On Solving the Flow Line Scheduling Problem with Setup Costs

Roger Z. Ríos-Mercado* Jonathan F. Bard†

Abstract

This paper presents a branch-and-cut (B&C) algorithm for the exact optimization of the flowshop scheduling problem with setup times, (SDST flowshop). This technique embeds a constraint generation stage (which reduces the size of the region of feasible solutions) into a branch-and-bound (B&B) enumerative scheme. It is found that the B&C approach outperforms the conventional B&B on a set of randomly generated instances representative of real-world data.

1 Introduction

In this paper, we address the problem of finding a permutation schedule of n jobs in an m-machine flowshop environment that minimizes the maximum completion time $C_{\max}$ of all jobs, also known as the makespan. The jobs are available at time zero and have sequence-dependent setup times on each machine. All parameters, such as processing and setup times, are assumed to be known with certainty. This problem is referred in the scheduling literature as the sequence-dependent setup time flowshop (SDST flowshop) and is regarded as a hard problem in the sense that the time it takes for any algorithm to search for an optimal solution grows, in the worst case, exponentially with the size of the input. In complexity theory terminology [3] we say that the problem is $\mathcal{NP}$-hard.

Applications of sequence-dependent scheduling are commonly found in most manufacturing environments. In the printing industry, for example, presses must be cleaned and settings changed when ink color, paper size or receiving medium differ from one job to the next. Setup times are strongly dependent on the job order. In the container manufacturing industry machines must be adjusted whenever the dimensions of the containers are changed, while in printed

*Ph.D. Candidate, Graduate Program in Operations Research/Industrial Engineering, Dept. of Mechanical Engineering, U. of Texas at Austin

†Professor, Graduate Program in Operations Research/Industrial Engineering, Dept. of Mechanical Engineering, U. of Texas at Austin

circuit board assembly, rearranging and restocking component inventories on the magazine rack is required between batches. In each of these situations, sequence-dependent setup times play a major role and must be considered explicitly when modeling the problem.

One of the solution techniques widely used to solve these hard combinatorial optimization problems is branch-and-bound (B&B). However, a variation of this method based on generating valid violated inequalities of the set of feasible solutions (called branch-and-cut (B&C)) has been found very successful in related difficult problems (e.g., see [2, 7]).

As far as the solution to the SDST flowshop is concerned, the largest instance ever solved to optimality is a 5-machine 7-job problem and took 6 hrs of CPU execution time on a PC. The objective of this paper is to present a B&C approach to the SDST flowshop, and to show its effectiveness when compared to the conventional B&B technique. We report on the solution to optimality of several instances of the SDST flowshop. The randomly generated test instances possess real-world data attributes. Our computational results demonstrate the superiority of the B&C technique.

The rest of the paper is organized as follows. In Section 2 we introduce the mathematical formulation. This is followed in Section 3 with some background material on the B&B method and the description of our B&C algorithm. The computational experiments are reported in Section 4. Finally, Section 5 contains our conclusions and directions for future research.

2 Mathematical Formulation

In the flowshop environment, a set of n jobs must be scheduled on a set of m machines, where each job has the same routing. Therefore, without loss of generality, we assume that the machines are ordered according to how they are visited by each job. Although for a general flowshop the job sequence may not be the same for every machine, here we assume a permutation schedule; i.e., a subset of the feasible schedules that requires the same job sequence on every machine. We suppose that each job is available at time zero and has no due date. We also assume that there is a setup time which is sequence dependent so that for every machine i there is a setup time that must precede the start of a given task that depends on both the job to be processed (k) and the job that immediately precedes it (j). The setup time on machine i is denoted by s_{ijk} and may be asymmetric; i.e., $s_{ijk} \neq s_{ikj}$. We also assume that the triangle inequality, stated as follows:

$$s_{ijk} + s_{ikl} \quad \geq \quad s_{ijl} \quad \text{for } i \in I,\ j,k,l \in J, \tag{1}$$

holds for the setup times. This property has been observed in real-world instances (e.g., see [11, 12]) and it means that the time it takes to set up a machine from job j to job l is less than the time it takes to set up a machine from j to another job k, and then set up the machine from k

to l. After the last job has been processed on a given machine, the machine is brought back to an acceptable "ending" state. We assume that this last operation takes zero time because we are interested in job completion time rather than machine completion time. Our objective is to minimize the time at which the last job in the sequence finishes processing on the last machine, also known as makespan. This problem is denoted by $Fm|s_{ijk},\ prmu|C_{\max}$ or SDST flowshop.

In modeling this problem as a mixed integer program (MIP), we present a formulation due to Srikar and Ghosh [11]. This formulation and an alternate formulation based on the TSP are widely discussed in [8]. In this paper we omit the latter one because it carries a larger number of binary variables and constraints.

2.1 Notation

In the development of the mathematical model, we make use of the following notation.

Indices and sets

m number of machines

n number of jobs

i machine index; $i \in I = \{1, 2, \ldots, m\}$

j, k, l job indices; $j, k, l \in J = \{1, 2, \ldots, n\}$

J_0 $= J \cup \{0\}$ extended set of jobs, including a dummy job denoted by 0

Input data

p_{ij} processing time of job j on machine i; $i \in I$, $j \in J$

s_{ijk} setup time on machine i when job j is scheduled right before job k; $i \in I$, $j \in J_0$, $k \in J$

Computed parameters

A_i upper bound on the time at which machine i finishes processing its last job; $i \in I$,

$$A_i = A_{i-1} + \sum_{j \in J} p_{ij} + \min \left\{ \sum_{j \in J_0} \max_{k \in J} \{s_{ijk}\},\ \sum_{k \in J} \max_{j \in J_0} \{s_{ijk}\} \right\}$$

where $A_0 = 0$

B_{ij} lower bound on the starting time of job j on machine i; $i \in I$, $j \in J$

$$B_{ij} = \max \{s_{i0j},\ B_{i-1,j} + p_{i-1,j}\} \qquad i \in I,\ j \in J$$

where $B_{0j} = 0$ for all $j \in J$

Real variables

y_{ij} nonnegative real variable equal to the starting time of job j on machine i; $i \in I, j \in J$

$C_{\max}$ nonnegative real variable equal to the makespan;

$$C_{\max} = \max_{j \in J} \{y_{mj} + p_{mj}\}$$

2.2 Formulation

Let $A = \{(j,k) : j,k \in J,\ j < k\}$. The decision variables are defined as follows:

$$x_{jk} = \begin{cases} 1 & \text{if job } j \text{ is scheduled any time before job } k;\ (j,k) \in A \\ 0 & \text{otherwise} \end{cases}$$

The MIP formulation is

$$\text{Minimize} \quad C_{\max} \tag{2.1}$$

subject to

$$y_{ij} + p_{ij} + s_{ijk} \leq y_{ik} + A_i(1 - x_{jk}) \quad i \in I, (j,k) \in A \tag{2.2}$$

$$y_{ik} + p_{ik} + s_{ikj} \leq y_{ij} + A_i(x_{jk}) \quad i \in I, (j,k) \in A \tag{2.3}$$

$$y_{mj} + p_{mj} \leq C_{\max} \quad j \in J \tag{2.4}$$

$$y_{ij} + p_{ij} \leq y_{i+1,j} \quad i \in I \setminus \{m\},\ j \in J \tag{2.5}$$

$$x_{jk} \in \{0,1\} \quad (j,k) \in A \tag{2.6}$$

$$y_{ij} \geq B_{ij} \quad i \in I,\ j \in J \tag{2.7}$$

Constraints (2.2) and (2.3) ensure that time precedence is not violated. They also eliminate cycles. Here, A_i is a large enough number (an upper bound on the completion time on machine i). Equation (2.4) establishes the makespan criterion. Equation (2.5) states that a job cannot start processing on one machine if it has not finished processing on the previous machine. A lower bound on the starting time of each job on each machine is set in (2.7).

In formulation (2.1)-(2.7), we assume that s_{ij0}, the time required to bring machine i to an acceptable end state when job j is processed last, is zero for all $i \in I$. Thus the makespan is governed by the completion times of the jobs only. We are also assuming that all jobs need processing on all machines. If this last condition were not true, then eq. (2.4) could be replaced by

$$y_{ij} + p_{ij} \leq C_{\max} \quad i \in I,\ j \in J$$

at the expense of increasing the number of makespan constraints from n to mn.

Srikar and Ghosh point out that the triangle inequality (1) must hold in order for constraints (2.2)-(2.3) to hold. However, Stafford and Tseng [12] provide a stronger condition for constraints (2.2)-(2.3) to be valid; i.e.,

$$s_{ijk} + s_{ikl} + p_{ik} \quad \geq \quad s_{ijl} \quad \text{for all } i \in I,\ j,k,l \in J. \tag{3}$$

Note that (3) is stronger than the triangle inequality (1), and implies that constraints (2.2)-(2.3) of the model hold, even if (1) does not hold for setup times. They illustrate this by means of an example.

2.3 Linear Programming Relaxation

The linear programming (LP) relaxation of the above formulation consists of replacing the integrality condition (2.6) by

$$0 \leq x_{jk} \leq 1 \qquad (j,k) \in A$$

so that all variables are real. To date, it has not been possible to tackle even moderate size instances of the SDST flowshop due mainly to the weakness of its LP-relaxation lower bounds. LP-based enumeration procedures such as B&B and B&C require good LP-relaxation lower bounds. For example, Stafford and Tseng required about 6 hours of CPU time on a 80286-based PC to optimally solve a 5×7 instance using LINDO with this formulation. To improve the polyhedral representation of the relaxed feasible regions it is necessary to generate valid inequalities that might be violated by a fractional solution, the strongest being facets. In [8], we develop several valid inequalities for this formulation, namely 3-subsequence elimination constraints (3-SEC), 4-subsequence elimination constraints (4-SEC), and lower bound mixed-integer inequalities. The k-SEC are inequalities that eliminate "cycles" (in the precedence sense) for any k-job subsequence. For a detailed background on polyhedral theory the reader is referred to [6].

3 The Branch-and-Cut Method

The branch-and-cut (B&C) approach to combinatorial optimization problems is a variant of the branch-and-bound (B&B) approach. The B&C method was introduced by Crowder and Padberg [2] to successfully solve large-scale instances of the well-known traveling salesman problem (TSP). B&C is considered state-of-the-art for the exact optimization of TSPs. The success of this method depends on the ability to find "strong" valid inequalities of the convex hull of the set of feasible solutions for a given mixed integer program. This has been the case for the TSP, where several valid inequalities have been developed over the past few years. The SDST flowshop, however, has not been studied from a polyhedral perspective so it is one of our aims to assess the effectiveness of B&C approaches on this type of problem. The B&B method is so well known (e.g., see [6]) such that a formal definition is omitted in this paper. Instead we outline the general scheme of our B&C algorithm.

cut type	constraint		
3-SECs	$x_{jk} + x_{kl}$	$\leq$	$1 + x_{jl}$
	$x_{jl} + (1 - x_{kl})$	$\leq$	$1 + x_{jk}$
4-SECs	$x_{jk} + x_{kl} + x_{lm}$	$\leq$	$2 + x_{jm}$
	$x_{jk} + x_{km} + (1 - x_{lm})$	$\leq$	$2 + x_{jl}$
	$x_{jl} + (1 - x_{kl}) + x_{km}$	$\leq$	$2 + x_{jm}$
	$x_{jl} + x_{lm} + (1 - x_{km})$	$\leq$	$2 + x_{jk}$
	$x_{jm} + (1 - x_{km}) + x_{kl}$	$\leq$	$2 + x_{jl}$
	$x_{jm} + (1 - x_{lm}) + (1 - x_{kl})$	$\leq$	$2 + x_{jk}$
Lower bound MICs	$(p_{ij} + s_{ijk} + B_{ij} - B_{ik})x_{jk} - y_{ik}$	$\leq$	$-B_{ik}$
	$(p_{ik} + s_{ikj} + B_{ik} - B_{ij})(1 - x_{jk}) - y_{ij}$	$\leq$	$-B_{ij}$

Table 1: *Family of valid inequalities for the SDST flowshop*

A B&B algorithm maintains a list of subproblems (nodes) of the original problem whose union of feasible solutions contains all feasible solutions of the original problem. The list is initialized with the original problem itself. In each major iteration step the algorithm selects a current subproblem from the list of unevaluated nodes. In this subproblem, several of the binary variables have already been fixed to either zero or one when the node was generated. The algorithm solves the LP-relaxation of this subproblem. This LP-relaxation provides a lower bound (for a minimization problem) for the original problem. Depending on the value of the solution, the node is either "fathomed" (e.g., if the relaxed LP is infeasible, or if the lower bound value exceeds the value of the best known integer solution), which means that no further processing of the node is necessary, or split into new subproblems (children nodes) whose union of feasible solutions contains all feasible solutions of the current subproblem. These newly generated subproblems are added to the list of unevaluated subproblems. This iteration process is performed until the list of subproblems to be fathomed is empty. The crucial part of a successful B&B algorithm is the computation of the lower bounds. The better the LP representation of the problem is, the tighter the lower bound is. This has a tremendous impact in the time effort because it is then more likely for a node to be fathomed which means that larger portions of the B&B tree will not have to be evaluated. As it has been shown, one way to improve the LP representation of a given problem is by adding valid inequalities (cutting planes or cuts). However, the main drawback of the B&B approach to this respect is that it does not allow for cut generation; that is, if violated valid inequalities are identified and added to the problem, the algorithm starts from scratch. To overcome this, the B&C algorithm integrates the cutting plane generation with enumeration phase.

Figure 1 shows a flow chart of the B&C algorithm. We now discuss the relevant steps of

the algorithm.

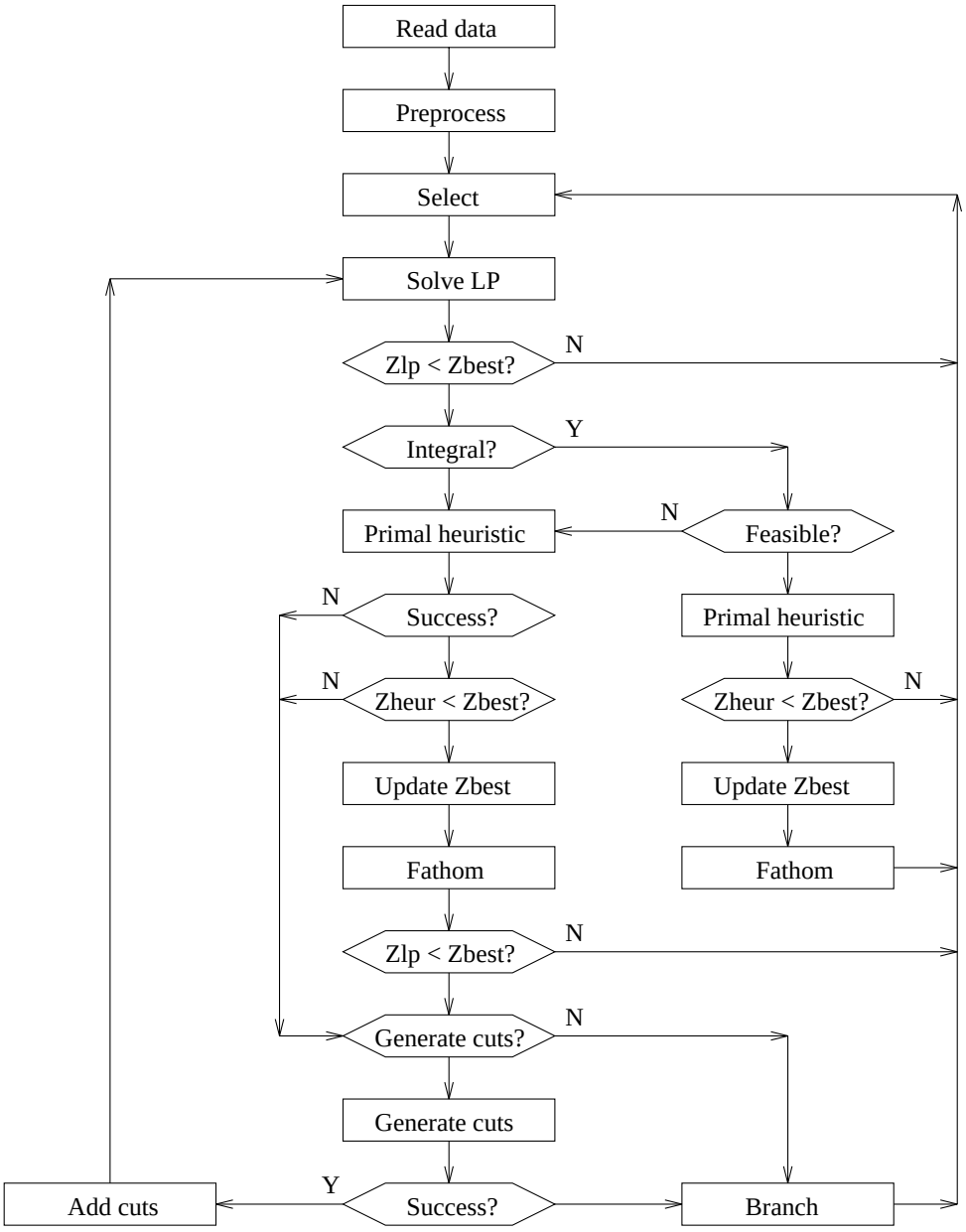

Figure 1: *Flow chart of the B&C algorithm*

Read data: Read problem data and initialize the best global feasible solution value Z_{best} to infinity.

Preprocess: After the data has been read, this stage attempts to improve the original formulation by removing redundant constraints and applying some probing techniques. The underlying idea of probing [10] is to analyze each of the inequalities of the system of inequalities defining the feasible region in turn,

trying to establish whether the inequality forces the feasible region to be empty, whether the inequality is redundant, whether the inequality can be used to improve the bounds on the variables, whether the inequality can be strengthened by modifying its coefficients, or whether the inequality forces some of the binary variables to either zero or one.

Select: A subproblem is chosen from the list of unevaluated subproblems. Here we use a best-bound node selection strategy, which selects the subproblem with the best bound; e.g., the smallest lower bound in case of a minimization problem. Best-bound search is motivated by the observations that the subproblem with the best lower bound has to be evaluated anyway and that it is more likely to contain the optimal solution that any other node.

Solve LP: The LP relaxation of the current subproblem is solved. We call its solution value Z_{lp}. If the problem is inconsistent or $Z_{lp} > Z_{best}$ the node is fathomed and go back to the selection step. If the solution has all integer variables integer and this solution is feasible, then we update the current best global feasible solution (if $Z_{lp} < Z_{best}$), fathom the node, and go back to the selection stage. Otherwise, we may apply a heuristic algorithm to attempt to find an integer feasible solution.

Primal heuristic: A heuristic (approximation) algorithm is applied to attempt to find an integer solution for the current subproblem. If successful, update the current best global feasible solution (if $Z_{heur} < Z_{best}$), fathom the node, and go back to the selection stage. In our implementation, we apply the SETUP heuristic (discussed in [9]) to the root node only to start with a good feasible solution.

Generate cuts: An attempt is made to identify a violated valid inequality. This is the most important component of the algorithm. The generated inequalities are 3-SECs (facet-inducing), 4-SECs, and lower bound mixed integer cut. See Section 2.3. If successful, we add the generated constraints to the formulation of the current subproblem and go back to solve the LP.

Branch: We need to specify how to partition (branch) the set of feasible solutions at the current node. For this type of formulation we do 0-1 variable fixing. This is based on fixing the value of a binary variable to either 0 or 1; i.e., two nodes are created. One way of determining the branching variable is to select the one with fractional value closest to $\frac{1}{2}$. The idea behind it is that it fixes a variable whose value in the optimal solution is hard to determine. These two newly created subproblems are added to the list of unevaluated nodes.

Although the conceptual algorithm stops when the list of unevaluated nodes is empty, certain stopping criteria is usually applied. Such stopping criteria includes: (i) relative gap percentage; i.e., stop when a global integer feasible solution is within $p\%$ of optimality, (ii) time limit, (iii) number of evaluated nodes limit.

4 Computational Experience

We have embedded all variants of our algorithms in MINTO. MINTO, a Mixed INTeger Optimizer [5], is a software system that solves MIPs by a branch-and-bound algorithm with linear relaxations. The user can enrich the basic algorithm by providing a variety of specialized application functions that can customize MINTO to achieve maximum efficiency for a problem class. We also use CPLEX [1], a state-of-the-art set of optimization libraries, to solve the LP relaxations. Our functions were written in C++ and linked to the MINTO 2.2 and CPLEX 4.0 libraries using the Sun C++ compiler CC version 2.0.1 and optimization flag set to -O. CPU times were obtained through MINTO.

To conduct our experiments we use randomly generated data. It has been documented [4] that the main feature in real-world data for this type of problem is the relationship between processing and setup times. It is observed that in practice the setup times are about 10-40% of the processing times. Because the experiments are expensive, we generate one class of random sets with the setup times being 30% of the processing times: $p_{ij} \in [10, 100]$ and $s_{ijk} \in [10, 30]$.

In the first experiment we aim at comparing both B&B and B&C. While it is true that B&C allows to strengthen the LP representation of the subproblems by cut generation, it also true that the size of the linear programs to be solved grows with the number of added cuts. Thus, if the generated cuts are not "good" enough, then the lower bound improvement will be downplayed by the size increase of the LP and more computational effort will have to be made. To make this comparison, we generate 5 instances for each machine combination $m \in \{2, 4, 6\}$ of an 8-job problem. For the B&C cut generation phase, we implemented the 3-SECs and the lower bound MICs.

Table 2 displays the results. The problem size is given by number of constraints (NC), number of variables (NV), and number of nonzeros (NZ). The number of binary variables is given in parenthesis (B). The average algorithmic performance over the five instances is shown in terms of number of evaluated nodes (nodes), number of linear programs solved (LPs), maximum number of rows in the LP (LP rows), CPU time in minutes. All instances were solved to optimality.

As we can see, even though the size of the LPs increases considerably (LP rows), the generated cuts are found to be effective on reducing the size of the feasible region as the B&C evaluates far fewer nodes and runs significantly faster.

instance size					average performance			
$m \times n$	NC	NV(B)	NZ	method	nodes	LPs	LP rows	time
2×8	128	45(28)	368	B&B	62780	62780	128	44.2
				B&C	37360	39901	236	36.3
4×8	256	61(28)	736	B&B	54896	54896	256	52.2
				B&C	30930	33137	368	40.7
6×8	384	77(28)	1104	B&B	55320	55320	384	68.4
				B&C	31353	33718	497	58.8

Table 2: *Comparison of B&B and B&C on 8-job instances*

More computational experience is necessary to evaluate the effectiveness of the valid inequalities within the B&C frame.

5 Directions for Future Research

We have developed a branch-and-cut based exact optimization algorithm for the SDST flowshop scheduling problem. The algorithm has proved superior to an existing branch-and-bound approach for the class of instances tested. We have solved the largest instance known to date (6-machine, 8-job). This algorithm can still be improved substantially by observing that more valid inequalities defining the convex hull of feasible solutions can be derived and implemented. This will improve the LP-relaxation of the corresponding mixed integer program. A different branching scheme must be developed as well. The current criteria branches in the fractional variable closest to 1/2. However, other partition strategy could be to branch in the position of the next job to be scheduled; i.e., given that at a given subproblem k jobs have been scheduled (in the first k positions), we partition the current node into $n-k$ subproblems where a job is chosen (among the $n-k$ unscheduled jobs) and fixed at position $k+1$. By doing this we increase the number of children at every node from 2 under the current branching scheme to $n-k$; however, this branching scheme allows us to set at once $n-k$ binary variables to either zero or one rather than just fixing one variable under the current criteria. The development of a good heuristic to find an integer solution from a fractional solution within the algorithm is also of primary concern. Currently, we use the heuristic to find a good starting feasible solution (at the root node only). In addition, the B&C algorithm must be extensively tested in other data classes. For instance, in data sets where the setup times are about 10% and 40% of the processing times (which seem to be the extreme values on real-world data sets). We do believe that the efficient implementation of B&C approaches can help both researchers and practitioners solve larger instances of this difficult problem within a reasonable time frame.

6 Acknowledgments

The research of Roger Ríos-Mercado was partially supported by the Mexican National Council of Science and Technology (CONACYT) and by an E. D. Farmer fellowship from The University of Texas at Austin. Jonathan Bard was partially supported by a grant from the Texas Higher Education Coordinating Board under the Advanced Research Program, ARP 003658–003.

References

[1] CPLEX Optimization, Inc. *Using the CPLEX Callable Library, Version 4.0*, 1995.

[2] H. Crowder and M. W. Padberg. Solving large-scale a symmetric traveling salesman problems to optimality. *Management Science*, 26(5):495–509, 1980.

[3] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[4] J. N. D. Gupta and W. P. Darrow. The two-machine sequence dependent flowshop scheduling problem. *European Journal of Operational Research*, 24(3):439–446, 1986.

[5] G. L. Nemhauser, M. W. P. Savelsbergh, and G. C. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15:48–59, 1994.

[6] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, 1988.

[7] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991.

[8] R. Z. Ríos-Mercado and J. F. Bard. The flowshop scheduling polyhedron with setup times. *Mathematical Programming*, 1996. (Submitted).

[9] R. Z. Ríos-Mercado and J. F. Bard. New heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, 1996. (Submitted).

[10] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.

[11] B. N. Srikar and S. Ghosh. A MILP model for the n-job, m-stage flowshop with sequence dependent set-up times. *International Journal of Production Research*, 24(6):1459–1474, 1986.

[12] E. F. Stafford and F. T. Tseng. On the Srikar-Ghosh MILP model for the $N \times M$ SDST flowshop problem. *International Journal of Production Research*, 28(10):1817–1830, 1990.