Available online at www.sciencedirect.com

SCIENCE DIRECT®

European Journal of Operational Research 174 (2006) 1–10

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

Discrete Optimization

A branch and bound algorithm to minimize the total tardiness for *m*-machine permutation flowshop problems

Chia-Shin Chung [a], James Flynn [a,*], Ömer Kirca [b]

[a] *Department of Operations Management and Business Statistics, Cleveland State University, 2121 Euclid Avenue BU 539, Cleveland, OH 44115-2214, United States*

[b] *Department of Industrial Engineering, Middle East Technical University, Ankara 06531, Turkey*

Received 3 January 2002; accepted 30 December 2004
Available online 14 April 2005

Abstract

The *m*-machine permutation flowshop problem with the total tardiness objective is a common scheduling problem, which is known to be NP-hard. Here, we develop a branch and bound algorithm to solve this problem. Our algorithm incorporates a machine-based lower bound and a dominance test for pruning nodes. We undertake a numerical study that evaluates our algorithm and compares it with the best alternative existing algorithm. Extensive computational experiments indicate that our algorithm performs better and can handle test problems with $n \leqslant 20$.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Scheduling; Permutation flowshop; Tardiness; Branch and bound

1. Introduction

In the *permutation flowshop problem*, each of n jobs has to be processed on machines $1,\ldots,m$ in that order. The processing times of each job on each machine are known. At any time, each machine can process at most one job and each job can be processed on at most one machine. Once the processing of a job on a machine has started, it must be completed without interruption. Also, each job must be processed in the same order at every machine. The usual objectives are the minimization of the make-span, flow-time, tardiness, lateness, and the number of jobs late (see Pinedo [13] for a review of the general flowshop problem). Here, the objective is to minimize the total tardiness. Tardiness equals the amount by which a job's completion time exceeds its due date.

* Corresponding author. Tel.: +1 216 687 4741/621 6349; fax: +1 216 687 9343.
E-mail address: j.flynn@csuohio.edu (J. Flynn).

0377-2217/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2004.12.023

Schedules where each job must be processed in the same order at every machine are called *permutation schedules*. When $m \leqslant 2$, the restriction to such schedules is harmless; however, when $m > 3$, there may exist a general schedule whose total tardiness is strictly less than the total tardiness of any permutation schedule (see [13]). Finding such a schedule is often computationally impractical; moreover, as discussed in Kim [8] there are many real situations where only permutation schedules are feasible. Most approaches to the m machine flowshop problem restrict attention to permutation schedules.

The general m machine flow time problem with the total tardiness time objective is NP-hard in the ordinary sense for $m = 1$ and NP-hard in the strong sense for $m \geqslant 2$ (the special case where the due dates are equal to 0 is NP-hard in the strong sense for $m \geqslant 2$): see Appendix D of [13]. Consequently, there has been great interest in developing heuristic solutions [5–7,16,22]. See [10] for more details on heuristic approaches and other general issues.

Most optimal algorithms for single machine problems combine dynamic programming or branch and bound with decomposition properties developed by Lawler [12], Potts and Wassenhove [15] and Szwarc [18]. Harikawa [4] proposes an effective algorithm, based on Lawler's decomposition theorem. Szwarc and Mukhopadhyay [20] and Tansel [21] develop branch and bound algorithms incorporating decomposition properties, which can handle problem with over 100 jobs. Recently, Szwarc et al. [19] employ an improved decomposition rule, which allows them to solve problems with 300 jobs.

Only a few articles deal with optimal algorithms for multiple machine problems. Kim [7] and Sen et al. [17] apply branch and bound to two machine problems. Kim [8] applies branch and bound algorithm to general m machine problems using a special branching scheme, which he calls "backward branching". In his scheme, a node corresponds to a partial sequence of jobs placed at the end of a whole job sequence. When a node branches, one or more nodes are defined by placing one more job in front of the partial sequence associated with the current branch node. After characterizing properties of nodes, Kim derives lower bounds for the branch and bound algorithm together with a dominance rule. The dominance rule leads to a problem size reduction procedure, which sometimes yields a problem with a smaller n.

In Section 2, we briefly describe the problem and derive a new machine based lower bound. Our bound employs estimates of the earliest start time for jobs assigned to each position in the job sequence. (These estimates were derived in Chung et al. [1], which dealt with the flow-time objective.) Section 3 presents a dominance test, which helps to prune nodes. An outline of the algorithm appears in Section 4. Section 5 reports on a numerical study that evaluates our algorithm and compares it with Kim [8]—the best alternative existing algorithm. Extensive computational experiments involving over 40,000 test problems suggest that our algorithm performs better and can handle problems with $n \leqslant 20$. Section 6 states our conclusions.

2. The problem description and a lower bound

Branch and bound algorithms are commonly applied to the permutation flowshop problem, which is known to be NP hard. The effectiveness of a branch and bound algorithm is heavily influenced by its lower bounds. Such bounds are more useful when they are tighter and easier to compute. In this section, we develop a useful machine-based lower bound.

Consider the search tree for our algorithm [11]. The root node $\varnothing$ represents the null schedule. Every other node represents a *partial schedule* $\sigma = (\sigma(1), \ldots, \sigma(s))$, indicating that job $\sigma(j)$ occupies the jth position on each machine, for $1 \leqslant j \leqslant s$, where $1 \leqslant s \leqslant n$. Any permutation $\bar{\sigma}$ of the set of unscheduled jobs defines a complete schedule $\sigma\bar{\sigma} = (\sigma(1), \ldots, \sigma(s), \bar{\sigma}(1), \ldots, \bar{\sigma}(n-s))$. By placing any unscheduled job i in position $(s+1)$, we produce a descendant node $\sigma i = (\sigma(1), \ldots, \sigma(s), i)$. This paper employs the following additional notation:

x^+	$\max\{x, 0\}$, for any real number x
N	$\{1, \ldots, n\}$, the set of all jobs

p_{jk}	processing time of job j on machine k
d_j	due date of job j
$G(\sigma)$	total tardiness for jobs in the partial schedule σ
s	the number of jobs in the partial schedule σ
U	the set of jobs that are not included in the partial schedule σ
$L_k(\sigma)$	completion time of the last job in σ on machine k
q_{juv}	$\begin{cases} \sum_{r=u}^{v} p_{jr}, & \text{for } 1 \leqslant j \leqslant n \text{ and } 1 \leqslant u \leqslant v \leqslant m, \\ 0 & \text{otherwise} \end{cases}$
h_{jk}	$d_j - q_{jkm}, j = 1,\ldots,n; k = 1,\ldots,m$
$h_{[j]k}$	the jth smallest value of $\{h_{ik} : i \in U\}, j = 1,\ldots,n-s; k = 1,\ldots,m$
$d_{[j]}$	the jth smallest value of $\{d_i : i \in U\}, j = 1,\ldots,n-s$
$p_{[j]k}$	the jth smallest value of $\{p_{ik} : i \in U\}, j = j = 1,\ldots,n-s; k = 1,\ldots,m$
$q_{[1],r,k-1}$	the smallest value of $\{q_{j,r,k-1} : j \in U\}$, $1 \leqslant r \leqslant k-1; k = 2,\ldots,m.$

(For simplicity, our notation suppresses the dependence of the last four objects on σ.)

Turn to our lower bound on the tardiness at node σ. Temporarily assume machine k has unit capacity and the others have infinite capacity. Chung et al. [1] find that for $t \geqslant s+1$, the earliest start time of the tth job on machine k is bounded below by E_{tk} where E_{tk} satisfies the recursive equations below:

Step 1. Set $E_{s+1,1} = L_1(\sigma)$ and $R_{s,1} = L_1(\sigma)$,

$$R_{t1} = R_{s,1} + \sum_{r=1}^{t-s} p_{[r]1}, \quad t = s+1, \ldots, n,$$

$$E_{t1} = R_{t-1,1}, \quad t = s+2, \ldots, n.$$

Step 2. For $k = 2,\ldots,m$

$$E_{s+1,k} = \max\left\{L_k(\sigma), \max_{1rk-1}\{E_{s+1,r} + q_{[1],r,k-1}\}\right\}, \tag{1}$$

$$R_{sk} = E_{s+1,k}, \tag{2}$$

$$R_{tk} = R_{s,k} + \sum_{r=1}^{t-s} p_{[r]k}, \quad t = s+1, \ldots, n, \tag{3}$$

$$E_{tk} = \max\{R_{t-1,k}, R_{t,k-1}\}, \quad t = s+2, \ldots, n. \tag{4}$$

(For $t \geqslant s+1$, R_{tk} is a lower bound on the earliest completion time of the tth job on machine k.) The theorem below presents our lower bound on the tardiness at a node. (Chu [2] obtains a similar lower bound for the case where $m = 1$. Neither bound dominates the other bound.)

Theorem 2.1. *The tardiness at node σ is bounded below by g, where*

$$g = \max_k g_k + G(\sigma), \tag{5}$$

$$g_k = \sum_{t=s+1}^{n} T_{tk}, \tag{6}$$

$$T_{tk} = (E_{tk} - h_{[t-s]k})^+. \tag{7}$$

Proof. Let $1 \leqslant k \leqslant m$ and $s+1 \leqslant t \leqslant n$. Also let $(\rho(s+1), \rho(s+2), \ldots, \rho(n))$ be a permutation of the integers in U. Clearly E_{tk} is a lower bound on the start time of the tth job on machine k, while $(E_{tk} + q_{\rho(t),k,m} - d_{\rho(t)})^+ = (E_{tk} - h_{\rho(t)k})^+$ is a lower bound on the tardiness of the tth job when the tth job is $\rho(t)$. Using the fact that E_{tk} and $h_{[t-s]k}$ are nondecreasing in t, one can easily show that $\sum_{t=s+1}^{n}(E_{tk} - h_{[t-s]k})^+$ minimizes $\sum_{t=s+1}^{n}(E_{tk} - h_{\rho(t)k})^+$ among all permutations $(\rho(s+1), \rho(s+2), \ldots, \rho(n))$ of the integers in U. The basic idea is that any permutation with $h_{\rho(i),k+1} > h_{\rho(i+1),k+1}$, for some i, can be improved upon by interchanging $\rho(i)$ and $\rho(i+1)$. Hence $g_k = \sum_{t=s+1}^{n}(E_{tk} - h_{[t-s]k})^+$ is a lower bound on the tardiness of all jobs in U, implying that g is a lower bound on the total tardiness. $\square$

Our implementation of the branch and bound algorithm reduces computation by performing preliminary calculations of certain quantities that never change and storing the results in tables for ready access. Given such tables, the calculations of $p_{[1]k}$ through $p_{[n-s]k}$ and $h_{[1]k}$ through $h_{[n-s]k}$ for fixed k and $q_{[1],r,k-1}$ for fixed r and k require $O(n)$ calculations. Hence, one can evaluate g for a given node σ using $O(m^2n)$ calculations. The details are spelled out in Remark 2.1 of [1].

Before moving on to Section 3, we compare our lower bound g with the lower bound of Kim [8]. Kim's tree structure and node definition, which stem from his "backward branching" approach, differs considerably from ours, which follows the conventional approach of Lageweg et al. [11]. Kim's lower bound for a node σ in his search tree is the sum of a lower bound on the tardiness of jobs in σ and a lower bound on the tardiness of jobs not in σ, while our lower bound for a node σ in our search tree is the sum of the exact value of the tardiness of jobs in σ (i.e., $G(\sigma)$) and a lower bound on the tardiness of jobs not in σ (i.e., $\max_k g_k$). (It is not possible to compute the exact value of the tardiness of jobs in σ for Kim's search tree.) Our use of exact values may partially account for the superior performance of our algorithm in the numerical study of Section 5.

3. Dominance criteria

Here, we define a dominance relation between partial schedules, which can help us to prune nodes. Specifically, let σ_1 and σ_2 be two partial schedules for the same set of jobs S. We say that σ_1 dominates σ_2, if $G(\sigma_1\sigma) \leqslant G(\sigma_2\sigma)$ for every permutation σ of the jobs in $N - S$. If the previous inequality is strict, we say that σ_1 strictly dominates σ_2. Extending results in Section 3 of [1], the next theorem and its corollary develop conditions for one partial schedule to dominate another.

Theorem 3.1 (Node dominance). *Let σ_1 and σ_2 be two partial schedules for the same set of jobs S, and let $s = |S|$. If*

$$G(\sigma_2) - G(\sigma_1) \geqslant \max\left(0, (n-s)\max_{1\leqslant k\leqslant m}\{L_k(\sigma_1) - L_k(\sigma_2)\}\right), \tag{8}$$

then σ_1 dominates σ_2. Moreover, dominance is strict if inequality (8) is strict.

The proof of Theorem 3.1 requires the following lemma which is stated without proof.

Lemma 3.1. Max$(0, a-b) \geqslant \max(0, a-c) - \max(0, b-c)$, *for all real numbers a, b, and c.*

Proof of Theorem 3.1. Let $\pi = (\pi(1), \pi(2), \ldots, \pi(n-s))$ denote any permutation of the jobs in $N - S$, and let $\pi_i = (\pi(1), \pi(2), \ldots, \pi(i))$, for $1 \leqslant i \leqslant n-s$. The arguments for Theorem 3.1 of [1] prove that for $i = 1, 2, \ldots, n-s$,

$$L_m(\sigma_1\pi_i) - L_m(\sigma_2\pi_i) \leqslant \max_{1\leqslant k\leqslant m}\{L_k(\sigma_1) - L_k(\sigma_2)\}. \tag{9}$$

Next, $G(\sigma_j\pi) = G(\sigma_j) + \sum_{i=1}^{n-s} \max(0, L_m(\sigma_j\pi_i) - d_{\pi(i)})$, for $j = 1, 2$. Hence,

$$G(\sigma_2\pi) - G(\sigma_1\pi) = G(\sigma_2) - G(\sigma_1) - \sum_{i=1}^{n-s}(\max(0, L_m(\sigma_1\pi_i) - d_{\pi(i)}) - \max(0, L_m(\sigma_2\pi_i) - d_{\pi(i)})).$$

By Lemma 3.1 and (9),

$$\begin{aligned}(G(\sigma_2\pi) - G(\sigma_1\pi) &\geqslant G(\sigma_2) - G(\sigma_1) - \sum_{i=1}^{n-s} \max(0, L_m(\sigma_1\pi_i) - L_m(\sigma_2\pi_i)) \\ &\geqslant G(\sigma_2) - G(\sigma_1) - \max(0, (n-s) \max_{1\leqslant k\leqslant m} \{L_k(\sigma_1) - L_k(\sigma_2)\}),\end{aligned}$$

which is non-negative by (8), as required. □

The following corollary applies to the special case of consecutive jobs.

Corollary 3.1 (Job dominance). *Let σ be a partial schedule for the set of jobs S, let s = |S|, and let i and j be distinct jobs in N − S. If*

$$G(\sigma ij) - G(\sigma ji) \geqslant \max\left(0, (n-s-2) \max_{1\leqslant k\leqslant m} \{L_k(\sigma ji) - L_k(\sigma ij)\}\right), \tag{10}$$

then σji dominates σij. Moreover, dominance is strict if inequality (10) is strict.

4. The algorithm

Branch and bound is commonly applied to scheduling problems. Following [1], our algorithm employs an *adaptive depth-first plus backtracking* search strategy and uses a *dominance test and lower bound* to fathom nodes. The advantages of a depth-first strategy are: (1) the number of active nodes is always less than or equal to n, (2) the bottom of the tree is reached faster so good upper bounds can be found earlier, (3) a stack can be used to reduce computations. Our current algorithm differs from the algorithm of [1] in its choice of dominance test and lower bound (see Sections 2 and 3) and in the procedures described below for reducing the problem size and generating an initial incumbent solution. See [1] for details concerning other aspects of our algorithm—including a step-by-step outline.

Kim [8] provides a *problem size reduction procedure*, which sometimes yields a problem that has a smaller n and is easier to solve. He observes that for permutation flowshop problems with the total tardiness criterion, there always exists an optimal schedule among the class of *nondelay schedules* (i.e., schedules where no machine is kept idle when it could begin processing some operation). His Proposition 7 shows that the completion time of all jobs under any nondelay schedule is bounded above by

$$K = \min\left[\left\{\sum_{i=1}^{n} p_{i(\max)} + (m-1) \max_{1\leqslant i\leqslant n} p_{i(\max)}\right\}, \left\{\sum_{j=1}^{n} p_{(\max)j} + (n-1) \max_{1\leqslant j\leqslant m} p_{(\max)j}\right\}\right],$$

where $p_{i(\max)} = \max_{1\leqslant j\leqslant m} p_{ij}$ and $p_{(\max)j} = \max_{1\leqslant i\leqslant n} p_{ij}$. Jobs whose due dates exceed K have zero tardiness and thus can be scheduled last. Kim calls such jobs *dominated jobs*, since schedules where these jobs precede other jobs are dominated by other schedules. This leads to the following procedure. Find all dominated jobs. Schedule them in the last available positions. Then delete them from the problem. Repeat this until a problem is found with no dominated jobs. Both Kim's algorithm and our algorithm employ this procedure as a preprocessing step before starting the branch and bound algorithm.

To obtain an initial incumbent, we use the following simple m-machine heuristic. First, select a number of heuristics for a single machine problem. Second, for each u and v with $1 \leqslant u \leqslant v \leqslant m$, apply each single machine heuristic to the single machine problem with due dates $d_1, d_2, \ldots, d_n$ and processing times $q_{1,u,v}$, $q_{2,u,v}, \ldots, q_{n,u,v}$ and evaluate its total tardiness. Third, select a schedule with the smallest total tardiness as the initial incumbent schedule and the corresponding total tardiness as the initial upper bound. The rationale for this is that if one treats machine u through machine v as a single consolidated machine with unit capacity and assumes the other machines have infinite capacity, then one has a single machine problem, which is easier to handle. The number of schedules evaluated by this heuristic equals the number of single machine heuristics multiplied by $m(m+1)/2$ (the number of consolidated machines). This heuristic was originally used in [1] for the flow time objective with the shortest processing time (SPT) single machine heuristic. The numerical study of Section 5 employs the following single machine heuristics: SPT, earliest due date (EDD), minimum slack time (MST), minimum critical ratio (MCR), and Montagne's ratio method (MRM) (i.e., Procedure 1 of Kondakci et al. [9]). That study also considers adjustments where for any job, the processing time for those machines with infinite capacity is deducted from its due date.

5. Computational experiments

This section reports on computational experiments that evaluate the effectiveness of our branch and bound algorithm (CFK) and compare its performance with Kim's branch and bound algorithm (KIM)—the best alternative algorithm for the multiple machine case. The results suggest that CFK outperforms KIM and can handle test problems with $n \leqslant 20$.

CFK and KIM were coded in C and run under Borland C++ version 5.0 on a 1700 MHz Pentium 4 under Windows 2000. Our implementations of these algorithms use data structures to minimize redundant computations and employ integer variables where practical. All random numbers are generated by the *ran 1* procedure from Chapter 7 of Press et al. [14]. A simple heuristic described in Section 4 produces the initial incumbent node. Both algorithms *stop* if the node count reaches the *upper limit* of 4×10^6 without finding an optimal node. To measure performance, we record the mean node count and computation time and the percentage of problems stopped before reaching an optimal node. Note that when comparing CFK and KIM, the most important measures are the mean computation time and the percentage of problems stopped early. This is because the work per node may be different for CFK and KIM.

Our problem generation procedure outlined below yields test problems encompassing a wide variety of situations. For all problems, the processing times are generated using a scheme like that of [11]. Specifically, for $i = 1, \ldots, n$ and $k = 1, \ldots, m, p_{ik}$ follows a discrete uniform distribution on $[a_{ik}, b_{ik}]$, where a_{ik} and b_{ik} are dependent on a trend and a correlation factor. A positive trend in the processing time for job i indicates that p_{ik} increases as k increases, while a negative trend indicates that p_{ik} decreases as k increases. Similarly, a correlation between the processing times of job i exists if $p_{i1}, \ldots, p_{in}$ are consistently relatively large or relatively small. For problems with correlation, additional integers r_i, $i = 1, \ldots, n$, are randomly drawn from $\{0, 1, 2, 3, 4\}$. Depending on the existence of a trend and/or a correlation, we consider the six *p-types* summarized below.

(I) Neither correlation nor trend: $a_{ik} = 1$ and $b_{ik} = 100$.
(II) Correlation only: $a_{ik} = 20r_i$ and $b_{ik} = 20r_i + 20$.
(III) Positive trend only: $a_{ik} = 12\frac{1}{2}(k-1) + 1$, and $b_{ik} = 12\frac{1}{2}(k-1) + 100$.
(IV) Correlation and positive trend: $a_{ik} = 2\frac{1}{2}(k-1) + 20r_i + 1$, and $b_{ik} = 21/2(k-1) + 20r_i + 20$.
(V) Negative trend only: $a_{ik} = 12\frac{1}{2}(m-k) + 1$, and $b_{ik} = 12\frac{1}{2}(m-k) + 100$.
(VI) Correlation and negative trend: $a_{ik} = 2\frac{1}{2}(m-k) + 20r_i + 1$, and $b_{ik} = 2\frac{1}{2}(m-k) + 20r_i + 20$.

The due dates are generated using an approach like that of Fisher [3]. For $i = 1,\ldots,n$, the due date d_i follows a discrete uniform distribution on $[\mu_i(1 - \tau - \rho/2),\ \mu_i(1 - \tau + \rho/2)]$, where

$$\mu_i = \gamma n \sum_{k=1}^{m} p_{ik},$$

τ, ρ, and γ, respectively, are called the tardiness factor, the relative due-date range, and the tightness factor. Note that d_i has range $\rho\mu_i$ and mean $\mu_i(1 - \tau)$. Our computational experiments use the following values: $\gamma = 0.5$, 1.0, and 2.0; $\tau = 0.5$, 0.65, and 0.8; $\rho = 0.2$, 0.4, and 1.0.

Our numerical study uses twelve combinations of m and n values: $(m,n) = (2,10)$, $(4,10)$, $(6,10)$, $(8,10)$, $(2,15)$, $(4,15)$, $(6,15)$, $(8,10)$, $(2,20)$, $(4,20)$, $(6,20)$, $(8,20)$. For each (m,n) with $n = 10$ or $n = 15$, and each *p-type* (i.e., I–VI above), γ, τ, and ρ, we generate 30 problems. Similarly, for each (m,n) with $n = 20$, and each *p-type*, γ, τ, and ρ, we generate 10 problems. There are thus 45,360 test problems (i.e., 19,440 for $n = 10, 19{,}440$ for $n = 15$, and 6480 for $n = 20$). (We select a large test bed because the variances of the node counts and computation times are very high.) Tables 1–4 summarize our results.

Table 1
For KIM and CFK, the mean of the node count and computation time in seconds, and the % stopped early as a function of p-type when $n = 15$, $m = 4$, and $\gamma = 0.5$

p-Type	Node count		Time		% Stop	
	KIM	CFK	KIM	CFK	KIM	CFK
Neither correlation nor trend	1,750,712.3	94,389.3	61.994	3.056	37.4	1.1
Correlation only	417,935.4	3767.5	15.472	0.111	5.2	0.0
Positive trend only	2,334,157.0	142,389.9	63.126	3.690	48.9	0.7
Correlation and positive trend	439,677.1	2648.3	15.671	0.076	5.2	0.0
Negative trend only	1,682,905.0	320,876.7	62.363	10.038	38.5	4.1
Correlation and negative trend	455,793.9	4424.7	17.242	0.133	5.9	0.0
Average	1,180,196.8	94,749.4	39.311	2.851	23.5	1.0

Table 2
For KIM and CFK, the mean of the node count and computation time in seconds, and the % stopped early as a function of n and m when $\gamma = 0.5$

n/m	Node count		Time		% Stop	
	KIM	CFK	KIM	CFK	KIM	CFK
10/2	1570.0	56.5	0.020	0.001	0.0	0.0
10/4	12,477.5	157.5	0.220	0.003	0.0	0.0
10/6	27,552.4	197.8	0.599	0.007	0.0	0.0
10/8	41,633.5	140.4	1.038	0.007	0.0	0.0
15/2	395,089.6	3792.3	9.763	0.069	2.7	0.0
15/4	1,180,196.8	94,749.4	39.311	2.851	23.5	1.0
15/6	1,158,191.9	31,618.0	46.707	1.684	24.6	0.2
15/8	1,037,385.2	52,249.8	48.132	3.822	23.1	0.8
20/2	1,906,437.6	578,854.1	80.952	15.078	41.8	8.5
20/4	1,996,904.4	688,387.4	99.748	36.227	45.2	12.0
20/6	1,719,361.9	513,405.7	93.237	44.304	40.4	9.3
20/8	1,401,725.7	355,177.9	92.961	41.292	33.0	5.4

Table 3
For KIM and CFK, the mean of the node count and computation time in seconds and the % stopped early as a function of γ and n

γ	n	Node count		Time		% Stop	
		KIM	CFK	KIM	CFK	KIM	CFK
0.5	10	20,808.4	138.1	0.469	0.005	0.0	0.0
0.5	15	942,715.9	45,602.4	35.978	2.107	18.5	0.5
0.5	20	1,756,107.4	533,956.3	91.725	34.225	40.1	8.8
1.0	10	1687.2	54.5	0.036	0.001	0.0	0.0
1.0	15	141,390.7	12,908.1	4.344	0.286	2.0	0.2
1.0	20	467,347.2	125,456.7	18.070	4.624	10.2	2.3
2.0	10	93.8	7.8	0.001	0.000	0.0	0.0
2.0	15	6402.5	2108.4	0.135	0.038	0.1	0.0
2.0	20	58,215.3	6276.7	1.649	0.262	0.7	0.2

Table 4
For CFK, the mean of the computation time, and the % stopped early as a function of ρ, τ and m when $n = 15$ and $\gamma = 0.5$

ρ	τ	$m = 2$		$m = 4$		$m = 6$		$m = 8$	
		Time	% Stop	Time	% Stop	Time	% Stop	Time	% Stop
0.2	0.50	0.118	0.0	9.523	6.1	0.805	1.1	0.000	0.0
0.2	0.65	0.019	0.0	0.455	0.0	5.206	0.6	16.889	7.2
0.2	0.80	0.020	0.0	0.201	0.0	0.475	0.0	0.863	0.0
0.4	0.50	0.094	0.0	4.147	2.8	0.000	0.0	0.000	0.0
0.4	0.65	0.020	0.0	0.245	0.0	0.589	0.0	0.182	0.0
0.4	0.80	0.014	0.0	0.136	0.0	0.625	0.0	1.016	0.0
1.0	0.50	0.050	0.0	0.030	0.0	0.042	0.0	0.007	0.0
1.0	0.65	0.032	0.0	0.133	0.0	0.083	0.0	0.083	0.0
1.0	0.80	0.018	0.0	0.226	0.0	0.722	0.0	0.535	0.0
Average		0.043	0.0	1.677	1.0	0.950	0.2	2.175	0.8

Each test problem was solved using both KIM and CFK. Table 1 reports on how the percentage of problems stopped early by our stopping rule and the mean of the node count and computation time vary with the p-type for $n = 15$, $m = 4$, and $\gamma = 0.5$ when averaged over the τ and ρ values. The situation depicted there is typical of other n, m, and γ values in two ways: First, problems with correlation (i.e., p-types II, IV, or VI) tend to be easier for both algorithms because under correlation, some jobs tend to be more dominant than others, which increases the effectiveness of the bounds. Second, CFK substantially outperforms KIM. In particular, KIM averages almost *fourteen* times as long to solve problems as CFK, and KIM fails to solve 23.5% of the problems while CFK only fails to solve 1.0%. (The differences are greater for problems with correlation.)

Table 2 reports on how performance varies with n and m for $\gamma = 0.5$ when averaged over all p-type, τ and ρ values. As expected, the problem difficulty tends to increase dramatically with n. The relationship between problem difficulty and m is less clear. Problems with $m = 2$ tend to be easier than problems with $m = 4, 6$, or 8; however, it is not clear that problems with $m = 6$ or 8 are more difficult than ones with $m = 4$, especially when one takes the percentage of problems stopped early into account. The relationship between m and problem difficulty appears especially weak for $\gamma = 1.0$ and 2.0. (We do not present tables relating performance to n and m for $\gamma = 1.0$ and 2.0.)

The data in Table 2 indicate that CFK substantially outperforms KIM when $n = 10$ and $n = 15$ in terms of *both* computation time *and* percentage of problems stopped early: KIM averages 18.224 s and stops early with frequency 9.2%, while CFK averages 1.056 s and stops early with frequency 0.3%. For $n = 20$, CFK continues to outperform KIM in terms of computation time but not as dramatically as when $n = 10$ or 15; however when one also considers the great differences between the percentage of problems stopped early, CFK substantially outperforms KIM for $n = 20$: KIM averages 91.725 s and stops early with frequency 40.1%, while CFK averages 34.225 s and stops early with frequency 8.8%. Note that the high frequency with which KIM stops early makes the computation times for KIM artificially small. Overall, KIM's performance seems unsatisfactory when $n = 20$ and $\gamma = 0.5$.

Table 3 reports on how performance varies with n and the tightness factor γ when averaged over all m, p-type, τ and ρ values. Notice that the performance numbers are highly sensitive to γ and tend to improve as γ increases. Indeed all the performance numbers seem good when $\gamma = 2.0$. Two factors contribute to this. First, the problem size reduction procedure of Section 4 (which is used by CFK and KIM) is more effective for larger γ. (For our test problems, it reduces the mean computation times of CFK by 6.3% for $\gamma = 0.5$, 10.6% for $\gamma = 1.0$, and 27.0% for $\gamma = 2.0$.) Second, the percentage of cases where the initial incumbent objective function g^I equals 0 tends to increase with γ. (For our test problems, it equals 13.8% for $\gamma = 0.5$, 48.6% for $\gamma = 1.0$, and 64.8% for $\gamma = 2.0$.) Notice also that the numbers in Table 3 indicate that CFK continues to outperforms KIM; however, the differences decrease as γ becomes larger. Hence, CFK substantially outperforms KIM for the hard problems (i.e., $\gamma = 0.5$ and 1.0) and CFK outperforms KIM for the easy problems (i.e., $\gamma = 2.0$).

Table 4 reports on how the mean computation time of CFK varies with τ, ρ, and m for $n = 15$ and $\gamma = 0.5$ when averaged over all p-type values (each cell represents 30 randomly generated problems). Notice that many problems are solved quickly while a few problems take a long time. As before, problem difficulty does not seem to depend on m (the hardest case seems to be $\tau = 0.50$, $\rho = 0.2$, and $m = 4$). Neither τ nor ρ seem to greatly affect the mean computation time—except that fewer large mean times seem to occur with large ρ values. There is a weak negative correlation of -0.29 between the mean computation time and ρ and a very weak negative correlation of -0.11 between the mean computation time and τ. Similar patterns holds for the other n values (the details are not reported here). Note that the calculations for Table 4 were done in a later revision of this article and used Visual C++ version 6.0 rather than Borland C++ version 5.0. This is apparently why the mean computation times reported in Table 4 are smaller than those in the other tables.

6. Conclusions

The m-machine permutation flowshop problem with the total tardiness objective is a common scheduling problem, which is known to be NP-hard for all m. Here, we develop a branch and bound algorithm to solve this problem. Our algorithm incorporates a machine-based lower bound and a dominance test for pruning nodes. Extensive computational experiments suggest that the algorithm can handle test problems with $n \leqslant 20$ and dramatically outperforms Kim [8]—the best alternative algorithm for the multiple machine case.

For future research, we believe the following topics are potentially useful: (i) the application of other solution techniques to the problem, e.g., Lagrangean relaxation and slack variable decomposition; (ii) extending our branch and bound algorithm to other objectives, e.g., the minimization of total weighted tardiness; (iii) the development of efficient heuristics. Since the problem is NP-hard, it is unlikely that an effective algorithm for solving large problems exists, so (iii) is especially important. Our optimal branch and bound algorithm can play a useful role in evaluating the performance of heuristics for multiple machine problems.

References

[1] C.S. Chung, J. Flynn, O. Kirca, A branch and bound algorithm to minimize the total flowtime for m-machine permutation flowshop problems, International Journal of Production Economics 79 (2002) 185–196.

[2] C. Chu, A branch and bound algorithm to minimize total tardiness with different release dates, Naval Research Logistics 39 (1992) 265–283.
[3] M. Fisher, A dual algorithm for the one-machine scheduling problem, Mathematical Programming 11 (1976) 229–251.
[4] Y. Hirakawa, A quick optimal algorithm for sequencing on one machine to minimize total tardiness, International Journal of Production Economics 60–61 (1999) 549–555.
[5] J. Holsenback, R. Russell, A heuristic algorithm for sequencing on one machine to minimize total tardiness, Journal of Operational Research Society 43 (1992) 53–62.
[6] Y.D. Kim, Heuristics for flowshop scheduling problems minimizing mean tardiness, Journal of Operational Research Society 44 (1993) 19–28.
[7] Y.D. Kim, A new branch and bound algorithm for minimizing mean tardiness in 2-machine flowshops, Computers and Operations Research 20 (1993) 391–401.
[8] Y.D. Kim, Minimizing tardiness in permutation flowshops, European Journal of Operational Research 85 (1995) 541–555.
[9] S. Kondakci, O. Kirca, M. Azizoglu, An efficient algorithm for the single machine tardiness problem, International Journal of Production Economics 36 (1994) 213–219.
[10] C. Koulamas, The total tardiness problem: Review and extensions, Operations Research 42 (1994) 1025–1040.
[11] B.J. Lageweg, J.K. Lenstra, A.H.G. Rinnooy Kan, A general bounding scheme for the permutation flow-shop problem, Operations Research 26 (1978) 53–67.
[12] E. Lawler, A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness, Annals of Discrete Mathematics 1 (1997) 331–342.
[13] M. Pinedo, Scheduling, second ed., Prentice-Hall, Englewood Cliffs, NJ, 2002.
[14] W. Press, S. Teukolsky, W. Vetterling, B. Flannery, Numerical Recipes in C: The Art of Scientific Computing, second ed., Cambridge University Press, Cambridge, 1992.
[15] C. Potts, L. Wassenhove, A decomposition algorithm for the single machine total tardiness problem, Operations Research Letters 1 (1982) 177–181.
[16] R. Russell, J. Holsenback, Evaluation of leading heuristics for the single machine tardiness problem, European Journal of Operational Research 96 (1997) 538–545.
[17] T. Sen, P. Dileepan, J. Gupta, The two-machine flowshop scheduling problem with total tardiness, Computers and Operations Research 16 (1989) 333–340.
[18] W. Szwarc, Single machine total tardiness problem revisited, In: Y. Ijiri (Ed.), Creative and Innovative Approaches to the Science of Management, Quorum Books, 1993, pp. 407–419.
[19] W. Szwarc, F. Croce, A. Grosso, Solution of the single machine total tardiness problem, Journal of Scheduling 2 (1999) 55–71.
[20] W. Szwarc, S. Mukhopadhyay, Decomposition of the single machine total tardiness problem, Operations Research Letters 19 (1996) 243–250.
[21] B. Tansel, B. Kara, I. Sabuncuoglu, An efficient algorithm for the single machine total tardiness problem, IIE Transactions 33 (2001) 661–674.
[22] A. Volgenant, E. Teerhuis, Improved heuristic for the *n*-job single machine weighted tardiness problem, Computers and Operations Research 26 (1999) 35–44.