INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Enumeration of Pareto Optima for a Flowshop Scheduling Problem with Two Criteria

Vincent T'kindt, Federico Della Croce, Jean-Louis Bouquard,

INFORMS Journal on Computing
Vol. 19, No. 1, Winter 2007, pp. 64–72
ISSN 1091-9856 | EISSN 1526-5528 | 07 | 1901 | 0064

DOI 10.1287/ijoc.1050.0167
© 2007 INFORMS

Enumeration of Pareto Optima for a Flowshop Scheduling Problem with Two Criteria

Vincent T'kindt
Laboratory of Computer Science, University of Tours, 64 Avenue Jean Portalis, 37200 Tours, France, tkindt@univ-tours.fr

Federico Della Croce
Departimento di Automatica ed Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy, federico.dellacroce@polito.it

Jean-Louis Bouquard
Laboratory of Computer Science, University of Tours, 64 Av. J. Portalis, 37200 Tours, France, bouquard@univ-tours.fr

We consider a two-machine flowshop-scheduling problem with an unknown common due date where the objective is minimization of both the number of tardy jobs and the unknown common due date. We show that the problem is NP-hard in the ordinary sense and present a pseudopolynomial dynamic program for its solution. Then, we propose an exact ϵ-constraint approach based on the optimal solution of a related single-machine problem. For this latter problem a compact ILP formulation is explored: a powerful variable-fixing technique is presented and several logic cuts are considered. Computational results indicate that, with the proposed approach, the pareto optima can be computed, in reasonable time, for instances with up to 500 jobs.

Key words: scheduling; flowshop; multiple criteria programming
History: Accepted by John N. Hooker, Jr., Area Editor for Constraint Programming and Optimization; received November 2004; revised June 2005; accepted December 2005.

1. Introduction

In this paper we consider a two-machine flowshop-scheduling problem where n jobs have to be processed. All jobs must be processed first by machine 1 and next by machine 2, and each job i is defined by processing times a_i and b_i on machines 1 and 2, respectively. All jobs share the same due date d, which is unknown and is a variable of the problem. Let $C_{i,j}$ be the completion time of job i on machine j, and $U_i = 1$ if job i is late, i.e., if $C_{i,2} > d$. Otherwise, $U_i = 0$. Let $\bar{U}$ be the number of late jobs, $\sum_{i=1}^{n} U_i$, and the aim is to minimize $\bar{U}$ as well as the common due date d. A solution is a pair (s, d^s), where s is a schedule and d^s is its common due date, to which a criteria vector $[\bar{U}(s); d^s]$ is associated. A solution (s, d^s) is a strict pareto optimum if and only if there does not exist another solution $(s', d^{s'})$ such that $\bar{U}(s') \le \bar{U}(s)$ and $d^{s'} \le d^s$ with at least one strict inequality. Accordingly, criteria vector $[\bar{U}(s); d^s]$ is said to be *strictly nondominated*. The enumeration of strict pareto optima is often restricted to enumeration of one solution per strictly nondominated criteria vectors. Let E be this restricted set of strict pareto optima. We focus on enumeration of set E: following the classic three-field notation (Graham et al. 1979) extended to multicriteria scheduling problems (T'kindt and Billaut 2002), this problem can be referred to as $F2|d_i = d$, *unknown* $d|d, \bar{U}$. This work takes place in the context of common due date assignment and scheduling problems (Gordon et al. 2002, 2004).

Related problems have been tackled in the literature. When only criterion $\bar{U}$ is minimized, the $F|d_i|\bar{U}$ problem is $\mathcal{NP}$-hard and a branch-and-bound algorithm is provided by Hariri and Potts (1989). For some particular cases, polynomial-time algorithms are presented by Ho and Gupta (1995). When only two machines are available, several heuristics and polynomially solvable particular cases are proposed by Gupta and Hariri (1997). Jozefowska et al. (1994) show that the $F2|d_i = d|\bar{U}$ problem is weakly $\mathcal{NP}$-hard and a dynamic-programming algorithm is provided by Lawler and Moore (1969). Several heuristics (Gupta and Hariri 1994) and an efficient branch-and-bound algorithm (Della Croce et al. 2000) have also been designed. When considering more than a single criterion (see the recent survey of Hoogeveen 2005), the most closely related problem in the literature is the $F2|d_i|C_{\max}, \bar{U}$ problem (Liao et al. 1997), for which a branch-and-bound algorithm to calculate E is set up. However, to the best of our knowledge, no work exists on the $F2|d_i = d$, *unknown* $d|d, \bar{U}$ problem in this paper.

Structural properties and complexity issues are discussed in §2. A mathematical-programming formulation, from which cuts and a simple variable-fixing technique are derived, is introduced in §3. Section 4 presents an heuristic and a branch-and-bound algorithm, and in §5 computational experiments show the effectiveness of the proposed algorithms.

2. Properties and Complexity

To enumerate strict pareto optima, a classic approach iteratively solves ϵ-constrained problems (T'Kindt and Billaut 2002). The ϵ-constrained problem we consider minimizes the common due date d subject to $\bar{U} \leq \epsilon$ for $\epsilon = 0, 1, \ldots, n-1, n$. The corresponding scheduling problem is referred to as $F2|d_i = d$, *unknown* $d|\epsilon(d/\bar{U})$. Generally speaking, for each strict pareto optimum (s, d^s), there exists an ϵ such that (s, d^s) is an optimal solution of the $F2|d_i = d$, *unknown* $d|\epsilon(d/\bar{U})$ problem. However the converse is not always true (T'Kindt and Billaut 2002), i.e., the optimal solution of an ϵ-constrained problem may not be a strict pareto optimum for some values of ϵ. Despite this general bad result, we can state the following lemma.

Lemma 1. *Any optimal solution (s, d^s) to the $F2|d_i = d$, unknown $d|\epsilon(d/\bar{U})$ problem is a strict pareto optimum for criteria d and $\bar{U}$.*

Proof. All optimal solutions (s, d^s) of the ϵ-constrained problems are such that there exists a job i that completes, on the second machine, at time d^s (this can be simply proved by contradiction as the common due date must be minimized). The first implication is that the constraint $\bar{U} \leq \epsilon$ of the $F2|d_i = d$, *unknown* $d|\epsilon(d/\bar{U})$ problem is necessarily satisfied at the equality in all optimal solutions. The proof of the result is now straightforward. Consider an optimal solution (s, d^s) of the ϵ-constrained problem for a given ϵ. We have $\bar{U}(s) = \epsilon$ and due to the optimality of solution (s, d^s) we cannot find a solution $(s', d^{s'})$ such that $d^{s'} < d^s$. Hence, (s, d^s) is a strict pareto optimum. □

Due to the remarks in Lemma 1, solving the $F2|d_i = d$, *unknown* $d|\epsilon(d/\bar{U})$ problem reduces to finding a schedule of $n - \epsilon$ jobs with a minimum makespan value. We now turn to the enumeration of strict pareto optima and complexity results.

Lemma 2. *There are exactly $n+1$ strict pareto optima in E.*

Proof. A consequence of Lemma 1 and its proof is that there are at most $n+1$ distinct values of criterion $\bar{U}$. As there is no ϵ for which this problem is infeasible, we have $|E| = (n+1)$. □

Lemma 3. *The $F2|d_i = d$, unknown $d|\epsilon(d/\bar{U})$ problem is $\mathcal{NP}$-hard.*

Proof. Consider the symmetric ϵ-constrained problem, denoted $F2|d_i = d$, *unknown* $d|\epsilon(\bar{U}/d)$. This is equivalent to the $F2|d_i = D|\bar{U}$ problem where D is the bound imposed on the common due date d in the symmetric problem. Hence, as the $F2|d_i = D|\bar{U}$ problem is $\mathcal{NP}$-hard (Jozefowska et al. 1994), the symmetric problem is also $\mathcal{NP}$-hard. Since $\bar{U}$ can take $n+1$ distinct values, the decision problem associated with the symmetric problem, referred to as $F2|d_i = d$, *unknown* d, $d \leq D$, $\bar{U} \leq \epsilon|-$, is $\mathcal{NP}$-complete. It is remarkable that this decision problem is also the decision problem associated with the $F2|d_i = d$, *unknown* $d|\epsilon(\bar{U}/d)$ problem in this paper. Hence, the latter problem is $\mathcal{NP}$-hard. □

We are now ready to state the complexity of the enumeration problem.

Lemma 4. *The enumeration problem $F2|d_i = d$, unknown $d|d, \bar{U}$ is weakly $\mathcal{NP}$-hard.*

Proof. Due to Lemmas 2 and 3, the enumeration problem is also $\mathcal{NP}$-hard. We now provide a pseudo-polynomial-time algorithm to solve this problem. Consider the pseudo-polynomial dynamic programming procedure for the $F2|d_i = d|\bar{U}$ problem presented by Jozefowska et al. (1994). That procedure computes the minimum number of late jobs in a two-machine flowshop, with common due date d, with complexity $O(nd^2)$. The optimal solution value is $f_1^d(0, 0)$ where $f_k^d(t_1, t_2)$ denotes for a problem with common due date d, the minimum weighted number of late jobs for the jobs in $\{i \mid k \leq i \leq n\}$, provided that the first early job in $\{i \mid k \leq i \leq n\}$ starts processing at time t_1 on the first machine and no earlier than time t_2 on the second machine (as all possible starting times from 0 to d must be considered for both machines, this gives rise to the above complexity). Let us apply the above procedure with common due date D corresponding to the makespan obtained by the Johnson (1954) algorithm when computing the optimal schedule for the $F2\|C_{\max}$ problem. Since $f_1^D(j, j) = f_1^{D-j}(0, 0)$, the values $f_1^D(j, j)$ for $j = 0, \ldots, D$ provide the optimal solution to the $F2|d_i = d|\bar{U}$ problem for all values of d with $0 \leq d \leq D$. To solve the enumeration problem, it is then sufficient to compare the pairs $f_1^D(j, j)$, $f_1^D(j+1, j+1)$ for $j = 0, \ldots, D$ (taking $O(D)$ time). Whenever $f_1^D(j, j) < f_1^D(j+1, j+1)$ occurs, we can assume that $[f_1^D(j, j), D - j]$ is a strictly non dominated criteria vector as $D - j$ is the smallest value of the common due date that enables us to obtain only $f_1^D(j, j)$ late jobs. By comparing all the above pairs, we get E. As the dynamic-programming step requires $O(nD^2)$ time and the comparison step requires $O(D)$ time, the overall complexity remains $O(nD^2)$, which is pseudo-polynomial. □

Notice that, as the enumeration problem is weakly $\mathcal{NP}$-hard, the same occurs with the $F2|d_i = d$, *unknown* $d|\epsilon(d/\bar{U})$ problem. Besides, solving the former can be easily achieved by simply running an optimal procedure $n+1$ times for the latter. Accordingly, we focus below on the solution of the $F2|d_i = d$, *unknown* $d|\epsilon(d/\bar{U})$ problem for any given ϵ. A dominance condition proposed by Della Croce et al. (2000) for the $F2|d_i = d|\bar{U}$ problem is also valid for the ϵ-constrained problem.

Theorem 1. *Let j and i be two jobs such that $a_j \le a_i$ and $b_j \le b_i$. There exists an optimal solution such that:*
1. *if j is late, then i is also late, and*
2. *if i is early, then j is also early.*

From this result we derive two simple properties of our bicriteria problem.

Lemma 5. *Let i be a job and T be the set of jobs j such that $a_j \le a_i$ and $b_j \le b_i$. If $|T| \ge n - \epsilon$ then there exists at least one optimal solution of the $F2|d_i = d$, unknown $d|\epsilon(d/\bar{U})$ problem in which job i is late.*

Proof. Assume that $|T| \ge n-\epsilon$. Due to Theorem 1, there exists an optimal solution in which job i is late, because otherwise all the above-quoted jobs j should also be early, which would result in having too many early jobs. □

Lemma 6. *Let i be a job and T be the set of jobs j such that $a_j \ge a_i$ and $b_j \ge b_i$. If $|T| \ge \epsilon$ then there exists at least one optimal solution of the $F2|d_i = d$, unknown $d|\epsilon(d/\bar{U})$ problem in which job i is early.*

Proof. Similar to the proof of Lemma 5. □

Notice that the three results above are used in the design of the exact algorithm presented in §4.

3. An Integer-Programming Formulation

The integer-programming formulation proposed in this section is very similar to Della Croce et al. (2000) for the $F2|d_i = d|\bar{U}$ problem, which is based on the formulation of a special knapsack problem. First assume that jobs are ordered using the Johnson (1954) algorithm and indexed according to this order. The correctness of the model is based on the following two straightforward properties.

Property 1. *If the set Ω of the $n-\epsilon$ early jobs is fixed, then the optimal value of the common due date d is $d = C_{\max}(J(\Omega))$ where J refers to Johnson's algorithm.*

Property 2. *Let s be the schedule calculated by Johnson's algorithm on a set Ω of n jobs. Consider a job $k \in \Omega$ and let s_k be the schedule s where job k has been removed. We have $C_{\max}(s_k) = C_{\max}(J(\Omega - \{k\}))$, where J refers to Johnson's algorithm.*

The model is given in Figure 1 and involves two sets of constraints. Constraint (I) implies that there are exactly ϵ late jobs, and constraints (II) are critical-path constraints that define the value of the common due date d. Notice that these last constraints are "knapsack-like" constraints.

Data:	n, number of jobs
	a_i, $1 \le i \le n$, processing time on machine 1 of the job in ith position in Johnson's schedule
	b_i, $1 \le i \le n$, processing time on machine 2 of the job in ith position in Johnson's schedule
	ϵ, number of late jobs
Variables:	x_i, $1 \le i \le n$, is equal to 1 if the ith job is early and 0 otherwise
	d, the common due date
Objective:	Minimize d
Constraints:	$\sum_{i=1}^{n} x_i = n - \epsilon$ (I)
	$\sum_{i=1}^{u} a_i x_i + \sum_{i=u}^{n} b_i x_i \le d$, $\forall u = 1, \ldots, n$ (II)
	$x_i \in \{0; 1\}$, $\forall i, j = 1, \ldots, n$

Figure 1 **A Mathematical-Programming Formulation of the $F2|d_i = d$, *unknown* $d|\epsilon(d/\bar{U})$ Problem**

It is obvious that the solution of the linear-programming relaxation of the mathematical formulation given in Figure 1 leads to a lower bound on the optimal solution value. Let LB_{LP} be this lower bound. It is also possible to derive a straightforward upper bound starting from the solution of the linear-programming relaxation and by applying a basic rounding procedure: sort the variables x_j by increasing the value of $\min(x_j, 1-x_j)$ and assign variables, following the obtained order, to the closest integer value until we obtain either ϵ variables equal to 0, or $n-\epsilon$ variables equal to 1. All remaining fractional variables are therefore rounded to 0 or 1 depending on the stopping configuration. Let UB_{LP} be the common due date for the integer solution.

Consider now the linear-programming formulation of the problem where we assume that job $k+1$ in Johnson's order is the first job such that $a_{k+1} > b_{k+1}$ (hence, $a_i \le b_i$ for $i = 1, \ldots, k$). Substitute in each constraint the term $a_i + b_i$ with $\max\{a_i, b_i\}$. The first knapsack-like constraint becomes $\sum_{i=1}^{n} b_i x_i \le d$, the kth knapsack-like constraint becomes $\sum_{i=1}^{k-1} a_i x_i + \sum_{i=k}^{n} b_i x_i \le d$, the $(k+1)$th knapsack-like constraint becomes $\sum_{i=1}^{k+1} a_i x_i + \sum_{i=k+2}^{n} b_i x_i \le d$, and the last knapsack-like constraint becomes $\sum_{i=1}^{n} a_i x_i \le d$.

Let (PR) be the corresponding model. Obviously, the optimal solution value LB_{PR}, of model (PR) constitutes a lower bound on the optimal solution of the original problem. Further, it is easy to see that in (PR), all knapsack-like constraints except the first and the last become redundant, namely the first knapsack-like constraint dominates the jth constraint for $j = 2, \ldots, k$ (all the coefficients of the first constraint are greater than or equal to the corresponding coefficients of the jth constraint) and, analogously, the last knapsack-like constraint dominates the ith constraint for $i = k+1, \ldots, n-1$. Then, the optimal solution of (PR) and correspondingly LB_{PR} can be computed in $O(n \log n)$ time by enumerating all possible cases: first take the

$n-\epsilon$ variables with the smallest b_i for $i=1,\ldots,k$, next take the $n-\epsilon-1$ variables with the smallest b_i for $i=1,\ldots,k$ and the smallest a_j for $j=k+1,\ldots,n$, etc.

A straightforward upper bound on the optimal solution value of the original model can be determined by computing the value of the common due date in the original model if the same assignment of the x_i variables in the optimal solution of model (PR) is maintained: let UB_{PR} be this upper bound.

Lemma 7. *The equation*

$$\frac{\min\{\text{UB}_{\text{LP}},\text{UB}_{\text{PR}}\}}{\max\{\text{LB}_{\text{LP}},\text{LB}_{\text{PR}}\}}\le 2,$$

and this ratio is asymptotically tight.

Proof. Consider $\text{UB}_{\text{PR}}/\text{LB}_{\text{PR}}$ and remember that UB_{PR} is obtained by calculating the value of the common due date in the original model, starting from the feasible solution given by the optimal solution of (PR). As in (PR), we have simply substituted a_i+b_i with $\max\{a_i,b_i\}$, so the coefficients in the original model are at most twice the coefficients of (PR). Hence, $\text{UB}_{\text{PR}}/\text{LB}_{\text{PR}}\le 2$ and, correspondingly, $\min\{\text{UB}_{\text{LP}},\text{UB}_{\text{PR}}\}/\max\{\text{LB}_{\text{LP}},\text{LB}_{\text{PR}}\}\le \text{UB}_{\text{PR}}/\text{LB}_{\text{PR}}\le 2$. For the tightness, consider n jobs with $a_i=b_i=1$ in the case $\epsilon=n-1$. In that case, $\text{LB}_{\text{PR}}=1$ and $\text{UB}_{\text{LP}}=\text{UB}_{\text{PR}}=2$. We have $\text{LB}_{\text{LP}}=2/n+(n-1)/n=(n+1)/n$. Then, $\min\{\text{UB}_{\text{LP}},\text{UB}_{\text{PR}}\}/\max\{\text{LB}_{\text{LP}},\text{LB}_{\text{PR}}\}=2n/(n+1)$ and $\lim_{n\mapsto\infty}\min\{\text{UB}_{\text{LP}},\text{UB}_{\text{PR}}\}/\max\{\text{LB}_{\text{LP}},\text{LB}_{\text{PR}}\}=2$. □

In the remainder of this section we provide a variable-fixing technique to reduce the size of the problem, and several logic cuts on the mathematical formulation.

3.1. A Variable-Fixing Technique

In our problem, the gap $\text{UB}_{\text{LP}}-\text{LB}_{\text{LP}}$ is rather limited. Let r_j indicate the reduced cost of variable x_j and let M indicate the optimal basis in the relaxed continuous problem. The following classic inequality on the reduced costs in linear programing obviously holds: $\sum_{j\notin M} r_j x_j\le\lceil \text{UB}_{\text{LP}}-\text{LB}_{\text{LP}}\rceil$. Then, as shown by Osorio et al. (2002), it is possible to fix to 0 all variables x_j presenting reduced costs $r_j\ge\lceil \text{UB}_{\text{LP}}-\text{LB}_{\text{LP}}\rceil$ as the setting $x_j=1$ immediately induces a solution whose value is certainly at least UB_{LP}. Though not mentioned by Osorio et al. (2002), this variable-fixing technique can be applied not only to the x_j variables of the original problem not belonging to the optimal basis of the relaxed continuous problem, but also to the slack variables s_j coupled to the variables x_j in the constraints $x_j\le 1$, which can be reformulated as $x_j+s_j=1$. This leads to fixing to 1 those x_j that have necessarily $s_j=0$.

3.2. Logic Cuts on the Linear-Programming Formulation

In this section we provide problem-dependent cuts to strengthen the linear-programming relaxation of the model in Figure 1. Consider in the linear-programming formulation a pair of consecutive constraints expressed as:

$$\begin{aligned}A&=a_1x_1+\cdots+a_{i-1}x_{i-1}+(a_i+b_i)x_i+b_{i+1}x_{i+1}\\&\quad+\cdots+b_nx_n\le d\\B&=a_1x_1+\cdots+a_ix_i+(a_{i+1}+b_{i+1})x_{i+1}+b_{i+2}x_{i+2}\\&\quad+\cdots+b_nx_n\le d\end{aligned}$$

Notice that the two constraints differ only for the coefficients of variables x_i and x_{i+1}. Consider now a surrogate relaxation of these two constraints with surrogate vector $[1,1]$. With respect to the linear-programming formulation, we get $A+B\le 2d$. However, if we take into account the fact that variables x_i and x_{i+1} can be only 0 or 1, the following exhaustive four cases hold:

1. $x_i=0$, $x_{i+1}=1$. Then, $A=B-a_{i+1}$ and as $B\le d$, we have $A+B=2B-a_{i+1}\le 2d-a_{i+1}$.
2. $x_i=1$, $x_{i+1}=0$. Then, $B=A-b_i$ and as $A\le d$, we have $A+B=2A-b_i\le 2d-b_i$.
3. $x_i=1$, $x_{i+1}=1$. Then, let $C=a_1x_1+\cdots+a_{i-1}x_{i-1}+a_i+b_{i+1}+\cdots+b_nx_n$ and we have $A=C+b_i$ and $B=C+a_{i+1}$. Hence, if $b_i>a_{i+1}$, then $A>B=A+a_{i+1}-b_i$ and $A+B=2A+a_{i+1}-b_i\le 2d+a_{i+1}-b_i$. Else, if $b_i<a_{i+1}$, then $B>A=B-a_{i+1}+b_i$ and $A+B=2B-a_{i+1}+b_i\le 2d-a_{i+1}+b_i$. Consequently $A+B\le 2d-|a_{i+1}-b_i|$.
4. $x_i=0$, $x_{i+1}=0$. In this case both constraints i and $i+1$ are not tight and are therefore dominated by the first constraint (preceding or following constraints i and $i+1$) whose corresponding variable is set to 1. If it is the hth constraint of type $H=a_1x_1+\cdots+a_{h-1}x_{h-1}+a_h+b_h+b_{h+1}x_{h+1}+\cdots+b_nx_n\le d$ preceding constraint i, then we have $A=B\le H-b_h$ and hence $A+B\le 2d-2b_h$. If it is the lth constraint of the type $L=a_1x_1+\cdots+a_{l-1}x_{l-1}+(a_l+b_l)+b_{l+1}x_{l+1}+\cdots+b_nx_n\le d$ following constraint i, then $A=B\le L-a_l$ and hence $A+B\le 2d-2a_l$. Consequently, $A+B\le 2d-\min_{h<i,\,l>i+1}\{2b_h,2a_l\}$ (more precisely, if ϵ is the number of late jobs, then $A+B\le 2d-\min_{i-\epsilon+1\le h<i,\,i+\epsilon\ge l>i+1}\{2b_h,2a_l\}$).

Putting things together, we obtain:

$$\begin{gathered}A+B\le 2d-\tau\quad\text{where}\\ \tau=\min_{i-\epsilon+1\le h<i,\,i+\epsilon\ge l>i+1}\{a_{i+1},b_i,|a_{i+1}-b_i|,2b_h,2a_l\}\end{gathered}\qquad(1)$$

Thus it is possible to generate, for all pairs of consecutive constraints, a first set of $n-1$ problem-dependent cuts. It is possible to particularize these cuts for a given subproblem, i.e., when some variables are fixed. Going back to the above exhaustive four cases, if we know the value of x_i or x_{i+1}, the value of τ may obviously be increased. Let $\tau_{j,k}$ be the value of τ if variable x_j has value $k(\in\{0,1\})$. We obtain $\tau_{i,0}=\min_{i-\epsilon+1\le h<i,\,i+\epsilon\ge l>i+1}\{a_{i+1},2b_h,2a_l\}$, $\tau_{i,1}=\min\{b_i,|a_{i+1}-b_i|\}$, $\tau_{i+1,0}=\min_{i-\epsilon+1\le h<i,\,i+\epsilon\ge l>i+1}\{b_i,2b_h,

$2a_l\}$, and $\tau_{i+1,1} = \min\{a_{i+1}, |a_{i+1} - b_i|\}$. In addition, if both x_i and x_{i+1} are fixed then it is useless to generate the cut related to the constraints in which A and B were defined.

The final expression of the cuts is conservative in the sense that we consider the smallest value to be subtracted from $2d$ in the surrogate constraint. However, the linear-programming solution may have noninteger values for some variables, and we may wish to penalize that as well. Hence, the four exhaustive cases can be updated as follows:

1. The minimum value τ is obtained for $x_i = 0$, $x_{i+1} = 1$. Then, we want to penalize the possibility of having $x_i > 0$ and $x_{i+1} < 1$. Notice that, in the optimal solution, if $x_i > 0$, then $x_i = 1$. Analogously, if $x_{i+1} < 1$, then $x_{i+1} = 0$. Hence, $A + B \le 2d - \tau - (\tau_{i,1} - \tau)x_i$ and $A + B \le 2d - \tau - (\tau_{i+1,0} - \tau)(1 - x_{i+1})$.
2. The minimum value τ is obtained for $x_i = 1$, $x_{i+1} = 0$ and we want to penalize the possibility of having $x_i < 1$ and $x_{i+1} > 0$. In this case, $A + B \le 2d - \tau - (\tau_{i,0} - \tau)(1 - x_i)$ and $A + B \le 2d - \tau - (\tau_{i+1,1} - \tau)x_{i+1}$.
3. The minimum value τ is obtained for $x_i = 1$, $x_{i+1} = 1$ and we want to penalize the possibility of having $x_i < 1$ and $x_{i+1} < 1$. In this case, $A + B \le 2d - \tau - (\tau_{i,0} - \tau)(1 - x_i)$ and $A + B \le 2d - \tau - (\tau_{i+1,0} - \tau) \cdot (1 - x_{i+1})$.
4. The minimum value τ is obtained for $x_i = 0$, $x_{i+1} = 0$ and we want to penalize the possibility of having $x_i > 0$ and $x_{i+1} > 0$. In this case, $A + B \le 2d - \tau - (\tau_{i,1} - \tau)x_i$ and $A + B \le 2d - \tau - (\tau_{i+1,1} - \tau)x_{i+1}$.

Considering these four updated cases, we can generate, for all pairs of consecutive constraints, a second set of $2(n-1)$ problem-dependent cuts that are stronger than those derived from (1). In the remainder of the paper we consider these cuts as having been generated. Besides, it is possible to secialize them, for a given subproblem, as with the cuts introduced first.

3.3. Covering Inequalities and 1-Cuts

In §3.2, $2(n-1)$ new knapsack-like constraints were derived for the problem formulation. In addition to these cuts, it is possible to define covering inequalities. Consider a derived knapsack-like constraint, or even one from the original problem formulation. It is then possible to build covering inequalities called contiguous 1-*cuts* by Osorio et al. (2002), taking care to eliminate the redundant ones. The procedure for generating such cuts is in Figure 2. For each constraint, at most n 1-cuts are obtained, where generating each cut requires linear time. Hence, $O(n^2)$ 1-cuts with overall $O(n^3)$ complexity can be obtained.

```
$s = d_1$; $k = 1$; $i = 1$; $j = 2$;
While ($j < n - 1$) && ($k < n - \epsilon$) Do
   $s = s + d_j$; $l = j + 1$;
   If ($s > b$) Then
      While ($s - d_i + d_l > b$) && ($l \le n$) Do
         $s = s - d_i + d_j$; $i = i + 1$; $l = l + 1$;
      End While;
      Generate cut $x_1 + x_2 + \cdots + x_{l-1} \le k$;
      $j = l$;
   Else $j = j + 1$;
   End If;
   $k = k + 1$;
End While;
```

Figure 2 Generation of All 1-Cuts for a Constraint $d^T x \le b$ with $d_1 \ge d_2 \ge \cdots \ge d_n > 0$

4. A Branch-and-Bound Algorithm

In this section we describe a branch-and-bound algorithm for the $F2|d_i = d$, *unknown* $d|\epsilon(d/\bar{U})$ problem. This algorithm computes an optimal schedule by exploring a search tree where each node represents a subproblem. The branching scheme is binary, similar to that of the knapsack problem. A node is defined by a set X of jobs scheduled early and a set T of jobs scheduled tardy. The set of unscheduled jobs for a given node is denoted by Ω. A leaf node is one with $|T| = \epsilon$ (and hence $X = X \cup \Omega$) or $|X| = n - \epsilon$ and for this node the common due date is given by $d = C_{\max}(J(X))$ where J refers to Johnson's algorithm. Starting from a node, two child nodes are created by selecting a job $j \in \Omega$ and assigning it to X and T. The choice of job j influences the effectiveness of the branch-and-bound algorithm. For the $F2|d_i = d|\bar{U}$ problem, Della Croce et al. (2000) propose branching from Johnson's sequence, i.e., select the first job j in sequence $J(\Omega)$. However, preliminary computational experiments showed that, for the $F2|d_i = d$, *unknown* $d|\epsilon(d/\bar{U})$ problem, the best results are obtained when branching from the fractional variable closest to 0.5 in the linear-programming formulation. Intermediate results are obtained when selecting the fractional variable x_j such that the jth critical-path constraint gives the value of d in the solution of the linear-programming relaxation (if none exists, branch from the variable the closest to 0.5). We also investigated the branching scheme based on pseudo-costs as described by Linderoth and Savelsbergh (1999). It follows that, even if it helps in reducing the number of explored nodes, it is too costly to be used. However, as mentioned in the next section, the branching on pseudo-costs is applied to some particular instances for which branching on the variable closest to 0.5 is inefficient.

Notice that, whenever the branching on a given job j is applied, Theorem 1 is checked possibly to fix further jobs from Ω. In §§4.1–4.4 we describe the other features of the branch-and-bound algorithm.

4.1. Lower-Bound Computation

The basic lower bound LB_{LP} we use is calculated by solving the linear-programming relaxation of the model given in §3. If we consider a given node s of a branch-and-bound search tree, for the jobs in sets X and T, the corresponding variables x_j are set to 1 and 0, respectively, and we denote by $\mathrm{LB}_{\mathrm{LP}}(X, T, \Omega)$ the corresponding lower bound.

We consider also the following improved lower bounds:

1. Let $\mathrm{LB}_{cut}(X, T, \Omega)$ be the lower bound obtained by solving the linear-programming formulation when adding the $2(n-1)$ problem-dependent cuts given in §3.2.
2. Let $\mathrm{LB}^{cover}(X, T, \Omega)$ be the lower bound obtained by solving the linear-programming formulation when adding all the 1-cuts given in §3.3. Notice that to limit the CPU time required to solve the model and the memory space required to generate the cuts, we compute only the 1-cuts related to the first tight constraint in the father-node linear-programming relaxation. This leads to adding at most n cuts to the model.
3. Let $\mathrm{LB}^{cover}_{cut}(X, T, \Omega)$ be the lower bound obtained by solving the linear-programming formulation when adding the $2(n-1)$ problem-dependent cuts given in §3.2 and all the 1-cuts, given in §3.3, calculated on those problem-dependent cuts. This leads to adding at most $O(n^2)$ cuts to the model.

Hence, we derive four possible lower bounds at each node of the search tree.

It is remarkable that, on a small number of instances, the lower bound quickly becomes "flat," i.e., from a father node to its child nodes, the increase in the value of the lower bound is negligible. This yields a situation where the branch-and-bound algorithm is not capable of solving the corresponding instances since it reduces to a full-enumeration algorithm. An accurate analysis of this phenomenon leads us to the conclusion that this is due to the branching scheme, which does not select a "good" variable, i.e., a variable capable of making a significant increase in the lower-bound value at the child nodes. In these cases, a very efficient branching scheme is in Linderoth and Savelsbergh (1999) as branching on pseudo-costs. Henceforth, in our branch-and-bound algorithms, when the lower-bound value stagnates, we switch to branching on pseudo-costs.

4.2. Upper-Bound Computation

We compute an upper bound UB at the root node of the search tree as follows. We first calculate the feasible solution starting from the linear-programming relaxation, as indicated in §3. Next, we apply a light local search algorithm (based on swapping pairs of jobs i, j with i early and j tardy) that does not explore the complete neighborhood of the current solution at a given iteration, but rather tries to converge quickly towards a local optimum. From early computational experiments, it appears that exploring the whole neighbordhood does not lead to improved results for the branch-and-bound algorithm, but significantly increases the overall solution time.

```
/* Let $\Omega$ be the initial set of unscheduled jobs */
/* Let $H(X, T, \Omega)$ be the value of the common
   due date calculated by the heuristic */
/* given in §4.2 when applied to sets $X, T$, and $\Omega$ */
/* Let $\mathrm{LB}_{cut}(X, T, \Omega)$ be the value of the common due
   date calculated by solving the */
/* linear-programming relaxation of the model given
   in Figure 1 tightened by logic cuts */
/* taking into account sets $X, T$, and $\Omega$ */
/* Let $X^0$ and $T^0$, be the set of early and tardy jobs,
   respectively, derived from Lemmas 5 and 6 */
$\Omega^0 = \Omega / \{X^0 \cup T^0\}$;
Apply Theorem 1 on $X^0$, $T^0$, and $\Omega^0$ and update these sets;
If ($|X^0| = n - \epsilon$ or $|T^0| = \epsilon$) Then
  /* The problem is solved to optimality
     by the preprocessing procedure */
  END;
End If;
$\mathrm{UB} = H(X^0, T^0, \Omega^0)$; $\mathrm{LB} = \mathrm{LB}_{cut}(X^0, T^0, \Omega^0)$;
Perform variable fixing (§3.1) and update
  sets $X^0$, $T^0$, and $\Omega^0$;
Return LB and UB;
```

Figure 3 Reduction of the Initial Set of Unscheduled Jobs

4.3. Preprocessing the Set of Jobs

It is possible to reduce the set of unscheduled jobs at the root node s^0 of the search tree by applying a preprocessing procedure. This procedure partitions the initial set of unscheduled jobs Ω into three subsets Ω^0, X^0, and T^0, with $\Omega = \Omega^0 \cup X^0 \cup T^0$, where sets Ω^0, T^0, and X^0 belong to s^0. This is done by applying both dominance conditions of §2, the variable-fixing technique of §3.1, and the logic cuts of §3.2, as described in Figure 3. The logic cuts added in the computation of the lower bound in the preprocessing phase are left in the linear-programming formulations of all child nodes of the search tree. This option was shown experimentally to be the best compromise between effectiveness of the preprocessing and overall computing time.

4.4. A Numerical Example

In this section we apply the branch-and-bound algorithm to computation of a strict pareto optimum to an eight-job example with $\epsilon = 6$. In this algorithm at each node we calculate the lower bound $\mathrm{LB}_{\mathrm{LP}}(X, T, \Omega)$ and the cuts are used only in the preprocessing phase. This version of the branch-and-bound algorithm is called *BaB* in the §5. The data set on which the algorithm is applied is:

i	1	2	3	4	5	6	7	8
a_i	41	40	39	38	50	71	70	80
b_i	20	50	81	91	80	11	40	10

In the preprocessing phase, due to the dominance conditions given in Lemmas 5 and 6, job 5 is scheduled late. Both an upper bound UB^0 and a lower bound LB^0 are computed with values 110 and 103, respectively, where the upper-bound value is in fact the optimal solution value (with jobs 1 and 2 early and all the other jobs tardy). Notice that without using the cuts, the obtained lower bound value is 89. The application of the variable-fixing technique leads to scheduling jobs 6 and 7 late. Again, notice that without the cuts, only job 7 would have been scheduled late.

Next, the branching process starts. After creation of the root node P^0, job 2 is selected for branching. In the first branch, it is scheduled early and the associated lower bound is equal to 106. At this child node P^1, the application of the variable-fixing technique leads to scheduling jobs 3 and 4 late. Consequently, only jobs 1, 5, and 8 remain unassigned. In the second branch, job 2 is scheduled late and the associated lower bound is again equal to 106. At this child node P^2, the application of the variable-fixing technique leads to scheduling job 1 early. Consequently, only jobs 3, 4, 5, and 8 remain unassigned.

From both P^1 and P^2, we branch on job 8 to build 4 child nodes that are all closed because either their lower bound is equal to their upper bound and UB^0 is not improved, or their lower bound is greater than UB^0. Hence, the optimal solution value is 110, and 7 nodes have been developed. Notice that, without using the cuts in the preprocessing phase, 11 nodes are developed.

5. Computational Experiments

In this section we present computational experiments conducted on the $F2|d_i = d,\ d\ unknown|\epsilon(d/\bar{U})$ problem first, and then on the $F2|d_i = d,\ d\ unknown|d, \bar{U}$ problem. The data to be generated are the a_i's and b_i's, which are drawn at random using a uniform distribution between 10 and 100 (further testing on a uniform distribution between 1 and 100 provided very similar results, so the performance of the algorithm is substantially unaffected by the distribution of the data). As mentioned in §2 the value ϵ can take $n + 1$ values. However, the easiest problems seem to be those for which ϵ is close to 0 or n since they are either solved by the preprocessing step, or solved in the top of the search tree. Henceforth, we consider in these experiments the supposed hardest problems, those with $\epsilon = \lceil n/2 \rceil$. Problems with 100, 200, 300, 400, 500, 1,000, 1,500, 2,000, 2,500, and 3,000 jobs are considered and for each problem size, we generate 30 instances. All algorithms were coded in C and tested on a PC Pentium IV 2.8 GHz with 256 MB of memory.

The aim of these experiments is to test different implementations of the branch-and-bound algorithm and to compare them with CPLEX applied to the MIP model given in Figure 1. All solution algorithms were limited to problem sizes on which the average required CPU time on the 30 generated instances is lower than 180 seconds. If an algorithm meets this time limit it is declared "Out of time."

In Table 1, six distinct versions of the branch-and-bound algorithm are compared to the solution given by CPLEX. In comparison to §4,

- *BaB* is the algorithm with the lower bound $\mathrm{LB}_{\mathrm{LP}}(X, T, \Omega)$ calculated at each node,
- BaB_{cut} is the algorithm with the lower bound $\mathrm{LB}_{cut}(X, T, \Omega)$ calculated at each node,
- BaB^{cover} is the algorithm with the lower bound $\mathrm{LB}^{cover}(X, T, \Omega)$ calculated at each node,
- BaB^{cover}_{cut} is the algorithm with the lower bound $\mathrm{LB}^{cover}_{cut}(X, T, \Omega)$ calculated at each node,
- *JBaB* is the algorithm *BaB* in which we branch from Jonhson's sequence,
- $JBaB^{cover}_{cut}$ is the algorithm BaB^{cover}_{cut} in which we branch from Jonhson's sequence.

Table 1 provides average (t_{avg}) and maximum ($t_{\max}$) CPU times (in seconds) for each solution algorithm. For the branch-and-bound algorithms, the average (nd_{avg}) and maximum ($nd_{\max}$) number of nodes are also provided.

Table 1 shows that the most efficient branch-and-bound algorithm is *BaB*, which can solve problems with up to 3,000 jobs in 156 seconds on average. For higher problem sizes, *BaB* is not capable of running due to high memory requirements of the formulation. CPLEX is outperformed by *BaB* even if the former requires fewer nodes, on average, to solve the problems.

From the results of *BaB*, BaB_{cut}, BaB^{cover}, and BaB^{cover}_{cut}, within the branch-and-bound process the cuts do not save time. Only the covering inequalities help reduce the number of explored nodes. This can be explained by the fact that, at the root node of the search tree, we already tighten the LP relaxation by adding the problem-dependent cuts which are left at each node of the search tree. Adding cuts at each node appears useless.

Comparing *JBaB* and *BaB* on the one hand, and BaB^{cover}_{cut} and $JBaB^{cover}_{cut}$ on the other hand, leads to the conclusion that the branching scheme has a non negligible impact on the effectiveness of the solution process. Branching on the fractional variable closest to 0.5 or on pseudo-costs, when necessary, strongly outperforms the branching following Johnson's order.

Finally, comparing *JBaB* and $JBaB^{cover}_{cut}$ shows the effectiveness of the cuts since the number of nodes is significantly reduced in $JBaB^{cover}_{cut}$ in comparison to *JBaB*. This is more perceptible than for *BaB* and BaB^{cover}_{cut} since there are more explored nodes due to the branching scheme.

Table 1 **Comparison of Branch-and-Bound Algorithms**

	CPLEX				*BaB*			
n	t_{avg}	t_{max}	nd_{avg}	nd_{max}	t_{avg}	t_{max}	nd_{avg}	nd_{max}
100	0.13	1	44.80	121	0.03	1	40.06	181
200	0.43	1	64.50	280	0.23	1	85.23	231
300	1.23	2	114.63	386	0.40	1	97.43	329
400	2.10	5	63.86	326	0.80	2	117.40	331
500	3.50	7	84.73	592	1.53	4	134.53	383
1,000	21.20	52	84.56	428	9.36	28	171.33	921
1,500	65.70	143	117.40	630	21.46	55	136.30	481
2,000	136.53	303	79.80	537	49.36	210	153.13	1,383
2,500	Out of time				91.50	177	208.73	987
3,000	Out of time				156.36	341	179.93	1,089
3,500	Out of time				Out of time			

	BaB_{cut}				BaB^{cover}				BaB^{cover}_{cut}			
n	t_{avg}	t_{max}	nd_{avg}	nd_{max}	t_{avg}	t_{max}	nd_{avg}	nd_{max}	t_{avg}	t_{max}	nd_{avg}	nd_{max}
100	0.06	1	39.80	189	0.10	1	34.53	167	0.03	1	35.40	169
200	0.36	1	84.96	231	0.26	1	78.23	229	0.70	2	77.63	223
300	1.03	3	96.76	329	0.90	2	90.43	321	1.06	5	95.50	329
400	2.10	6	117.80	333	1.70	5	108.26	305	2.70	11	108.26	333
500	3.26	8	130.06	381	2.93	8	118.20	341	4.56	20	129.53	367
1,000	15.56	56	163.53	921	13.50	33	121.00	397	19.03	71	157.00	921
1,500	33.66	108	124.23	425	32.33	88	120.63	477	42.40	170	122.23	375
2,000	71.13	144	118.20	315	65.06	188	119.33	585	96.56	226	124.53	439
2,500	150.00	323	204.46	987	125.93	282	157.00	579	Out of time			
3,000	Out of time				Out of time				Out of time			

	JBaB				$JBaB^{cover}_{cut}$			
n	t_{avg}	t_{max}	nd_{avg}	nd_{max}	t_{avg}	t_{max}	nd_{avg}	nd_{max}
100	0.00	0	39.13	165	0.03	1	38.06	165
200	0.30	1	121.43	693	0.43	3	118.03	619
300	0.76	5	158.16	911	1.30	7	153.76	779
400	1.40	5	189.46	697	3.03	16	179.46	697
500	2.83	16	244.46	1,793	5.00	29	230.46	1,751
1,000	15.80	110	339.13	2,897	23.36	127	321.53	1,989
1,500	Out of time				Out of time			

Table 2 evaluates the effectiveness of the lower and upper bounds returned by the preprocessing phase. We first evaluate the gap between the lower and upper bounds with respect to the optimal solution value. We use G_{avg} and G_{max} to refer to the average and maximum gaps, respectively, for a given problem size, where the gap for an instance is defined as $G = 1{,}000(\mathrm{UB} - \mathrm{LB})/\mathrm{Opt}$. The distance of the optimal solution value in comparison with the lower bound is also considered, and we make use of D_{avg} and D_{max} to refer to the average and maximum distances as percentages, respectively, for a given problem size, where the distance for an instance is $D = 100(\mathrm{Opt} - \mathrm{LB})/(\mathrm{UB} - \mathrm{LB})$.

Table 2 **Comparison of Bounds**

	$\mathrm{LB}_{cut}/\mathrm{UB}$						
n	G_{avg}	G_{max}	D_{avg}	D_{max}	t_{avg}	t_{max}	*Fix*
100	2.06	5.37	37.85	80.00	0.03	1	83.90
200	1.11	2.76	30.69	66.66	0.06	1	80.00
300	0.60	2.05	30.46	50.00	0.23	1	78.87
400	0.53	1.49	26.14	50.00	0.26	1	79.46
500	0.38	1.34	25.98	50.00	0.43	1	76.40
1,000	0.16	0.40	29.82	50.00	3.10	4	88.41
1,500	0.08	0.27	35.97	50.00	9.23	10	88.82
2,000	0.07	0.15	29.16	50.00	23.26	32	89.84
2,500	0.05	0.14	28.33	33.33	43.83	48	85.97
3,000	0.02	0.10	34.44	50.00	96.10	115	86.68

Table 2 provides average (t_{avg}) and maximum (t_{max}) CPU times (in seconds) for the preprocessing phase. The average number of fixed variables during this phase (*Fix*, as a percentage of the total number of variables) is also given. As indicated in Figure 3, the preprocessing phase is based on the lower bound LB_{cut}, i.e., the problem-dependent cuts are added in the linear relaxation. Early computational experiments show that, in comparison with the single linear relaxation, the added cuts significantly help in fixing more variables and tightening the initial interval [LB; UB]. However, considering LB^{cover} or $\mathrm{LB}^{cover}_{cut}$ does

Table 3 Enumeration of Strict Pareto Optima

n	nd_{avg}	nd_{max}	t_{avg}	t_{max}
100	2,311.60	4,026	1.70	3
200	9,487.06	26,568	23.00	50
300	20,904.13	44,956	96.83	179
400	43,702.13	187,086	293.10	841
500	77,048.83	655,391	745.43	3,284

not improve the quality of the preprocessing phase but increases the required CPU time.

Table 2 shows that the number of fixed variables before starting the branch-and-bound process is important, since at least 76% of the variables are fixed. The required CPU time represents almost half of that of the branch-and-bound process.

It is also interesting to notice that the distance indicator D is, on average, always lower than 40%, which means that the lower bound is closer to the optimal solution value than is the upper bound. Even in the worst case reported in column $D_{\max}$, the upper bound is rarely closer to the optimal solution value. The gap indicator G provides additional information and the obtained results show that the gap between the lower and upper bounds is substantially reduced in comparison with the optimal solution value, even in the worst case.

Finally, we test *BaB* on the enumeration of strict pareto optima, i.e., on each instance we run *BaB* with all possible values of ϵ from 0 to n. Table 3 presents the average (t_{avg}) and maximum (t_{max}) CPU times (in seconds) required to enumerate the whole set of strict pareto optima. Correspondingly, the average (nd_{avg}) and maximum (nd_{max}) number of nodes are also provided.

Table 3 illustrates the difficulty of the enumeration problem. Even if we use an efficient algorithm capable of calculating a strict pareto optimum for problems with up to 3,000 jobs, getting the $n+1=501$ strict pareto optima for problems with 500 jobs already requires a significant amount of CPU time. This is why we did not proceed further in solving larger problems.

6. Conclusions

The solution of a bicriteria two-machine flowshop scheduling problem has been investigated. This problem is $\mathcal{NP}$-hard and several structural properties have been proposed. The number of strict pareto optima is $n+1$. Several branch-and-bound algorithms were next proposed to calculate any strict pareto optimum. They are based on problem-dependent cuts and covering inequalities, as well as an initial preprocessing phase that enables a drastic reduction in the problem size. Experimental results show that this phase enables fixing at least 76% of the problem variables. Experiments showed that the best branch-and-bound algorithm is capable of solving instances of the $F2|d_i = d$, d $unknown|\epsilon(d/\overline{U})$ problem with up to 3,000 jobs, in less than 180 seconds on average. Instances of the $F2|d_i = d$, d $unknown|d, \overline{U}$ problem with up to 500 jobs are solved in less than 750 seconds on average.

A possible application of this work is related to the lower-bound computation for other two-machine flowshop problems, such as the $F2\|\sum_j C_j$ problem. For any ϵ, the solution value d of the $F2|d_i = d$, d $unknown|\epsilon(d/\overline{U})$ problem constitutes a lower bound on the completion of the job in position $n-\epsilon$. By computing this bound over all possible values of ϵ, an overall bound on the $\sum_j C_j$ criterion can be obtained. Also, it is interesting to note that the cuts established in this work can be adapted to several other machine-scheduling problems involving minimization of the number of tardy jobs, such as the $1|d_i|\overline{U}^w$ problem (Potts and van Wassenhove 1988).

References

Della Croce, F., J. Gupta, R. Tadei. 2000. Minimizing tardy jobs in a flowshop with common due date. *Eur. J. Oper. Res.* **120** 375–381.

Gordon, V., J.-M. Proth, C. Chu. 2002. A survey of the state-of-the-art of common due date assignment and scheduling research. *Eur. J. Oper. Res.* **139** 1–25.

Gordon, V., J.-M. Proth, V. Strusevich. 2004. Scheduling with due-date assignment. J.-T. Leung, ed. *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, Vol. 1, Chap. 21. Chapman and Hall/CRC Computer and Information Science series, Chapman and Hall/CRC, Boca Raton, FL.

Graham, R. L., E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.* **5** 287–326.

Gupta, J., A. Hariri. 1994. Integrating job selection and scheduling in a flowshop. Research report, Department of Management, Ball State University, Muncie, IN.

Gupta, J., A. Hariri. 1997. Two machine flow-shop to minimize number of tardy jobs. *J. Oper. Res. Soc.* **48** 212–220.

Hariri, A., C. Potts. 1989. A branch and bound algorithm to minimize the number of late jobs in a permutation flow-shop. *Eur. J. Oper. Res.* **38** 228–237.

Ho, J., J. Gupta. 1995. Flowshop scheduling with dominant machines. *Comput. Oper. Res.* **22** 237–246.

Hoogeveen, H. 2005. Multicriteria scheduling. *Eur. J. Oper. Res.* **167** 592–623.

Johnson, S. 1954. Optimal two and three stage production schedules with set-up time included. *Naval Res. Logist. Quart.* **1** 61–68.

Jozefowska, J., B. Jurish, W. Kubiak. 1994. Scheduling shops to minimize the weighted number of late jobs. *Oper. Res. Lett.* **10** 27–33.

Lawler, E., J. Moore. 1969. A functional equation and its application to resource allocation and sequencing problems. *Management Sci.* **16** 77–84.

Liao, C.-L., W.-C. Yu, C.-B. Joe. 1997. Bicriterion scheduling in the two-machine flowshop. *J. Oper. Res. Soc.* **48** 929–935.

Linderoth, J., M. Savelsbergh. 1999. A computational study of search strategies for mixed integer programming. *INFORMS J. Comput.* **11** 173–187.

Osorio, M. A., F. Glover, P. Hammer. 2002. Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions. *Ann. Oper. Res.* **117** 71–93.

Potts, C., L. van Wassenhove. 1988. Algorithms for scheduling a single machine to minimize the weighted number of late jobs. *Management Sci.* **34** 843–858.

T'kindt, V., J.-C. Billaut. 2002. *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer-Verlag, Heidelberg, Germany.