Computational Experience with a Branch-and-Cut Algorithm for Flowshop Scheduling with Setups[1]

Roger Z. Ríos-Mercado[2]
Graduate Program in Operations Research
University of Texas at Austin
Austin, TX 78712–1063
roger@bajor.me.utexas.edu

Jonathan F. Bard[3]
Graduate Program in Operations Research
University of Texas at Austin
Austin, TX 78712–1063
jbard@mail.utexas.edu

May 1997

[1]Technical Report ORP97–01, Graduate Program in Operations Research, University of Texas at Austin, Austin, TX 78712–1063

[2]Research 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

[3]Research partially supported by a grant from the Texas Higher Education Coordinating Board under the Advanced Research Program, ARP 003658–003

Abstract

This paper presents a branch-and-cut (B&C) algorithm for the problem of minimizing the makespan associated with scheduling n jobs in an m-machine flowshop with setup times (SDST flowshop). Two different mathematical formulations are considered. Model A is based on a traveling salesman problem-like formulation. Model B uses fewer binary variables and constraints, but is less structured than model A. A number of valid inequalities, including facet-inducing inequalities, for these two different formulations are evaluated. It was found that the B&C approach outperforms conventional branch and bound. With respect to computational effort, model B proved superior.

Keywords: flowshop scheduling, setup times, branch and cut, polyhedral representation

1 Introduction

The problem we address is one 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 evidently $\mathcal{NP}$-hard since the case where $m = 1$ is simply a traveling salesman problem (TSP).

Applications of sequence-dependent scheduling are commonly found in most manufacturing environments. In the container manufacturing industry, for example, machines must be adjusted whenever the dimensions of the containers are changed, while in printed circuit board assembly, rearranging and restocking component inventories on the magazine rack is required between batches. In the printing industry 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 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 hard combinatorial optimization problems is branch and bound (B&B). This method relies on good polyhedral representations of the convex hull of the set of feasible solutions. An extension of this method (known as branch and cut) based on generating violated valid inequalities of this convex hull has been found very effective on highly structured problems (e.g., see [5, 13]).

As far as the solution to the SDST flowshop is concerned, to the best of our knowledge, no effective methods to solve the SDST flowshop optimally have been developed to date. Efforts to solve this problem have been made by Srikar and Ghosh [20], and by Stafford and Tseng [21] in terms of solving MIP formulations. Srikar and Ghosh introduced a formulation that requires only half the number of binary variables as does the traditional TSP-based formulation. They used this model and the SCICONIC/VM mixed-integer programming solver (based on branch and bound) to solve several randomly generated instances of the SDST flowshop. The largest solved was a 6-machine, 6-job problem in about 22 minutes of CPU time on a Prime 550. Later, Stafford and Tseng corrected an error in the Srikar-Ghosh formulation and using LINDO solved a 5×7 instance in about 6 CPU hours on a PC. They also proposed three new MIP formulations of related flowshop problems based on the Srikar-Ghosh model. Other approaches have focused on heuristics [17, 19] and variations of the SDST flowshop.

In [16] we developed several valid inequalities (some of them facet-inducing) for two different formulations of the SDST flowshop. The objective of this paper is to empirically evaluate those valid inequalities within a branch-and-cut (B&C) framework, and to show the effectiveness of

this approach when compared to conventional B&B.

Our computational results demonstrate the superiority of the B&C technique. However, the size of instances that can be solved optimally is still small due to the weakness of the polyhedral representation of the convex hull.

The rest of the paper is organized as follows. In Section 2 we introduce two mathematical formulations. This is followed in Sections 3 and 4 with a summary of the valid inequalities developed in [16] which are to be evaluated, and a description of the separation procedures, respectively. In Section 5, we provide background material on B&C and then highlight our algorithm. The computational experiments are reported in Section 6. We close with a discussion of the results in Section 7.

2 Mathematical Formulation

2.1 Statement of Problem

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 is assumed to be *asymmetric*; i.e., $s_{ijk} \neq s_{ikj}$. 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 SDST flowshop or $F|s_{ijk},\ prmu|C_{\max}$ [14].

In modeling this problem as a mixed-integer program (MIP), we consider two different formulations. In the first case, a set of the binary variables is used to define whether or not one job is an immediate predecessor of another; in the second case, the binary variables simply determine whether or not one job precedes another. A set of nonnegative real variables is also included in the formulations. In either case, they have the same definition and are used to determine the starting time of each job on each machine.

Example 1 Consider the following instance of $F2|s_{ijk},\ prmu|C_{\max}$ with four jobs.

p_{ij}	1	2	3	4
1	6	3	2	1
2	2	2	4	2

s_{1jk}	1	2	3	4
0	3	4	1	7
1	-	5	3	2
2	5	-	3	1
3	2	1	-	5
4	3	2	5	-

s_{2jk}	1	2	3	4
0	2	3	1	6
1	-	1	3	5
2	4	-	3	1
3	3	4	-	1
4	7	8	4	-

A schedule $S = (3, 1, 2, 4)$ is shown in Figure 1. The corresponding makespan is 24, which is optimal. □

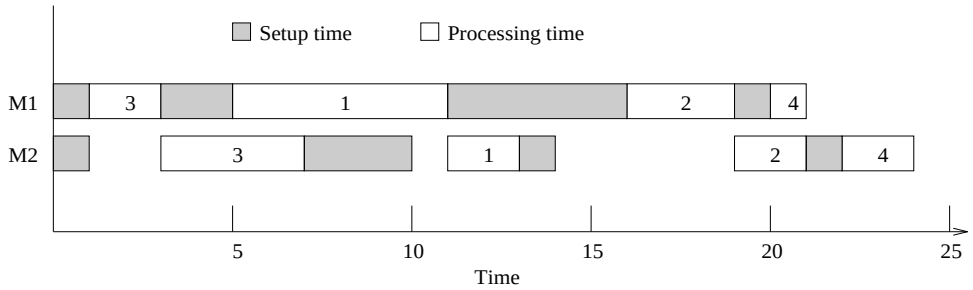


Figure 1: *Example of a* 2×4 *SDST flowshop*

Triangle inequality: The triangle inequality for the setup times is stated as follows:

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

Throughout the sequel, we will assume that the triangle inequality holds. In most operations (e.g., see [20, 21]), 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. Nevertheless, if there really exists a machine i and jobs j, k, l such that $s_{ijk} + s_{ikl} < s_{ijl}$, we can always replace s_{ijl} with $s'_{ijl} = s_{ijk} + s_{ikl}$ and force (1) to hold as an equality.

2.2 Notation

In the developments, 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 $j \in J$

Common 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;

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

2.3 Formulation A

Let $A = \{(j,k) : j,k \in J_0, j \neq k\}$ be the set of arcs in a complete directed graph induced by the node set J_0. We define the decision variables as follows:

$$x_{jk} = \begin{cases} 1 & \text{if job } j \text{ is the immediate predecessor of job } k; (j,k) \in A \\ 0 & \text{otherwise} \end{cases}$$

In the definition of x_{jk}, notice that $x_{0j} = 1$ ($x_{j0} = 1$) implies that job j is the first (last) job in the sequence for $j \in J$. Also notice that s_{i0k} denotes the initial setup time on machine i when job k has no predecessor; that is, when job k is scheduled first, for $k \in J$. This variable definition yields what we call a TSP-based formulation.

Minimize $C_{\max}$ (2.1)

subject to

$$\sum_{j \in J_0} x_{jk} = 1 \qquad k \in J_0 \tag{2.2}$$

$$\sum_{k \in J_0} x_{jk} = 1 \qquad j \in J_0 \tag{2.3}$$

$$
\begin{aligned}
y_{ij} + p_{ij} + s_{ijk} &\leq y_{ik} + A_i(1 - x_{jk}) && i \in I,\ j,k \in J && (2.4)\\
y_{mj} + p_{mj} &\leq C_{\max} && j \in J && (2.5)\\
y_{ij} + p_{ij} &\leq y_{i+1,j} && i \in I \setminus \{m\},\ j \in J && (2.6)\\
x_{jk} &\in \{0,1\} && j,k \in J_0,\ j \neq k && (2.7)\\
y_{ij} &\geq B_{ij} && i \in I,\ j \in J && (2.8)
\end{aligned}
$$

Equations (2.2) and (2.3) state that every job must have a predecessor and successor, respectively. Note that one of these $2n+2$ assignment constraints is redundant in the description of the feasible set. Time-based subtour elimination constraints are given by (2.4), and establish that if job j precedes job k, then the starting time of job k on machine i must not exceed the completion time of job j on machine i ($y_{ij} + p_{ij}$) plus the corresponding setup time. Here, A_i is a large enough number (an upper bound on the completion time on machine i). Constraint (2.5) assures that the makespan is greater than or equal to the completion time of all jobs on the last machine, while (2.6) states that a job cannot start processing on one machine if it has not finished processing on the previous one. A lower bound on the starting time for each job on each machine is set in (2.8).

In formulation (2.1)-(2.8), 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. (2.5) could be replaced by

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

at the expense of increasing the number of makespan constraints from n to mn. Note that it is possible to combine $p_{ij} + s_{ijk}$ in (2.4) into a single term $t_{ijk} = p_{ij} + s_{ijk}$, but that we still need to handle the processing times p_{ij} separately in constraints (2.5) and (2.6).

If the triangle inequality does not hold, the lower bound constraint (2.8) must be replaced by

$$B_{ij} \leq y_{ij} + C_i(1 - x_{0j}) \qquad i \in I,\ j \in J,$$

where C_i is a large number (an upper bound on the initial setup time for machine i).

2.4 Formulation B

Srikar and Ghosh [20] proposed a second MIP formulation for $F|s_{ijk}, prmu|C_{\max}$. A slight error in the constraints was later corrected by Stafford and Tseng [21]. The Srikar-Ghosh model does not consider the initial setup time s_{i0k} for the first job in the sequence, that is, it is assumed to be zero. Our formulation includes this parameter.

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

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

The MIP formulation is

$$\text{Minimize} \quad C_{\max} \tag{3.1}$$

subject to

$$y_{ij} + p_{ij} + s_{ijk} \leq y_{ik} + A_i(1 - x_{jk}) \quad i \in I, (j,k) \in \hat{A} \tag{3.2}$$

$$y_{ik} + p_{ik} + s_{ikj} \leq y_{ij} + A_i(x_{jk}) \quad i \in I, (j,k) \in \hat{A} \tag{3.3}$$

$$y_{mj} + p_{mj} \leq C_{\max} \quad j \in J \tag{3.4}$$

$$y_{ij} + p_{ij} \leq y_{i+1,j} \quad i \in I \setminus \{m\},\ j \in J \tag{3.5}$$

$$x_{jk} \in \{0,1\} \quad (j,k) \in \hat{A} \tag{3.6}$$

$$y_{ij} \geq B_{ij} \quad i \in I,\ j \in J \tag{3.7}$$

Constraints (3.2) and (3.3) ensure that time precedence is not violated. They also eliminate cycles. Equation (3.4) establishes the makespan criterion. Equation (3.5) 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 (3.7).

Srikar and Ghosh point out that the triangle inequality (1) must hold in order for constraints (3.2)-(3.3) to hold. However, Stafford and Tseng provide a stronger condition for constraints (3.2)-(3.3) to be valid; i.e.,

$$s_{ijk} + s_{ikl} + p_{ik} \geq s_{ijl} \quad \text{for all } i \in I,\ j,k,l \in J. \tag{4}$$

Note that (4) is stronger than the triangle inequality (1), and implies that constraints (3.2)-(3.3) of the model hold, even if (1) does not hold for setup times. They illustrate this by means of an example.

If the triangle inequality does not hold, constraints (3.2), (3.3) and (3.7) are no longer valid. One possible replacement is

$$y_{ij} + p_{ij} + s_{ijk} \leq y_{ik} + (n+1)A_i(1 - x_{jk}) + A_i[P(k) - P(j) - 1] \quad i \in I,\ (j,k) \in \hat{A}$$

$$y_{ij} + p_{ij} + s_{ijk} \leq y_{ik} + (n+1)A_i x_{jk} + A_i[P(j) - P(k) - 1] \quad i \in I,\ (j,k) \in \hat{A}$$

$$B_{ik} \leq y_{ik} + C_i[P(k) - 1] \quad i \in I,\ k \in J,$$

respectively, where C_i is a large enough number (upper bound on the starting processing time of all jobs on machine i), and $P(j)$ represents the position in the schedule of job j given by

$$P(j) = \sum_{p<j} x_{pj} + \sum_{q>j} (1 - x_{jq}) + 1 \quad j \in J. \tag{5}$$

In addition, the following constraints must be added to the formulation:

$$
\begin{aligned}
P(j)+1 &\leq P(k)+n(1-x_{jk}) \quad (j,k)\in \hat{A}\\
P(k)+1 &\leq P(j)+nx_{jk} \quad (j,k)\in \hat{A}
\end{aligned}
$$

Thus when the triangle inequality does not hold, the problem size increases considerably.

2.5 Model Comparison

	Model A		Model B	
Variables	Binary	$n(n+1)$	Binary	$\frac{1}{2}n(n-1)$
	Real	$mn+1$	Real	$mn+1$
	Total	$n(n+1)+mn+1$	Total	$\frac{1}{2}n(n-1)+mn+1$
Constraints	(2.2)	$n+1$	(3.2)	$\frac{1}{2}mn(n-1)$
	(2.3)	$n+1$	(3.3)	$\frac{1}{2}mn(n-1)$
	(2.4)	$mn(n-1)$		
	(2.5)	mn	(3.4)	mn
	(2.6)	$n(m-1)$	(3.5)	$n(m-1)$
	Total	$mn^2+mn+n+2$	Total	mn^2+mn-n
Nonzeros	(2.2)	$n(n+1)$	(3.2)	$\frac{3}{2}mn(n-1)$
	(2.3)	$n(n+1)$	(3.3)	$\frac{3}{2}mn(n-1)$
	(2.4)	$3mn(n-1)$		
	(2.5)	$2mn$	(3.4)	$2mn$
	(2.6)	$2n(m-1)$	(3.5)	$2n(m-1)$
	Total	$3mn^2+2n^2+mn$	Total	$3mn^2+mn-2n$

Table 1: *Problem size for models A and B*

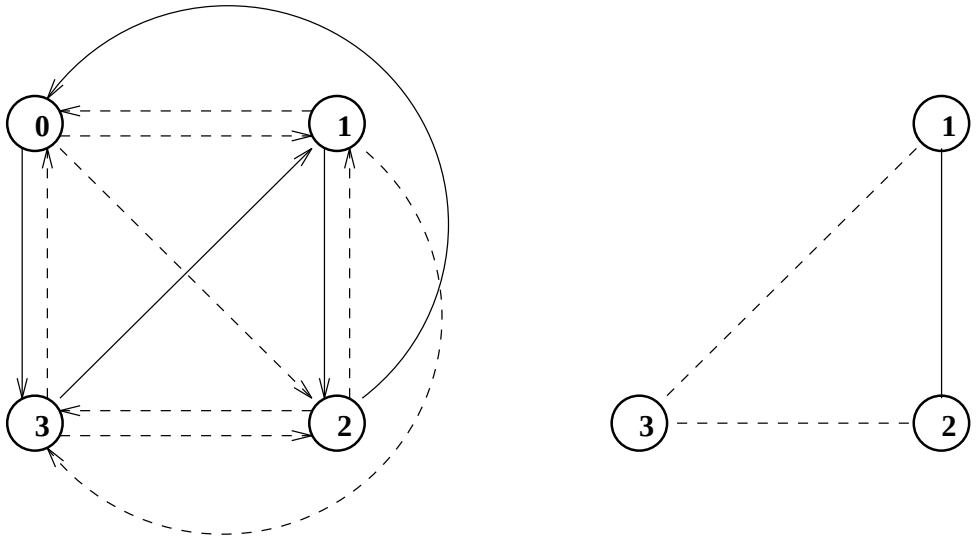
Table 1 shows the problem size in terms of number of variables, constraints, and nonzeros for either model. As can be seen, model B is considerably smaller than model A in terms of both the number of constraints and the number of binary variables. This would appear to make it more attractive when considering exact enumeration methods such as branch and bound and branch and cut. Nevertheless, the fact that much is known about the ATSP polytope gives added weight to model A. Table 2 displays the number of binary and real variables, number of constraints, number of nonzeros and density of the matrix of constraints for several values of m and n.

To date, it has not been possible to tackle even moderate size instances of the SDST flowshop with either of these formulations due mainly to the weakness of their 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 formulation B. To improve the polyhedral representation of the relaxed feasible regions it is necessary to generate valid inequalities, the strongest being facets. One way to achieve this is by looking into the

$m \times n$	Model	Binary	Real	Constraints	Nonzeros	Density
2×10	A	110	21	252	840	0.025
	B	45	21	230	620	0.041
2×20	A	420	41	902	3280	0.008
	B	190	41	860	2440	0.012
10×10	A	110	101	1212	3400	0.013
	B	45	101	1190	3180	0.018
10×20	A	420	201	4422	13200	0.005
	B	190	201	4380	12360	0.007

Table 2: *Problem size examples for models A and B*

related subspaces: the ATSP polytope and the Srikar-Ghosh (S-G) polytope for models A and B, respectively. Many facets have been developed for the ATSP polytope over the last 20 years (e.g., see [1, 2, 3, 6, 15]). For model B, though, the S-G polytope had remained unexplored. In an early paper [16], we showed that the facets of either of these polytopes can be extended to facets of the SDST flowshop polyhedron. We also developed several mixed-integer cuts for both models. The main results are summarized in Section 3.



(a) Model A (b) Model B

Figure 2: *Graph representations for schedule (3,1,2)*

When comparing the ATSP polytope with the S-G polytope fundamental differences can be observed. In the former, we have a clear picture of what a feasible solution (also called a tour) looks like in a graph. This makes it easier to visualize, for instance, when certain constraints, such as the subtour elimination constraints, may be violated. However, for model B, it is not a straightforward matter to identify in a graph a feasible solution from a given set of arcs.

Figure 2 shows the graph for a 3-job problem and the solution for schedule $S = (3, 1, 2)$ for both models. For model B, an undirected graph can be used because x_{jk} is only defined for $j < k$. The dotted lines represent all feasible arcs (12 for model A and 3 for B); the solid lines identify the solution.

3 Valid Inequalities

What distinguishes B&C from traditional cutting plane methods is that the inequalities generated are valid at each node of the search tree. Previously, we developed several valid inequalities for formulations A and B. We now summarize those results.

For model A, we showed that if $\{x \in P \ : \ \pi x = \pi_0\}$ is a facet of P, where P is the convex hull of the set of feasible solutions of a $(n+1)$-city ATSP, then

$$\{(x, y) \in P_A \ : \ (\pi, 0)(x, y)^T = \pi_0\}$$

is a facet of the convex hull of the set of feasible solutions of the SDST flowshop, where $x \in B^{n(n+1)}$ corresponds to an incidence vector of a tour in a $(n+1)$-city ATSP, $y \in R^{nm+1}$ is the vector of real-variables y in formulation A, and P_A denotes the convex hull of the set of feasible solutions of the SDST flowshop under formulation A. This result says that any of the facets developed for the ATSP can be applied to the SDST flowshop. In our work, we implemented subtour elimination constrains (SECs) and D_k^+ and D_k^- inequalities (e.g., see [9]) which are two of the most successful facets developed for the ATSP. Among these, we found that the SECs were much more effective. The D_k^+ and D_k^- inequalities had little or no impact on improving the polyhedral representation of the SDST flowshop polyhedron.

We also developed mixed-integer cuts (MICs) of the following form:

$$\begin{array}{lrcll} \text{(LBMICs)} & (p_{ij} + s_{ijk} + B_{ij} - B_{ik})x_{jk} - y_{ik} & \leq & -B_{ik} & \text{and} \\ \text{(UBMICs)} & (U_i + s_{ijk})(1 - x_{jk}) + y_{ik} - y_{ij} & \geq & p_{ij} + s_{ijk} & \end{array}$$

where, B_{ij} is a lower bound on y_{ij} as defined in Section 2.2, and U_i is an upper bound on the completion time of machine i.

For model B, we developed 3-subsequence elimination constraints (3-SEC), 4-subsequence elimination constraints (4-SEC), and both lower and upper bound mixed-integer inequalities. The k-SEC are inequalities that eliminate "cycles" (in the precedence sense) for any k-job subsequence. These are shown in Table 3, where B_{ij} and U_i are defined as before.

4 Separation Algorithms

For a given class of valid inequalities, the associated separation problem can be stated as follows: Given a point $\bar{x} \in R^p$ satisfying a certain subset of constraints, and a family F of SDST flowshop

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	$(p_{ij} + s_{ijk} + B_{ij} - B_{ik})x_{jk} - y_{ik}$	$\leq$	$-B_{ik}$
	$(p_{ik} + s_{ikj} + B_{ik} - B_{ij})(1 - x_{jk}) - y_{ij}$	$\leq$	$-B_{ij}$
Upper bound MICs	$(U_i + s_{ijk})(1 - x_{jk}) + y_{ik} - y_{ij}$	$\geq$	$p_{ij} + s_{ijk}$
	$(U_i + s_{ikj})x_{jk} + y_{ij} - y_{ik}$	$\geq$	$p_{ik} + s_{ikj}$

Table 3: *Family of valid inequalities for model B*

inequalities, find the most violated member of F, i.e., an inequality $\alpha x \leq \alpha_0$ belonging to F and maximizing the degree of violation $\alpha\bar{x} - \alpha_0$. When this problem is solved optimally, we say that we have an exact separation algorithm. However, sometimes the separation problem is as difficult as the original problem so it is necessary to resort to heuristics to identify violated inequalities. Below we describe the procedures developed for models A and B.

4.1 Separation Procedures for SECs for Model A

Let $(\bar{x}, \bar{y}) \in R^{|A|} \times R^{mn+1}$ be a point satisfying constraints (2.2)-(2.6). This point is obtained by relaxing the integrality restriction on the binary variables x and solving the corresponding LP. As stated in Section 3, any facet for the ATSP is a facet for the SDST flowshop, where only the binary variables x are considered. Therefore, we drop the real variables y and are left with the problem of finding a violation of the classical TSP subtour elimination constraint

$$\sum_{(j,k) \in A \,:\, j,k \in W} \bar{x}_{jk} \quad \leq \quad |W| - 1 \tag{6}$$

for some $W \subseteq J, 2 \leq |W| \leq n - 1$, or prove that none exists. Note that (6) is equivalent to

$$\sum_{\substack{j \in W \\ k \in J \setminus W}} \bar{x}_{jk} + \sum_{\substack{j \in J \setminus W \\ k \in W}} \bar{x}_{jk} \quad \geq \quad 2 \tag{7}$$

SECs for the ATSP are symmetric inequalities, that is, inequalities of the form $\alpha x \leq \alpha_0$ with $\alpha_{jk} = \alpha_{kj}$ for all $(j,k) \in A$. Symmetric inequalities for the ATSP have a very important property. It has been shown [9] that there exists a correspondence between valid inequalities for the ATSP and valid inequalities for the symmetric TSP (STSP). If we define the mapping $f : R^A \rightarrow R^E$ (A is the arc set of the complete digraph and E is the edge set of the corresponding

undirected graph) as follows: $f(\bar{x}) = \hat{x}$, where $\hat{x}_{jk} = \bar{x}_{jk} + \bar{x}_{kj}$ for all $j \neq k$, then we have $f(P) = Q$, where P and Q are the polytopes of the ATSP and STSP, respectively. In other words, every inequality $\sum_{e \in E} \alpha_e \hat{x}_e \leq \alpha_0$ for STSP can be transformed into a valid ATSP inequality by simply replacing $\hat{x}_e$ by $x_{jk} + x_{kj}$ for all $e = (j,k) \in E$. This produces the symmetric inequality $\alpha x \leq \alpha_0$, where $\alpha_{jk} = \alpha_{kj}$ for all $j,k \in J, j \neq k$. Conversely, every symmetric ATSP inequality $\alpha x \leq \alpha_0$ corresponds to the valid STSP inequality $\sum_{e \in E} \alpha_e \hat{x}_e \leq \alpha_0$.

The above correspondence implies that every separation algorithm for STSP can be used, as a black box, for ATSP as well. Therefore, given the point $\bar{x}$, we first define the symmetric counterpart $\hat{x}$ of $\bar{x}$ by the transformation $\hat{x}_{jk} = \bar{x}_{jk} + \bar{x}_{kj}$ for all $j,k \in J$, and then apply a STSP separation algorithm to $\hat{x}$.

Now, let us define the undirected support graph of $\hat{x}$, denoted $G(\hat{x})$, as the graph formed by $n+1$ vertices (n jobs plus a dummy job) and an edge (j,k) of weight $\hat{x}_{jk}$ for each $\hat{x}_{jk} > 0$. The problem of finding a violated SEC for STSP is equivalent to finding a cut in $G(\hat{x})$ that is less than 2. That is, given $\hat{x} \in R^E$ satisfying $0 \leq \hat{x}_{jk} \leq 1$ for all $(j,k) \in E$ and the assignment constraints (2.2)-(2.3), find a nonempty proper subset W of J such that

$$\sum_{\substack{j \in W \\ k \in J \setminus W}} \hat{x}_{jk} \quad < \quad 2 \tag{8}$$

holds, or prove that no such $W \subseteq J$ exists, where (8) is the violated version of (7) for the symmetric case.

Consequently, what we are interested in is finding a minimum capacity cut-set in the support graph $G(\hat{x})$ where the capacities are given by the weights $\hat{x}_{jk}$, $(j,k) \in E$. If the minimum cut-set in $G(\hat{x})$ has a capacity which is greater than or equal to 2, then we conclude that there exists no SEC that is violated by $\hat{x}$. Otherwise a vertex set W given by a minimum capacity cut-set defines a violated SEC.

To solve the separation problem, we use the MINCUT algorithm developed by Padberg and Rinaldi [12]. This algorithm has a time complexity of $O(n^4)$, which is the same complexity as the algorithm developed by Gomory and Hu [8]. However, empirical evidence over a large class of graphs has demonstrated the superiority of MINCUT over the Gomory-Hu procedure.

Example 2 Consider the following instance of $F2|s_{ijk},\ prmu|C_{\max}$ with seven jobs.

p_{ij}	$j=1$	2	3	4	5	6	7
$i=1$	68	43	95	95	69	66	55
2	44	66	74	92	34	55	52

s_{1jk}	$k=1$	2	3	4	5	6	7
$j=0$	30	33	25	29	39	32	31
1	-	37	24	26	27	34	39
2	22	-	39	28	31	29	31
3	25	32	-	40	33	23	40
4	35	28	40	-	25	25	27
5	40	28	29	29	-	40	23
6	32	26	32	29	20	-	28
7	37	25	28	37	35	26	-

s_{2jk}	$k=1$	2	3	4	5	6	7
$j=0$	35	33	24	40	21	27	40
1	-	35	20	33	37	20	32
2	27	-	24	28	35	20	33
3	30	20	-	36	24	34	35
4	29	36	25	-	20	40	27
5	35	32	20	38	-	28	29
6	34	26	22	23	39	-	27
7	20	39	20	37	40	25	-

Suppose that at some node in the B&C search tree, the following fractional solution is obtained (LP relaxation):

$$
\begin{array}{llllllllllll}
\bar{x}_{12} & = & 0.8540 & \bar{x}_{40} & = & 1.0000 & \bar{y}_{11} & = & 30 & \bar{y}_{21} & = & 98 \\
\bar{x}_{13} & = & 0.1460 & \bar{x}_{53} & = & 0.8540 & \bar{y}_{12} & = & 33 & \bar{y}_{22} & = & 76 \\
\bar{x}_{25} & = & 0.9113 & \bar{x}_{57} & = & 0.1460 & \bar{y}_{13} & = & 25 & \bar{y}_{23} & = & 120 \\
\bar{x}_{27} & = & 0.0887 & \bar{x}_{64} & = & 0.8723 & \bar{y}_{14} & = & 29 & \bar{y}_{24} & = & 124 \\
\bar{x}_{31} & = & 0.6375 & \bar{x}_{67} & = & 0.1278 & \bar{y}_{15} & = & 39 & \bar{y}_{25} & = & 108 \\
\bar{x}_{32} & = & 0.0386 & \bar{x}_{72} & = & 0.1074 & \bar{y}_{16} & = & 32 & \bar{y}_{26} & = & 98 \\
\bar{x}_{34} & = & 0.1278 & \bar{x}_{76} & = & 0.8926 & \bar{y}_{17} & = & 31 & \bar{y}_{27} & = & 164 \\
\bar{x}_{35} & = & 0.0887 & \bar{x}_{01} & = & 0.3625 & \bar{C}_{\max} & = & 216 & & & \\
\bar{x}_{36} & = & 0.1074 & \bar{x}_{07} & = & 0.6375 & & & & & &
\end{array}
$$

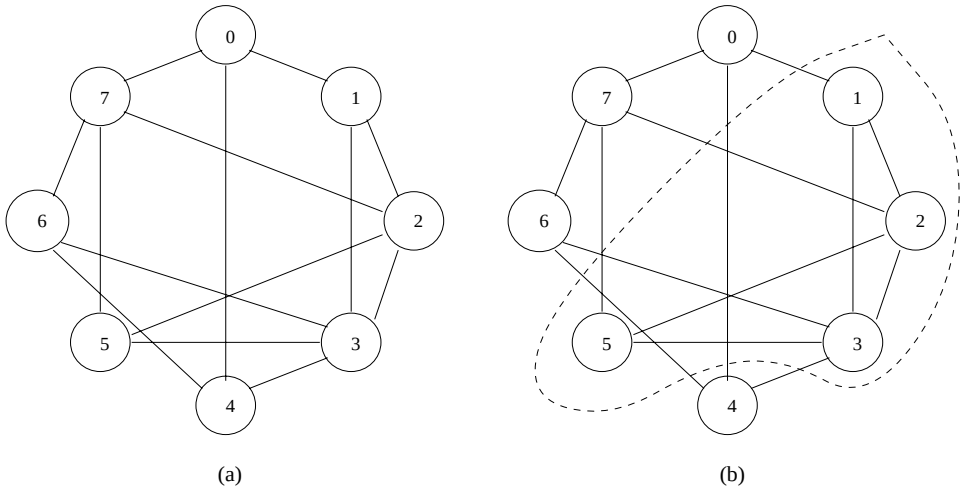


Figure 3: *The support graph of* $\hat{x}$

We transform $\bar{x}$ into its corresponding symmetric counterpart $\hat{x}$ using the transformation $\hat{x}_{jk} = \bar{x}_{jk} + \bar{x}_{kj}$ and then form $G(\hat{x})$, its support graph (depicted in Figure 3(a)). The edge weights are given by

Edge	Weight	Edge	Weight
(0,1)	0.3625	(2,7)	0.1961
(0,4)	1.0000	(3,4)	0.1278
(0,7)	0.6375	(3,5)	0.9427
(1,2)	0.8540	(3,6)	0.1074
(1,3)	0.7835	(4,6)	0.8723
(2,3)	0.0386	(5,7)	0.1460
(2,5)	0.9113	(6,7)	1.0204

By applying the MINCUT algorithm, we find that the minimum cut-set is given by $W = \{1,2,3,5\}$ (shown in Figure 3(b)) with cut capacity equal to $x_{01}+x_{27}+x_{34}+x_{36}+x_{57} = 0.9398$. Since $0.9398 < 2$, the set W violates the following SEC:

$$x_{12} + x_{13} + x_{15} + x_{21} + x_{23} + x_{25} + x_{31} + x_{32} + x_{35} + x_{51} + x_{52} + x_{53} \quad \leq \quad 3 = |W| - 1$$

for the ATSP. □

4.2 Separation Procedures for D_k^+ and D_k^- Inequalities

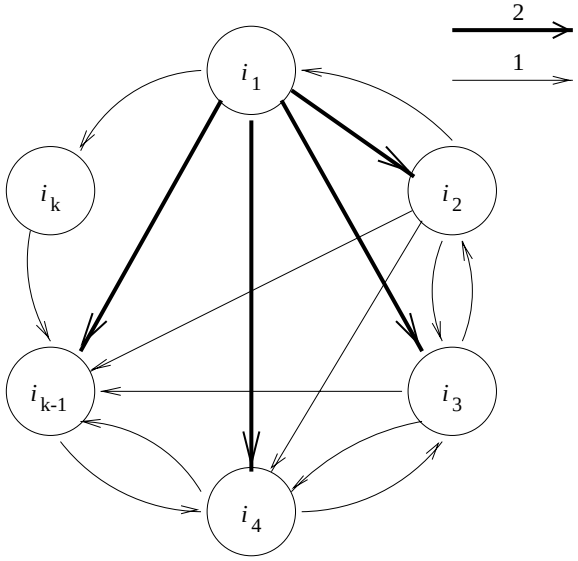


Figure 4: *The Support Multigraph of a D_k^+ inequality*

The following inequalities were derived by Grötschel and Padberg [9]:

$$(D_k^+) \quad x_{i_1 i_k} + \sum_{h=2}^{k} x_{i_h i_{h-1}} + 2\sum_{h=2}^{k-1} x_{i_1 i_h} + \sum_{h=3}^{k-1}\sum_{j=2}^{h-1} x_{i_j i_h} \quad \leq \quad k-1 \tag{9}$$

$$(D_k^-) \quad x_{i_k i_1} + \sum_{h=2}^{k} x_{i_{h-1} i_h} + 2\sum_{h=2}^{k-1} x_{i_h i_1} + \sum_{h=3}^{k-1}\sum_{j=2}^{h-1} x_{i_h i_j} \quad \leq \quad k-1 \tag{10}$$

where $(i_1 \ldots, i_k)$ is any sequence of $k \in \{3, \ldots, n-1\}$ distinct nodes. A D_k^+ inequality for $k = 6$ is depicted in Figure 4, where arcs in dotted and solid line have coefficient 2 and 1, respectively. D_k^+ and D_k^- inequalities are facet-inducing for the ATSP polytope [6], and are obtained by lifting the cycle inequality $\sum_{(j,l)\in C} x_{jl} \leq k-1$ associated with the circuit $C = \{(i_1, i_k), (i_k, i_{k-1}), \ldots, (i_2, i_1)\}$ and $C = \{(i_1, i_2), \ldots, (i_{k-1}, i_k), (i_k, i_1)\}$, respectively.

The separation problem for D_k^+ inequalities consists of finding a node sequence $(i_1, \ldots, i_k)$, $3 \leq k \leq n-1$, such that (9) is violated. An exact enumeration scheme is proposed by Fischetti and Toth [7]. Here we use the following procedure. We first attempt to find all cycles in $G(\bar{x})$. Although the number of cycles in a complete graph may be large, usually $G(\bar{x})$ (coming from the SDST flowshop fractional solution) is relatively sparse, which allows us to identify the cycles in a relatively short amount of time. Then, for each cycle we attempt to find a violated D_k^+ and store the one with the largest degree of violation.

As pointed out by Fischetti and Toth [7], the D_k^- inequalities can be thought of as derived from D_k^+ inequalities by swapping the coefficient of the two arcs (j,k) and (k,j) for all $j, k \in J$, $j < k$. This is called a transposition operation. They show how this transposition enables the use of the separation procedures designed for D_k^+ inequalities as a separation procedure for D_k^- inequalities.

After implementing the D_k^+ and D_k^- separation procedures, we found they had very little impact on the overall performance of our B&C algorithm. Empirically, the SECs did a far better job in tightening the polyhedral representation. In our computations, only a very small number of D_k^+ and D_k^- inequalities were identified and, when added to the set of cuts, provided an insignificant improvement in the value of the LP relaxation.

Example 3 (Example 2 continued)
For the same fractional point $(\bar{x}, \bar{y})$, consider the following node sequence $(3, 5, 2, 1, 0, 4)$. Evaluating eq. (9) for $k = 6$ and $(i_1, \ldots, i_6) = (3, 5, 2, 1, 0, 4)$, we see that

$$\begin{aligned}
\bar{x}_{i_1 i_6} + \sum_{h=2}^{6} \bar{x}_{i_h i_{h-1}} + 2\sum_{h=2}^{5} x_{i_1 i_h} + \sum_{h=3}^{5}\sum_{j=2}^{h-1} x_{i_j i_h} &= (\bar{x}_{34} + \bar{x}_{40} + \bar{x}_{01} + \bar{x}_{12} + \bar{x}_{25} + \bar{x}_{53}) \\
&\quad + 2(\bar{x}_{35} + \bar{x}_{32} + \bar{x}_{31} + \bar{x}_{30}) \\
&\quad + (\bar{x}_{52} + \bar{x}_{51} + \bar{x}_{21} + \bar{x}_{50} + \bar{x}_{20} + \bar{x}_{10}) \\
&= (4.1096) + 2(0.7648) + (0.0) \\
&= 5.6392 > 5 = k - 1
\end{aligned}$$

is a violated D_6^+ inequality at $\bar{x}$. □

4.3 Separation Procedures for 3-SECs and 4-SECs for Model B

Given that there is a polynomial number ($O(n^3)$ and $O(n^4)$) of 3-SECs and 4-SECs (see Table 3), the corresponding separation problem can be solved optimally by simply looping over all indices for each type of 3-SECs (2 types) and 4-SECs (6 types). Empirically we found that the implementation of 4-SECs had very little or no impact at all on the performance of the B&C algorithm.

4.4 Separation Procedures for LBMICs and UBMICs

From Section 3 we can see that LBMICs for both models can be expressed in the following form:

$$\alpha_{ijk} x_{jk} - y_{ik} \quad \leq \quad \beta_{ijk} \tag{11}$$

where α_{ijk} and β_{ijk} are constants depending on problem data for $i \in I$ and $(j,k) \in A$ ($\hat{A}$) for model A (B). Thus given a point $(\bar{x}, \bar{y})$, by looping over all possible index values i, j, k, we find the inequality such that

$$\alpha_{ijk} \bar{x}_{jk} - \bar{y}_{ik} - \beta_{ijk}$$

is maximized. This can be done in $O(mn^2)$ time.

Similarly, the UBMICs for both models can be expressed as

$$\gamma_{ijk} x_{jk} + y_{ik} - y_{ij} \quad \geq \quad \delta_{ijk}$$

where $\gamma_{ijk}, \delta_{ijk}$ are constants that depend on problem data. Again, by looping over all possible values of indices i, j, k the separation problem is solved exactly in $O(mn^2)$ time.

5 The Branch-and-Cut Method

Branch and cut (B&C) was introduced by Crowder and Padberg [5] who successfully solved large-scale instances of the well-known symmetric traveling salesman problem. It 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 many valid inequalities have been developed over the past 20 years. The SDST flowshop, however, has not been studied from a polyhedral perspective so one of our aims is to assess the effectiveness of B&C on this type of problem.

A typical B&B algorithm maintains a list of subproblems (nodes) 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 the algorithm selects a current subproblem from the list of unevaluated nodes. Typically 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 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 feasible 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.

Iterations are 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, the tighter the lower bound. This has a tremendous impact on the computational effort because it improves the chances that a node will be fathomed. Thus the corresponding portions of the search tree will not have to be evaluated. One way to improve the LP-representation of a given problem is by adding valid inequalities (cutting planes or cuts). B&C is the procedure developed to implement this idea.

Figure 5 shows a flow chart of our B&C algorithm which was coded within MINTO [11]. using many of its built-in features. To discuss the relevant steps of the algorithm, the following notation is used: Z_{lp} is the objective function value of the current subproblem's LP relaxation, Z_{best} is the objective function value of the best feasible solution known so far, and Z_{heur} is the objective function value of a feasible solution delivered by a heuristic.

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

Preprocess: After the data have been read in, this stage attempts to improve the original formulation by removing redundant constraints and applying some probing techniques. The underlying idea of probing [18] 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 candidates. Here we use a best-bound node selection strategy, which chooses the subproblem with the smallest lower bound.

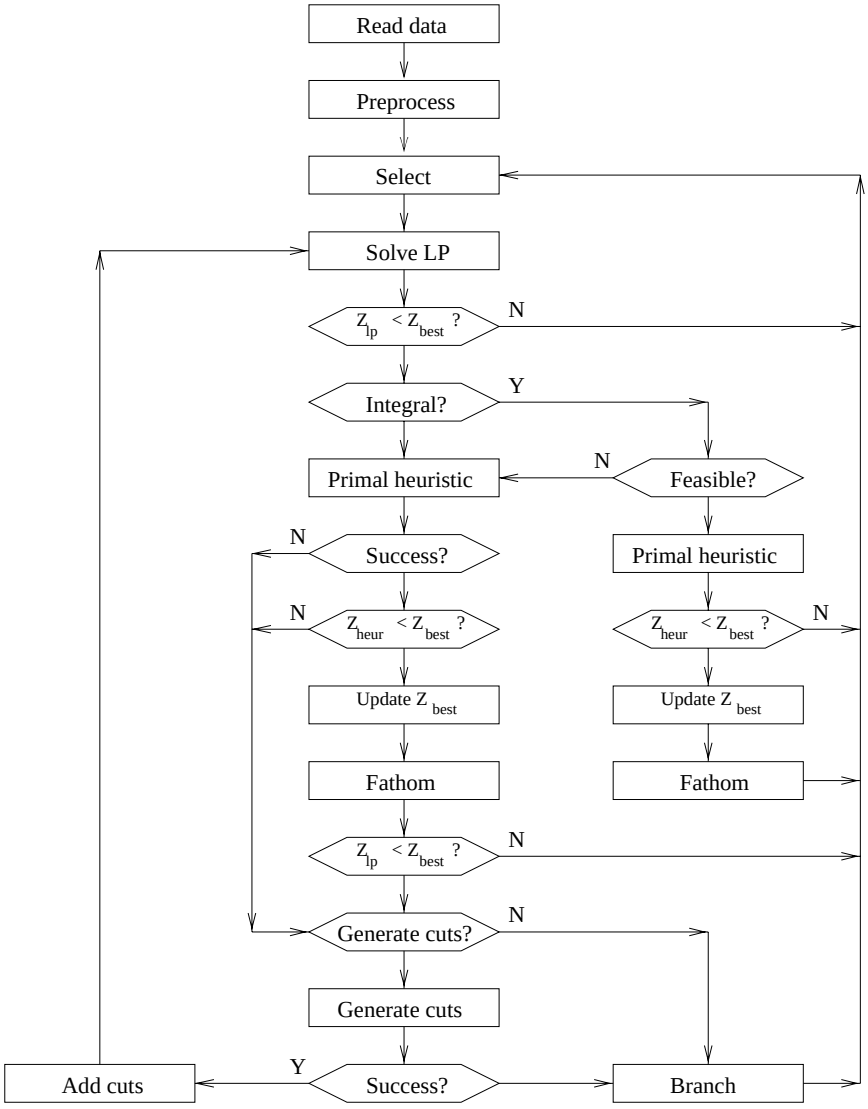


Figure 5: *Flow chart of the B&C algorithm*

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 we go back to the selection step. If the solution satisfies integrality and 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 step. Otherwise, we apply a heuristic in an attempt to find an integer feasible solution.

Primal heuristic: A heuristic is applied to see if it is possible to convert the current fractional solution to one that is integral. If successful, we update the current best global feasible solution (if $Z_{heur} < Z_{best}$), fathom the node, and go back to the selection step. In our implementation, we apply the SETUP heuristic

(discussed in [17]) to the root node to start with a good feasible solution and use MINTO's built-in heuristic thereafter (invoked every 25 nodes).

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 SECs (facet-inducing), D_k^+ and D_k^- inequalities (facet-inducing), LBMICs, and UBMICs for model A, and 3-SECs (facet-inducing), 4-SECs, UBMICs, and LBMICs for model B. 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. The way we determine the branching variable is by selecting 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. The 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, we apply the following stopping criteria: (i) relative gap percentage; i.e., stop when a global integer feasible solution is within $\rho\%$ of optimality, (ii) time limit, and (iii) number of evaluated nodes limit.

6 Computational Evaluation

For the purpose of evaluating the B&C approach, we embedded all algorithmic components discussed above in MINTO (Mixed INTeger Optimizer [11]). MINTO is a shell that facilitates the development of implicit enumeration and column generation optimization algorithms that rely on linear relaxations. The user can enrich its basic features by providing a variety of specialized application functions to achieve maximum efficiency for a problem class. CPLEX [4] was used 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 compiler CC, version 2.0.1, with the optimization flag set to -O. CPU times were obtained through MINTO. The code was validated by solving several 100- and 150-job, 1-machine instances to optimality. Recall that the 1-machine problem is an ATSP.

To conduct our experiments we used randomly generated data. It has been documented [10] that the main feature in real-world data for the SDST flowshop problem is the relationship between processing and setup times. In practice, the setup times are about 20-40% of the processing times. Because the experiments are expensive, we generated a single class of random

data sets with the setup times being 20-40% of the processing times: $p_{ij} \in [20, 100]$ and $s_{ijk} \in [20, 40]$.

6.1 Experiment 1: B&B vs. B&C

In the first experiment our aim was to compare B&B with B&C. While it is true that B&C provides a stronger LP-representation, 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 especially effective, the resulting lower bound improvement will be more than offset by the corresponding increase in computational effort. To make this comparison, we generated 5 instances for each machine combination $m \in \{2, 4, 6\}$ and $n \in \{7, 8\}$, with a stopping limit of 90 CPU minutes. In a preliminary experiment we determined the most effective cuts for each model within the B&C framework. The best performance was observed using SECs and UBMICs for model A, and 3-SECs and the UBMICs for model B. The remaining computations were made with these cuts only.

Instance size						Average performance			
$m \times n$	NC	NV(B)	NZ	Model	Method	Nodes	Cuts	LP rows	Time
2×7	114	71(56)	392	A	B&B	22687	0	114	10.1
				A	B&C	10091	129	236	6.7
	98	36(21)	280	B	B&B	11457	0	98	2.9
				B	B&C	7340	72	168	2.9
4×7	212	85(56)	672	A	B&B	21523	0	212	14.1
				A	B&C	9831	129	328	9.8
	196	50(21)	560	B	B&B	8392	0	196	3.6
				B	B&C	5261	73	266	3.4
6×7	310	99(56)	952	A	B&B	21635	0	310	20.2
				A	B&C	9864	132	435	14.1
	294	64(21)	840	B	B&B	9137	0	294	7.0
				B	B&C	5402	74	366	5.4

Table 4: *Performance of B&B and B&C on 7-job instances for models A and B*

Table 4 displays the results for models A and B for each machine instance. 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 cuts added (cuts), maximum number of rows in the LP (LP rows), and CPU time in minutes. All instances were solved to optimality.

As can be seen, even though the size of the LPs increases (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. For model A the average relative time savings with B&C are 51%, 44%, and 43%, in the 2-, 4- and 6-machine instances, respectively. For model B, we observe little difference for the 2-, and 4-machine instances. The B&C starts to have an effect, however, as the size of the instance gets large. This can be seen in the 6-machine instances where B&C results in a relative time savings of 31%.

Instance size					Average performance			
$m \times n$	NC	NV(B)	NZ	Method	Nodes	Cuts	LP rows	Time
2×8	128	45(28)	368	B&B	76096	0	128	61.4
				B&C	45072	114	138	46.0
4×8	256	61(28)	736	B&B	68579	0	256	68.6
				B&C	39149	116	366	55.3
6×8	384	77(28)	1104	B&B	59154	0	384	73.3
				B&C	34818	116	493	63.5

Table 5: *Comparison of B&B and B&C on 8-job instances for model B*

For model B, when we increase the number of jobs, B&C has a more pronounced impact. This can bee seen in Table 5 where the results for 8-job instances under model B are displayed. The B&C runs on average 33%, 24%, and 15%, faster than the B&B on the 2-, 4-, and 6-machine instances, respectively. Table 6 displays the results when model A was used. As can be seen, the algorithm was unable to solve the problem (after 90 minutes) under either B&B or B&C. However, the optimality gaps (shown in the last column) are smaller under the latter. The relative optimality gap in MINTO is computed as follows:

$$\frac{\text{best upper bound} - \text{best lower bound}}{\text{best upper bound}} \times 100\%$$

Instance size					Average performance				
$m \times n$	NC	NV(B)	NZ	Method	Nodes	Cuts	LP rows	Time	Gap (%)
2×8	146	89(72)	512	B&B	50637	0	146	90.0	50.3
				B&C	49090	276	354	90.0	38.3
4×8	274	105(72)	880	B&B	45329	0	274	90.0	43.6
				B&C	39825	289	487	90.0	37.5
6×8	402	121(72)	1248	B&B	42719	0	402	90.0	39.6
				B&C	32486	287	607	90.0	36.3

Table 6: *Comparison of B&B and B&C on 8-job instances for model A*

6.2 Experiment 2: Model A vs. Model B

In Section 2.5 we pointed out the trade-off between models A and B. On one hand, model A can benefit from a better structured underlying TSP. In contrast, model B is smaller, using

only about half the number of the binary variables used by model A.

By looking at the B&C rows for models A and B in Table 4 we can make a comparison of both models for 7-job instances. It can be seen that the size of the model (especially in terms of the number of binary variables) plays an important role. Computations are significantly better when model B is used. In fact, the effect is even more dramatic when we attempted to solve 8-job instances. By using model B, we were able to solve 8-job instances (Table 5) in an average of 46, 55.3, and 63.5 minutes of CPU for 2-, 4-, and 6-machines, respectively. When model A was used (Table 6), the algorithm stopped after 90 minutes with average optimality gaps of 38%, 37%, and 36%, respectively.

6.3 Experiment 3: Larger Instances

Instance size				Average performance			
$m \times n$	NC	NV(B)	NZ	Nodes	Cuts	LP rows	Gap (%)
2×10	200	66(45)	580	26428	241	412	34.8
4×10	400	86(45)	1160	20615	242	612	30.5
6×10	600	106(45)	1740	16453	241	812	26.7

Table 7: *Evaluation of B&C on 10-job instances for model B*

The last experiment assesses the limited scope of the polyhedral approach. Table 7 shows the average performance of B&C on 10-job instances with a 60-minute time limit for model B. We can see that the optimality gaps are 26-34%.

7 Conclusions

We provide empirical evidence that using model B with B&C yields better results on solving instances of the SDST flowshop problem. However, the fact that even with the development of valid inequalities we are still unable to solve instances with 10 or more jobs shows that LP-based enumeration methods are wanting. The polyhedral representation of the problem is still not strong enough. In fact, we made several attempts to improve the performance of the B&C algorithm, such as changing branching strategies, fixing variables in a preprocessing phase, and reduced cost fixing, but the improvements were not significant. This difficulty is inherent to the SDST flowshop (2 or more machines) since we were able to successfully solve 100- and 150-job instances restricted to the 1-machine case. Recall that minimizing the makespan in SDST flowshop is equivalent to finding the minimum length tour of an $(n+1)$-city ATSP when the number of machines is set equal to 1. It is evident that once we start adding machines, the ATSP structure starts to weaken. One explanation for this is that, unlike the ATSP where we are looking for a good sequence of nodes, it is difficult here to characterize fully what a good

sequence of jobs really is. What might be a good sequence for a certain machine, may be a bad sequence for the others. This makes this problem extremely nasty.

Further research is necessary to improve the polyhedral representation by developing more valid inequalities. We should point out that most of the valid inequalities developed in this work for the SDST flowshop can actually be applied to other scheduling problems involving sequence-dependent setup times. With some modifications, they could also be applied to the SDST flowshop when job ready times and/or due dates are present.

Alternatively, using non-LP-based enumeration schemes such as branch and bound and dynamic programming could lead to a substantial improvement, provided an effective and better lower bounding procedure (with respect to the LP relaxation lower bound) is developed. Such an approach is now being pursued.

8 Acknowledgments

We thank Manfred Padberg and Giovanni Rinaldi for providing us with a C implementation of their MINCUT algorithm [12].

References

[1] E. Balas. The asymmetric assignment problem and some new facets of the traveling salesman polytope on a directed graph. *SIAM Journal on Discrete Mathematics*, 2(4):425–451, 1989.

[2] E. Balas and M. Fischetti. The fixed-outdegree 1-arborescence polytope. *Mathematics of Operations Research*, 17(4):1001–1018, 1992.

[3] E. Balas and M. Fischetti. A lifting procedure for the asymmetric traveling salesman polytope and a large new class of facets. *Mathematical Programming*, 58(3):325–352, 1993.

[4] CPLEX Optimization, Inc., Incline Village, NV. *Using the CPLEX Callable Library, Version 4.0*, 1995.

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

[6] M. Fischetti. Facets of the asymmetric traveling salesman polytope. *Mathematics of Operations Research*, 16(1):42–56, 1991.

[7] M. Fischetti and P. Toth. A polyhedral approach for the exact solution of hard ATSP instances. *Management Science*, 1997. Forthcoming.

[8] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *SIAM Journal on Applied Mathematics*, 9:551–570, 1961.

[9] N. Grötschell and M. W. Padberg. Polyhedral theory. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan, and D. B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, pages 251–305. John Wiley & Sons, Chichester, 1985.

[10] 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.

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

[12] M. Padberg and G. Rinaldi. An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming*, 47(1):19–36, 1990.

[13] 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.

[14] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Englewood Cliffs, New Jersey, 1995.

[15] M. Queyranne and Y. Wang. Symmetric inequalities and their composition for asymmetric travelling salesman polytopes. *Mathematics of Operations Research*, 20(4):838–863, 1995.

[16] R. Z. Ríos-Mercado and J. F. Bard. The flowshop scheduling polyhedron with setup times. Technical Report ORP96–07, Graduate Program in Operations Research, University of Texas at Austin, Austin, TX 78712–1063, July 1996.

[17] R. Z. Ríos-Mercado and J. F. Bard. Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, 1997. Forthcoming.

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

[19] J. V. Simons Jr. Heuristics in flow shop scheduling with sequence dependent setup times. *OMEGA The International Journal of Management Science*, 20(2):215–225, 1992.

[20] 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.

[21] 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.