{max}|\sum C_i$ problem. The upper bound, denoted by $STPT$, is obtained by first establishing a job completion sequence by the shortest total processing time rule. That is, job $i$ completes before job $k$ if and only if $\sum_{j=1}^{i} p_j \leq \sum_{j=1}^{k} p_j$. Then, the linear program $LP1$ is applied to determine the exact job completion times. In this section, we describe a new upper bound for the $O_m|p_{max}|\sum C_i$ problem. This upper bound, called $EPC$, consists of $n$ iterations. At iteration $q$, $q=1,2,...,n$, the job to be placed in the $q$th position in the job completion sequence is determined. Given a partial job completion sequence with the first $q-1$ positions fixed, let $EPC_{q-1}$ be the earliest possible completion time of job $i$ if it is completed next. Let $S'$ be the set of unscheduled jobs, $C_{q-1}$ be the completion time of the job in the $q$th position in the sequence, and $u_{jk}$ be the idle time of machine $j$ during the interval $[C_{q-1}, C_q]$ for $j=1,2,...,m$. The upper bound $EPC$ can be computed as follows.

Step 1: Let $q=1$, $C_0 = 0$, $S = \{1,2,...,n\}$ and $u_{ij} = 0$ for $j=1,2,...,m$.
Step 2: For each unscheduled job $i \in S$, compute its earliest possible completion time $EPC_{q-1} = C_{q-1} + \sum_{j=1}^{m} (p_{j1} - \sum_{k=1}^{q-1} x_{jk})$ where $\{x_{jk}\}$ is an optimal solution of the following linear program:

$$LP2: \text{Max} \sum_{j=1}^{m} \sum_{k=1}^{q-1} x_{jk}$$

s.t. $\sum_{j=1}^{m} x_{jk} \leq C_{k} - C_{k-1}$, $k = 1,2,...,q-1$. (13)

$$\sum_{k=1}^{q-1} x_{jk} \leq p_j, \quad j = 1,2,...,m.$$ (14)

$$0 \leq x_{jk} \leq u_{jk}, \quad j = 1,2,...,m; \quad k = 1,2,...,q-1.$$ (15)

Step 3: Select job $i^*$ to be scheduled next if $EPC_{q-1} = \min\{EPC_{q-1} : i \in S\}$.
Set $C_q = C_{q-1} + EPC_{q-1}$.

Step 4: Update

$$u_{jk} = u_{jk} - x_{jk}, \quad \forall j = 1,2,...,m; k = 1,2,...,q-1$$

$$u_{ij} = C_q - C_{q-1} - \left(p_{ij} - \sum_{k=1}^{q-1} x_{jk}\right), \quad \forall j = 1,2,...,m$$

Set $q = q + 1$ and $S = S \setminus \{i^*\}$.

Step 5: Repeat steps 2-4 until $S = \emptyset$.

In step 2, the value $EPC_{q-1}$ is determined by scheduling job $i$ so that the intervals $[C_{q-1}, C_q]$, $k = 1,2,...,q-1$, are utilized as much as possible, while keeping the feasibility constraints satisfied. That is, for each such interval on each machine, the amount of processing time scheduled should be no more than its interval length. This is exactly what the linear program $LP2$ accomplishes. In fact, the linear program $LP2$ can be viewed as a max-flow problem as shown in Figure 1.
In this max-flow problem, we have a node \( A_j \) for each machine \( j = 1, 2, \ldots, m \), and a node \( B_k \) for each interval \( [C_{(k-1)}, C_k] \), \( k = 1, 2, \ldots, q-1 \). Nodes \( s \) and \( t \) are the dummy source and sink, respectively. There exists an arc \((A_j, B_k)\) with capacity \( u_{jk} \) for each \( j = 1, 2, \ldots, m \) and \( k = 1, 2, \ldots, q-1 \). Also, there exists an arc \((s, A_j)\) with capacity \( p_j \) for each \( j = 1, 2, \ldots, m \). Finally, for each node \( B_k \), \( k = 1, 2, \ldots, q-1 \), there exists an arc \((B_k, t)\) with capacity \( C_{[k]} - C_{[k-1]} \). The preflow algorithm proposed by Karzanov [8] can be used to solve this max-flow problem.

4. A new lower bound

Two lower bounds are used in Liu's branch-and-bound algorithm [13] to obtain a lower bound on the total completion time of the jobs not included in a partial job completion sequence \( \sigma \). These lower bounds are referred to as \( LB1 \) and \( LB2 \). The first lower bound \( LB1 \) is obtained by relaxing the problem to an identical parallel machine preemptive scheduling problem \( P_m|prmp|\sum C_i \) with job processing time \( \bar{p}_i = \sum_{j=1}^{m} p_{ij} \). It is well known that this relaxed problem provides a lower bound to the problem \( O_m|prmp|\sum C_i \) and can be optimally solved by sequencing jobs in \( SPT \) order and then scheduling each job on the earliest available machine. Let \( C(\sigma) \) be the completion time of the last job in \( \sigma \) and \( \sigma' \) be the set of jobs not included in \( \sigma \). Assume that there are \( k \) jobs in \( \sigma \) and \( n-k \) jobs in \( \sigma' \). Let \( t_j \), \( j = 1, 2, \ldots, m \), be the earliest time at which machine \( j \) becomes available. Lower bound \( LB1 \) is computed as follows:

Step 1: Renumber the jobs in \( \sigma' \) so that \( \bar{p}_1 \leq \bar{p}_2 \leq \ldots \leq \bar{p}_{n-k} \).

Step 2: Let \( t_j = \sum_{i \in \sigma} p_{ij}, \ j = 1, 2, \ldots, m \).

Step 3: Set \( i = 1 \) and \( LB1 = 0 \).

Step 4: While \( (i \leq n-k) \)

- \( t_r = \min\{t_j : j = 1, 2, \ldots, m\} \).
- \( C_i = \max\{C(\sigma), t_r + \bar{p}_i\} \).
- \( t_r = t_r + \bar{p}_i \).
- \( LB1 = LB1 + C_i \).

We remark that there is a flaw in lower bound \( LB1 \) as it does not properly deal with the machine idle time generated by the given partial job completion sequence \( \sigma \). A revised version of \( LB1 \), referred to as \( LB1' \), corrects this error as given follows:

Step 1: Renumber the jobs in \( \sigma' \) so that \( \bar{p}_1 \leq \bar{p}_2 \leq \ldots \leq \bar{p}_{n-k} \).

Step 2: Let \( I = \sum_{j=1}^{m} (C(\sigma) - \sum_{i \in \sigma} p_{ij}) \) be the total machine idle time in the interval \([0, C(\sigma)]\).

Step 3: Let \( r \) be the smallest index such that \( \sum_{i=1}^{r} \bar{p}_i \geq 1 \).

Step 4: Set \( t_j = C(\sigma), j = 1, 2, \ldots, m \).

Step 5: Set \( i = 1 \) and \( LB1' = 0 \).

Step 6: While \( (i \leq n-k) \)

- \( t_r = \min\{t_j : j = 1, 2, \ldots, m\} \).
- If \( i < r \), \( C_i = C(\sigma) \).
- Else if \( i = r \), \( C_i = C(\sigma) + \sum_{j=1}^{r} \bar{p}_j - 1 \).
- Else \( C_i = t_r + \bar{p}_i \).
- \( t_r = t_r + \bar{p}_i \).
- \( LB1' = LB1' + C_i \).

Example 1. Consider a \( O_m|prmp|\sum C_i \) problem with \( n = 4 \) and \( m = 2 \). The processing time matrix \([p_{ij}]\) is given by \[
\begin{bmatrix}
14 & 13 \\
12 & 14 \\
9 & 17 \\
22 & 44
\end{bmatrix}
\]

Consider a partial job completion sequence \( \sigma = \{2\} \) with \( C(\sigma) = p_{21} + p_{22} = 26 \). The corresponding partial schedule is given in Figure 2. The original lower bound \( LB1 \) for the unscheduled jobs in \( \sigma' \) is computed as follows:

Step 1: Renumber the jobs in \( \sigma' \) to get \( \bar{p}_1 = 26 \leq \bar{p}_2 = 27 \leq \bar{p}_3 = 66 \).

Step 2: \( t_1 = 12 \) and \( t_2 = 14 \).

Step 3: \( i = 1 \) and \( LB1 = 0 \).

Step 4:

- \( i = 1; f^* = 1; \)
- \( C_1 = \max\{26, 12 + 26\} = 38; t_1 = 12 + 26 = 38; \)
- \( LB1 = 0 + 38 = 38. \)
- \( i = 2; f^* = 2; \)
- \( C_2 = \max\{26, 14 + 27\} = 41; t_2 = 14 + 27 = 41; \)
- \( LB1 = 38 + 41 = 79. \)
- \( i = 3; f^* = 1; \)
- \( C_3 = \max\{26, 38 + 66\} = 104; t_1 = 38 + 66 = 104; \)
- \( LB1 = 79 + 104 = 183. \)

If the new lower bound \( LB1' \) is applied, we have:

![Figure 2. Partial schedule corresponding to \( \sigma = \{2\} \).](image-url)
Step 1: Renumber the jobs in \( \sigma' \) to get 
\( p_1 = 26 \leq p_2 = 27 \leq p_3 = 66. \)

Step 2: \( I = (26-12) + (26-14) = 26. \)

Step 3: Since \( \bar{p}_i = 26 \geq I \), we have \( r = 1. \)

Step 4: \( t_1 = t_2 = 26. \)

Step 5: \( i = 1 \) and \( LBI_{\text{revised}} = 0. \)

Step 6: \( i = 1; \quad j^* = 1; \)
\( C_1 = 26 + 26 - 26 = 26; \quad t_1 = 26; \)
\( LBI_{\text{revised}} = 0 + 26 = 26. \)

\( i = 2; \quad j^* = 1; \)
\( C_2 = 26 + 27 = 53; \quad t_2 = 53; \)
\( LBI_{\text{revised}} = 26 + 53 = 79. \)

\( i = 3; \quad j^* = 2 \)
\( C_3 = 26 + 66 = 92; \quad t_2 = 92; \)
\( LBI_{\text{revised}} = 79 + 92 = 171. \)

A feasible job completion sequence in line with the partial sequence \( \sigma = (2) \) in Example 1 is \( \{2, 3, 1, 4\} \). A corresponding schedule is given in Figure 3. As shown in Figure 3, the sum of completion times of unscheduled jobs \( \{3, 1, \) and \( 4\} \) is \( C_3 + C_1 + C_4 = 31 + 48 + 97 = 176 \), which is less than \( LBI = 183 \), and hence verifies the incorrectness of \( LBI \). On the other hand, the revised lower bound \( LBI_{\text{revised}} \) fills in all the machine idle time with shortest unscheduled jobs before solving the relaxed identical parallel machine preemptive scheduling problem \( P_{\text{mip}} | \text{mpm} | \sum C_i \), and hence generates a valid lower bound.

The second lower bound \( LBI_2 \) introduced by Liu [13] is obtained by deriving a lower bound of a new problem \( P' \) with processing time \( p_{ij} = p_{ij} \), where \( p_{ij} \) is the ith smallest processing time on machine \( j \). By allowing each job to be simultaneously processed on more than one machine, a lower bound on the completion time of the \( i \)th job in a job completion sequence is \( \max \{M_{ij} : j = 1, 2, \ldots, m\} \), where \( M_{ij} = \sum_{k=1}^i p_{jk} \). Hence, a lower bound of problem \( P' \) can be computed as \( LBI_2 = \sum_{j=1}^n \max \{M_{ij} : j = 1, 2, \ldots, m\} \). Note that for problem \( P' \), we have \( p_{ij} \leq \bar{p}_{i-1,j} \) for \( i = 1, 2, \ldots, n-1 \) and \( j = 1, 2, \ldots, m \). Jobs with such property are known as ordered jobs in Liu and Bulfin [15]. It has been shown by Liu and Bulfin [15] that problem \( P' \) can be optimally solved by completing jobs in the sequence \( 1, 2, \ldots, n \). To further improve the quality of \( LBI_2 \), we present in this paper a new lower bound \( LBI_{\text{revised}} \) obtained by applying the linear program \( LP_1 \) to solve problem \( P' \) with the job completion sequence \( 1, 2, \ldots, n \). Obviously, lower bound \( LBI_{\text{revised}} \) dominates lower bound \( LBI_2 \) since the former optimally solves problem \( P' \) while the later finds a lower bound of problem \( P' \).

Consider Example 1 again with a partial job completion sequence \( \sigma = (2) \) and \( C(\sigma) = p_{21} + p_{22} = 26 \). The processing time matrix \( [p_{ij}'] \) for problem \( P' \) is

\[
\begin{bmatrix}
9 & 13 \\
14 & 17 \\
22 & 44
\end{bmatrix}
\]

We remark that given a partial job completion sequence \( \sigma \), the definition of \( M_{ij} \) should be modified as \( M_{ij} = \sum_{k=1}^{i} p_{ik} + \sum_{k=1}^{j} p_{kj} \). Thus, we have \( LBI_2 = \max \{12 + 9, 14 + 13\} + \max \{21 + 14, 27 + 17\} + \max \{35 + 22, 44 + 44\} = 27 + 44 + 88 = 159 \). To compute \( LBI_{\text{revised}} \), we can solve problem \( P' \) using linear program \( LP_1 \) given in Section 2. However, as mentioned previously in the case \( m = 2 \), the \( LT_1 \) can be applied. The computation of \( LBI_{\text{revised}} \) is as follows:

Step 1: Set \( C_0 = C(\sigma) = 26 \), \( U_1 = p_{22} = 14 \), \( V_1 = p_{21} = 12 \)

Step 2:

\( i = 1; \)
\( C_1 = 26 + \max (0, 9-14) + \max (0, 13-12) = 27 \)
\( U_2 = \max (0, 14-9) + \max (0, 13-12) = 6 \)
\( V_2 = \max (0, 12-13) + \max (0, 0-14) = 0 \)

\( i = 2; \)
\( C_2 = 27 + \max (0, 14-6) + \max (0, 17-0) = 52 \)
\( U_3 = \max (0, 6-14) + \max (0, 17-0) = 17 \)
\( V_3 = \max (0, 0-17) + \max (0, 14-6) = 8 \)

\( i = 3; \)
\( C_3 = 52 + \max (0, 22-17) + \max (0, 44-8) = 93 \)
\( U_4 = \max (0, 17-22) + \max (0, 44-8) = 36 \)
\( V_4 = \max (0, 8-44) + \max (0, 22-17) = 5 \)

Thus,
\( LBI_{\text{revised}} = C_1 + C_2 + C_3 = 27 + 52 + 93 = 172. \)

Finally, the lower bound \( LB = \max \{LBI_{\text{revised}}, LBI_{\text{revised}}\} \) is used in the branch-and-bound algorithm presented in the next section.

5. A branch-and-bound algorithm

In this section, a branch-and-bound algorithm that implicitly examines all candidate job completion sequences is presented. The algorithm adopts a forward
branching strategy, where a node at level \( q-1 \) in
the search tree corresponds to a partial job completion
sequence with \( q-1 \) jobs fixed in the first \( q-1 \) positions.
For each node at level \( q-1 \) in the search tree, we
generate \( n-q+1 \) descendents, each corresponding to an
unscheduled job fixed in the \( q \)th position of the job completion
sequence.

Given a node at level \( q \) in the search tree, the linear
program \( LP(q) \) presented in Section 2 is applied to opti-
mally schedule the \( q \) jobs in the partial job completion
sequence. For the other \( n-q \) unscheduled jobs, the lower
bound \( LB \) given in Section 4 is computed. If the lower
bound \( LB \) for the unscheduled jobs plus the minimum
sum of job completion times of the scheduled jobs is
greater than or equal to the current upper bound, the
node is discarded.

Initially, the upper bound \( EPC \) is computed. The
upper bound is updated whenever a complete schedule
that improves the upper bound is found during the search
process. The depth-first strategy is used to choose the
next candidate node. Ties are broken by choosing the
node with the smallest cost (sum of the job completion
times of the scheduled jobs and the lower bound \( LB \)
corresponding to unscheduled jobs). The storage required
by this strategy is only linear in the number of jobs.

We remark that the following dominance rule given
by Liu [13] that can be used to eliminate unpromising
nodes in the search tree is also incorporated into our
branch-and-bound algorithm. This rule can be proved by
the classical pairwise interchange arguments.

**Dominance rule.** If \( p_j < p_k \) for all \( j = 1, 2, \ldots, m \),
then job \( i \) precedes job \( k \) in at least one optimal job
completion sequence.

6. Computational experiments

In this section, we describe our experiments to evaluate
the performance of the proposed branch-and-bound algo-
rithm. The algorithm is coded in C and implemented on
an PC with i7-2600 CPU and 8.0 GB RAM. We generate
randomly two types of problem instances: \( n > m \) and
\( n = m \). For \( n > m \) cases, we generate problem instances
with \( m = 2, 3, 4, 5 \) and \( n = 6, 8, 10, 12, 14 \). For \( n = m \) cases,
we generate problem instances with \( n = m = 6, 8, 10, 12, 14 \).
The processing times for all operations are generated
independently from the uniform distribution \( U[1, 100] \).
Thirty instances are generated for each combination of
\( n \) and \( m \), yielding a total number of 1350 problem
instances. To prevent excessive computation time, when-
ever a problem instance is not solved within the time
limit of one hour, computation is stopped for that prob-
lem instance. Tables 1–4 present the detailed results for
the branch-and-bound algorithm and the performance
evaluation for the proposed upper and lower bounds.
The information shown in these tables is as follows:

\[ n: \] Number of jobs.
\[ m: \] Number of machines.
\[ ND_{avg}: \] The average number of nodes generated.
\[ TM_{avg}, TM_{max}: \] The average (maximum) CPU time
in seconds.
\[ UNSOL: \] The number of problem instances unsolved (out
of 30) within the time limit of one hour. \( LB_{avg}(\%) \)
(\( LB_{max}(\%) \)): The average (maximum) percentage devia-
tion of the initial lower bound \( LB \). For each instance, the
percentage deviation of the initial lower bound \( LB \)
is defined by \( 100 \times (OPT-LB)/OPT \), where \( OPT \) is the opti-
mal value.

<table>
<thead>
<tr>
<th></th>
<th>( n )</th>
<th>( ND_{avg} )</th>
<th>( TM_{avg} )</th>
<th>( TM_{max} )</th>
<th>( UNSOL )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
<td>4</td>
<td>0.00</td>
<td>0.02</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>24</td>
<td>0.02</td>
<td>0.05</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>70</td>
<td>0.13</td>
<td>1.11</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>462</td>
<td>1.69</td>
<td>18.92</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>4383</td>
<td>37.68</td>
<td>276.67</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>21</td>
<td>0.02</td>
<td>0.09</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>67</td>
<td>0.14</td>
<td>1.25</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1259</td>
<td>8.02</td>
<td>51.29</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>4974</td>
<td>67.92</td>
<td>774.19</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>16,915</td>
<td>387.14</td>
<td>2432.17</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>11</td>
<td>0.02</td>
<td>0.14</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>65</td>
<td>0.30</td>
<td>4.21</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>827</td>
<td>10.26</td>
<td>115.85</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>7284</td>
<td>230.98</td>
<td>1592.88</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>8352</td>
<td>599.92</td>
<td>2897.49</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
<td>0.01</td>
<td>0.05</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>5</td>
<td>0.12</td>
<td>1.01</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>296</td>
<td>6.91</td>
<td>118.89</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>4494</td>
<td>215.50</td>
<td>2985.67</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>5592</td>
<td>438.64</td>
<td>2211.74</td>
<td>11</td>
</tr>
</tbody>
</table>
Table 2. Performance evaluation of the upper and lower bounds for the $n=m$ cases.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>$LB_{avg}(%)$</th>
<th>$LB_{max}(%)$</th>
<th>$UB_{avg}(%)$</th>
<th>$UB_{max}(%)$</th>
<th>$^*$LB(%)</th>
<th>$^*$UB(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
<td>3.13</td>
<td>16.31</td>
<td>0.00</td>
<td>0.00</td>
<td>13.33</td>
<td>100.00</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>3.37</td>
<td>15.63</td>
<td>0.00</td>
<td>0.10</td>
<td>3.33</td>
<td>96.67</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2.36</td>
<td>10.66</td>
<td>0.05</td>
<td>0.77</td>
<td>0.00</td>
<td>83.33</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>2.94</td>
<td>8.58</td>
<td>0.09</td>
<td>0.74</td>
<td>0.00</td>
<td>70.00</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>2.74</td>
<td>14.61</td>
<td>0.04</td>
<td>0.41</td>
<td>0.00</td>
<td>73.33</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>5.64</td>
<td>26.58</td>
<td>0.03</td>
<td>0.42</td>
<td>3.33</td>
<td>86.67</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1.96</td>
<td>7.49</td>
<td>0.16</td>
<td>1.34</td>
<td>23.33</td>
<td>70.00</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>3.00</td>
<td>13.51</td>
<td>0.18</td>
<td>0.99</td>
<td>13.33</td>
<td>66.67</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>3.36</td>
<td>11.85</td>
<td>0.17</td>
<td>0.57</td>
<td>3.33</td>
<td>50.00</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>2.78</td>
<td>8.02</td>
<td>0.10</td>
<td>0.83</td>
<td>4.17</td>
<td>45.83</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>0.25</td>
<td>2.70</td>
<td>0.02</td>
<td>0.65</td>
<td>83.33</td>
<td>96.67</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2.82</td>
<td>13.32</td>
<td>0.12</td>
<td>0.84</td>
<td>13.33</td>
<td>76.67</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1.40</td>
<td>6.53</td>
<td>0.10</td>
<td>0.78</td>
<td>33.33</td>
<td>70.00</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>2.46</td>
<td>8.57</td>
<td>0.18</td>
<td>0.65</td>
<td>10.34</td>
<td>48.28</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>1.15</td>
<td>9.38</td>
<td>0.33</td>
<td>2.24</td>
<td>23.81</td>
<td>23.81</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>0.29</td>
<td>4.81</td>
<td>0.00</td>
<td>0.00</td>
<td>90.00</td>
<td>100.00</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.02</td>
<td>0.25</td>
<td>0.05</td>
<td>0.73</td>
<td>93.33</td>
<td>93.33</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1.83</td>
<td>5.27</td>
<td>0.14</td>
<td>0.96</td>
<td>6.67</td>
<td>63.33</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>0.42</td>
<td>2.44</td>
<td>0.26</td>
<td>3.95</td>
<td>41.38</td>
<td>62.07</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>0.44</td>
<td>3.64</td>
<td>0.74</td>
<td>9.64</td>
<td>15.79</td>
<td>42.11</td>
</tr>
<tr>
<td>Overall avg.</td>
<td></td>
<td>2.12</td>
<td>26.58</td>
<td>0.14</td>
<td>9.64</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Overall max. |     | 26.58 | 9.64 |

Table 3. Results of the branch-and-bound algorithm for the $n=m$ cases.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>$ND_{avg}$</th>
<th>$TM_{avg}$</th>
<th>$TM_{min}$</th>
<th>UNSOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>0.02</td>
<td>0.05</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>2</td>
<td>0.19</td>
<td>0.42</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>74</td>
<td>7.22</td>
<td>92.07</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>144</td>
<td>22.38</td>
<td>218.82</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>108</td>
<td>101.47</td>
<td>1187.84</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4. Performance evaluation of the upper and lower bounds for the $n=m$ cases.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>$LB_{avg}(%)$</th>
<th>$LB_{max}(%)$</th>
<th>$UB_{avg}(%)$</th>
<th>$UB_{max}(%)$</th>
<th>$^*$LB(%)</th>
<th>$^*$UB(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>0.05</td>
<td>1.31</td>
<td>0.00</td>
<td>0.00</td>
<td>93.33</td>
<td>100.00</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>0.01</td>
<td>0.32</td>
<td>0.00</td>
<td>0.12</td>
<td>93.33</td>
<td>96.67</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0.01</td>
<td>0.19</td>
<td>0.02</td>
<td>0.36</td>
<td>93.33</td>
<td>93.33</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>0.08</td>
<td>1.12</td>
<td>0.00</td>
<td>0.10</td>
<td>86.21</td>
<td>96.55</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>0.04</td>
<td>1.10</td>
<td>0.01</td>
<td>0.35</td>
<td>76.92</td>
<td>96.15</td>
</tr>
<tr>
<td>Overall avg.</td>
<td></td>
<td>0.04</td>
<td>1.31</td>
<td>0.01</td>
<td>0.36</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As shown in Table 1, for a given number of machines $m$, the average number of nodes generated and CPU time increases as the number of jobs $n$ increases. However, for a fixed number of jobs $n$, the number of machines $m$ does not have a significant effect on the performance of the branch-and-bound algorithm. This is because the proposed branch-and-bound algorithm examines (implicitly) only candidate job completion sequences in searching for an optimal schedule, and the number of job completion sequences examined ($n!$ in the worst case) is independent of the number of machines $m$. The

$UB_{avg}(\%)$ ($UB_{max}(\%)$): The average (maximum) percentage deviation of the initial upper bound $UB$ (obtained by heuristic EPC). For each instance, the percentage deviation of the initial upper bound $UB$ is defined by 100× $(UB-OPT)/OPT$.

$^*$LB(%): The percentage ratio of the number of instances for which the initial $LB$ equals the optimal value over the total number of instances completely solved.

$^*$UB(%): The percentage ratio of the number of instances for which the initial $UB$ equals the optimal value over the total number of instances completely solved.
number of problem instances unsolved (out of 30) within the time limit of one hour, gradually increases as the problem size increases. For the largest problem size \( n = 14 \) and \( m = 5 \), 11 problem instances are not solved within the time limit, whereas the average CPU time for the other 19 solved problem instances is 438.64 s. Note that all the numbers given in Tables 1–4 do not include consideration of the abandoned problem instances.

In Table 2, we present the quality evaluation of the proposed upper and lower bounds for the \( n > m \) cases. It can be seen that both the number of machines \( m \) and the number of jobs \( n \) do not have a significant effect on the quality of the proposed lower bound \( LB \). The overall average and maximum percentage deviation of the initial lower bound \( LB \) is 2.12 and 26.58%. The overall percentage ratio of the number of instances for which the initial \( LB \) equals the optimal value over the total number of instances completely solved is 23.77%. However, the quality of the proposed \( UB \) deteriorates slightly as the number of jobs \( n \) or machines \( m \) increases. The overall average and maximum percentage deviation of the initial upper bound \( UB \) is 0.14 and 9.64%. The overall percentage ratio of the number of instances for which the initial \( UB \) equals the optimal value over the total number of instances completely solved is 70.94%.

By comparing Table 1 with Table 3, we observe that problems with \( n = m \) are easier to solve than problems with \( n > m \). The reason is that, as shown in Table 2 and 4, both the proposed upper and lower bounds are much tighter in the case \( n = m \) than in the case \( n > m \). The overall average percentage deviation of the initial \( LB \) and \( UB \) (\( LB_{\text{avg}}(\%) \) and \( UB_{\text{avg}}(\%) \)) for the case \( n > m \) is 2.12 and 0.14%, respectively, while \( LB_{\text{avg}}(\%) \) and \( UB_{\text{avg}}(\%) \) for the case \( n = m \) is 0.04 and 0.01%, respectively. Also, the percentage ratio of the number of instances for which the initial \( LB \) and \( UB \) equals the optimal value over the total number of instances completely solved (\( LB_{\text{opt}}(\%) \) and \( UB_{\text{opt}}(\%) \)) is 23.77 and 70.94%, respectively, while \( LB \) (\%) and \( UB \) (\%) for the case \( n = m \) is 88.62 and 96.54%, respectively.

To the best of the author's knowledge, the only exact solution method in literature for the problem under consideration is the enumeration method presented by Brasel and Hennes [1]. As indicated in [1], the enumeration method can only solve problems with size up to \( n = m = 7 \) within a reasonable amount of time. The branch-and-bound algorithm proposed in this paper can easily solve problems with size \( n = m = 14 \). The average CPU time for solving a problem with size \( n = m = 14 \) is 101.47 s (as shown in Table 3). In terms of computational effectiveness, the algorithm proposed in this paper significantly outperforms the existing enumeration algorithm.

Finally, we remark that the heuristic EPC proposed in Section 3 performs extremely well. On average, its percentage deviation from the optimal value is 0.14% for the \( n > m \) cases and 0.01% for the \( n = m \) cases. Moreover, it finds approved optimal solutions for over 70% of the instances completely solved in the \( n > m \) cases, and over 95% of the instances completely solved in the \( n = m \) cases. Hence, heuristic EPC can also be used alone as a solution method to efficiently solve large sized problems in practice.

7. Conclusions

In this article, the problem of scheduling preemptive open shops to minimize total completion time is examined. A branch-and-bound algorithm that searches over \( n! \) job completion sequences regardless of the number of machines is presented. Improved lower and upper bounds are developed to curtail the search space. The upper bound can be used as a heuristic to efficiently solve large sized problems while the branch-and-bound algorithm can be used to optimally solve medium sized problems. To the authors' knowledge, this is the first implementation of branch-and-bound scheme to solve general \( m \)-machine case for the problem under consideration. The proposed algorithm can also be used as a basis for developing branch-and-bound algorithms for the preemptive open shop scheduling problem with other performance measures such as total tardiness and total number of tardy jobs.

Computational results for randomly generated problem instances are reported. For the \( n > m \) cases, the branch-and-bound algorithm can handle problems of up to \( n = 14 \) and \( m = 5 \) in size within a reasonable amount of time. For the \( n = m \) cases, the branch-and-bound algorithm can handle problems of up to \( n = m = 14 \) in size within a reasonable amount of time. The average percentage deviation of the proposed lower bound is 2.12% for the case \( n > m \) and 0.04% for the case \( n = m \). The average percentage deviation of the proposed upper bound is 0.14% for the case \( n > m \) and 0.01% for the case \( n = m \). Furthermore, the proposed upper bound is equal to the optimal value for over 70% of the problem instances completely solved in the case \( n > m \), and over 95% of the problem instances completely solved in the case \( n = m \).

For future research, it is desirable to further improve the performance of the proposed branch-and-bound algorithm by deriving various dominance rules, and extend the approach to more general problems with release times, job weights, etc.

Notes on contributors

Ching-Fang Liaw is currently a professor in the Department of Industrial Engineering and Management at Chaoyang University of Technology, Taiwan. He received the Bachelor's degree in Industrial Management from the Cheng Kung University, Taiwan, in 1985, and the PhD degree in Industrial and Operation Engineering from the University of Michigan, USA in 1994. His present research interests include production scheduling, meta-heuristics, and optimization.
References