A Branch-and-Bound Algorithm for Permutation Flow Shops with Sequence-Dependent Setup Times

Roger Z. Ríos-Mercado
Department of Industrial Engineering
Texas A&M University
College Station, TX 77843–3131
roger@habanero.tamu.edu

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

November 1997
Revised August 1998, January 1999
Accepted January 1999

Abstract

This paper presents a branch-and-bound enumeration scheme for the makespan minimization of the permutation flow shop scheduling problem with sequence-dependent setup times. The algorithm includes the implementation of both lower and upper bounding procedures, a dominance elimination criterion, and special features such as a partial enumeration strategy. A computational evaluation of the overall scheme demonstrates the effectiveness of each component. Test results are provided for a wide range of problem instances.

Keywords: flow shop scheduling, sequence-dependent setup times, branch and bound, lower bounds, upper bounds, dominance rules

1 Introduction

In this paper, we address the problem of finding a permutation schedule of n jobs in an m-machine flow shop 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 regarded in the scheduling literature as the sequence-dependent setup time flow shop (SDST flow shop) 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 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 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.

In [17], we approached this problem from a polyhedral perspective; that is, we formulated the problem as a mathematical program using two different models and studied the convex hull of the set of feasible solutions. The motivation for that work was to attempt to exploit the underlying traveling salesman polytope. We developed several classes of valid inequalities and in [18], implemented them in a branch-and-cut (B&C) framework with limited success. The main difficulty was the weakness of the lower bound obtained from the linear programming (LP) relaxation. Despite efforts to improve the polyhedral representation of the SDST flow shop, the quality of the LP lower bound remained poor.

This motivated the investigation of a series of non-LP-based lower bounding procedures reported in this paper. By relaxing some machine requirements rather than the integrality conditions on the mixed-integer programming (MIP) formulations, alternate lower bounding procedures were devised. The first was a generalized lower bound (GLB) obtained by reducing the original m-machine problem to a 2-machine problem; the second was a machine-based lower bound (MBLB) obtained by reducing the original problem to a single machine problem. Both procedures were found to produce results that were tangibly better than the LP-relaxation lower bound, with the MBLB being more effective than the GLB. Extending these lower bounding procedures to handle partial schedules as well, enabled us to develop an effective branch-and-bound scheme.

The objective of this paper is to present and evaluate our enumeration algorithm for the SDST flow shop. This includes the development of lower bounding schemes, a dominance elimination rule, and upper bounding procedures. Our results indicate the effectiveness of the proposed algorithm when tested on a wide variety of randomly generated instances. We were able to find optimal solutions in about 50% of the instances tested, and near-optimal solutions in the others.

The rest of the paper is organized as follows. The most relevant work in the flow shop scheduling area is presented in Section 2. In Section 3, we introduce notation and formally define the problem. In Section 4, we give a full description of the branch-and-bound algorithm, followed in Section 5 by the presentation and evaluation of our computational experience. We conclude with a discussion of the results.

2 Related Work

In [1], Allahverdi et al. present an extensive review of machine scheduling research involving setup considerations. Another general review of flow shop scheduling (with and without setups), including computational complexity results, is given in [16].

2.1 Minimizing Makespan in Regular Flow Shops

The flow shop scheduling problem with no setups (denoted by $F||C_{\max}$) has been studied extensively over the past 25 years. Several exact optimization procedures, mostly based on branch and bound, have been proposed for this problem, including those of Lageweg et al. [10], Potts [15] and Carlier and Rebai [3]. The 3-machine case is considered by Ignall and Schrage [8] and Lomnicki [11], and the 2-machine case by Della Croce et al. [4].

2.2 Sequence-Dependent Setup Times

To the best of our knowledge, no effective methods to solve the SDST flow shop optimally have been developed to date. Efforts to solve this problem have been made by Srikar and Ghosh [22], and by Stafford and Tseng [23] 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 flow shop. The largest solved was a 6-machine, 6-job problem in about 22 minutes of CPU time on a Prime 550.

Subsequently, 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 flow shop problems based on the Srikar-Ghosh model.

In [17], we studied the polyhedral structure of the set of feasible solutions based on those models. We developed several classes of valid inequalities, and showed that some of them are indeed facets of the SDST flow shop polyhedral. In [18], a branch-and-cut scheme was implemented to test the effectiveness of the cuts. Even though we found B&C to provide better solutions than the previously published work which was based on straight branch and bound, we were still unable to solve (or provide a good assessment of the quality of the solutions in terms of the optimality gap) moderate to large size instances. The largest instance solved to optimality was a 6-machine, 8-job problem in about 60 minutes of CPU on a Sun Sparcstation 10. Other approaches have focused on heuristics [19, 20, 21], variations of the SDST flow shop, and work restricted to the 1- and 2-machine case (see [1] for an extensive review).

3 Statement of Problem

In the flow shop 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 flow shop 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., $s_{ijk} \neq s_{ikj}$, in general, for all indices. 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 can be done instantaneously 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. In the literature [14], this problem is denoted by $Fm|s_{ijk}, \ prmu|C_{\max}$ or SDST flow shop.

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	-

Here, $j = 0$ is a dummy job representing the initial state, so s_{i0k} is the time it takes to setup machine i when job k is scheduled first. A schedule $S = (3, 1, 2, 4)$ is shown in Figure 1. The corresponding makespan is 24, which is optimal. $\square$

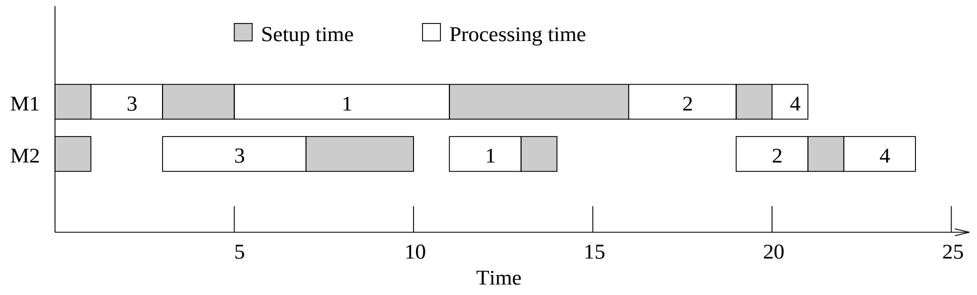

Figure 1: Example of a 2×4 SDST flow shop

3.1 Notation

In the reminder of the paper, when we refer to the SDST flow shop 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; $i \in I$, $j \in J_0$, $k \in J$

A job j (without brackets) refers to the job j itself, whereas job $[j]$ (with brackets) refers to the index of the job scheduled in the j-th position. In Section 4, indices i, j, k, l are used to represent entities of the search tree (subproblems, nodes).

4 Branch and Bound

The feasible set of solutions of the SDST flow shop problem from a combinatorial standpoint can be represented as $X = \{\text{set of all possible } n\text{-job schedules}\}$. This is a finite set so an optimal solution can be obtained by a straightforward method that enumerates all feasible solutions in X and then outputs the one with the minimum objective value. However, complete enumeration is hardly practical because the number of cases to be considered is usually enormous. Thus any effective method must be able to detect dominated solutions so that they can be excluded from explicit consideration.

A branch-and-bound (B&B) algorithm for a minimization problem has the following general characteristics:

- a *branching rule* that defines partitions of the set of feasible solutions into subsets
- a *lower bounding rule* that provides a lower bound on the value of each solution in a subset generated by the branching rule
- a *search strategy* that selects a node from which to branch

Additional features such as *dominance rules* and *upper bounding procedures* may also be present, and if fully exploited, could lead to substantial improvements in algorithmic performance.

A diagram representing this process is called an enumeration or search tree. In this tree, each node represents a subproblem P_i. The number of edges in the path to P_i is called the *depth* or *level* of P_i. The original problem P_0 is represented by the node at the top of the tree (root). In our case, the schedule S_0 associated with P_0 is the empty schedule.

The fundamentals of B&B can be found in Ibaraki [6, 7]. In this paper we limit the discussion to our proposed algorithm, BABAS (Branch-and-Bound Algorithm for Scheduling).

4.1 Branching Rule

The following branching rule is used in BABAS. Nodes at level k of the search tree correspond to initial partial sequences in which jobs in the first k positions have been fixed. More formally, each node (subproblem) of the search tree can be represented by P_j, with associated schedule S_j, where $S_j = ([1], \ldots, [k])$ is an initial partial sequence of k jobs. Let U_j denote the set of unscheduled jobs. Then, for $U_j \neq \emptyset$, an immediate successor of P_j has an associated schedule of the form $([1], \ldots, [k], l)$, where $l \in U_j$. Figure 2 illustrates this rule for a 4-job instance. Node P_1 represents a problem at level 1 of the enumeration tree; where only one job has been scheduled; i.e., $S_1 = (3)$.

4.2 Lower Bounds

We now develop two lower bounding procedures that turned out to be more effective than the linear programming relaxation lower bound. These procedures are based on machine completion times of partial schedules.

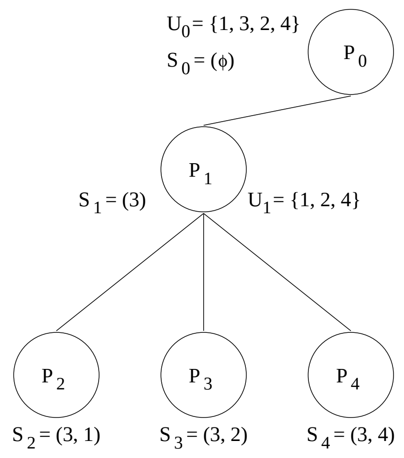

Figure 2: Illustration of the branching rule for a 4-job instance

Given a partial schedule S_i, let $\bar{S}_i$ denote a schedule formed by all unscheduled jobs. We shall now derive lower bounds on the value of the makespan of all possible completions $S_i\bar{S}_i$ of S_i, where $S_i\bar{S}_i$ represents the concatenation of jobs in S_i and $\bar{S}_i$. We shall be particularly concerned with the trade-off between the sharpness of a lower bound and its computational requirements. A stronger bound eliminates relatively more nodes of the search tree, but if its computational requirements become excessive, it may become advantageous to search through larger portions of the tree using a weaker bound that can be computed quickly.

Generalized Lower Bounds: The basic idea here is to obtain lower bounds by relaxing the capacity constraints on some machines, i.e., by assuming a subset of the machines to have infinite capacity. We know the only solvable case among flow shop problems is the 2-machine regular (no setups) flow shop (Johnson [9]). We also know that any problem involving three or more machines is likely to be $\mathcal{NP}$-hard. We therefore attempt to exploit this by reducing the m-machine problem to a 2-machine case. To pursue this, we arbitrarily choose two machines u and v, $1 \leq u < v \leq m$, and develop a lower bound g_{uv} by relaxing the capacity constraints on all machines except u and v. The development below shows how this lower bound can be reduced to the 2-machine case. This lower bound is similar to the one developed by Lageweg et al. [10] for $F||C_{\max}$.

Let the sequence of the first k jobs be $S_k = ([1], [2], \ldots, [k])$ and the set of remaining $n-k$ (unscheduled) jobs be U_k. Given S_k, the problem of determining an optimal sequence for the remaining jobs is called a subproblem of depth k and is represented by $\mathrm{FS}(S_k)$. Let $\bar{S}_k = ([k+1], [k+2], \ldots, [n])$ be an arbitrary sequence of jobs in U_k, and let $p_i(U_k) = \sum_{h \in U_k} p_{ih}$. Let $C_{i[j]}$ denote the completion time of job $[j]$ on machine i; $j \in J$, $i \in I$. Subproblem $\mathrm{FS}(S_k)$ is to determine the sequence $\bar{S}_k$ that minimizes $C_{\max}(S_k\bar{S}_k) \equiv C_{m[n]}$, the makespan of schedule $S_k\bar{S}_k$.

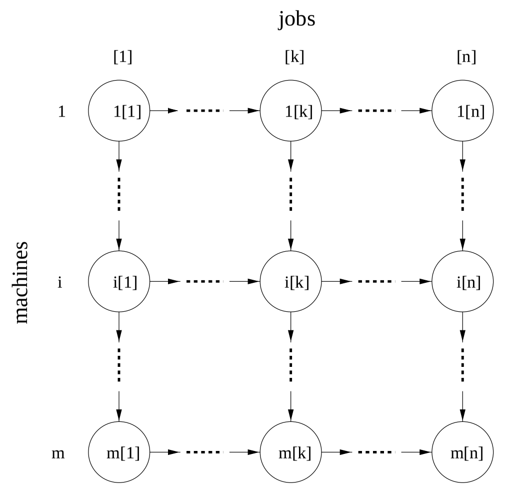

Figure 3: Directed graph G for SDST flow shops

To help understand the derivation of the lower bound consider the following directed graph G (depicted in Figure 3) which is constructed as follows: for each operation, say the processing of job $[j]$ on machine i, there is a node $(i[j])$ with a weight that is equal to $p_{i[j]}$. The setup times $s_{i[j][j+1]}$ are represented by an arc going from node $(i[j])$ to node $(i[j+1])$ with a weight that is equal to $s_{i[j][j+1]}$, for $i \in I$, $j = 1, \ldots, n-1$. Node $(i[j])$, $i = 1, \ldots, m-1$, $j = 1, \ldots, n$, also has an arc going to node $(i+1, [j])$ with zero weight. Note that nodes corresponding to machine m have only one outgoing arc, and that node $(m[n])$ has no outgoing arcs. This graph is a generalization of the graph model known for the classical flow shop problem [13].

Given a pair of machines (u, v), $u < v$, and a subsequence of jobs $S = ([j], [j+1], \ldots, [l])$, let $T_{uv}(S)$ be the elapsed time from the start of job $[j]$ on machine u until the finish of job l on machine v. It can be shown [16] that $T_{uv}(S)$ is equivalent to the maximum length path from node $(u[j])$ to node $(v[l])$ in G. Now let $R_{i[k]}$ denote the longest path (represented by a sequence of nodes) from node $1[1]$ to node $i[k]$ in G, $1 \leq i \leq u$, and let $r_{i[k]}$ denote the length of this path.

For a given pair of machines (u, v), $u < v$, consider the following paths:

$$(R_{i[k]}, i[k+1], \ldots, u[k+1], \ldots, u[t], \ldots, v[t], \ldots, v[n], \ldots, m[n])$$

for $1 \leq i \leq u$ and $k+1 \leq t \leq n$ in G (see Figure 4). It is easy to see that

$$\begin{aligned} C_{m[n]}(\bar{S}_k) \quad \geq \quad & \max_{1 \leq i \leq u} \left\{ r_{i[k]} + s_{i[k][k+1]} + \sum_{q=i}^{u-1} p_{q[k+1]} \right\} \\ & + \max_{k+1 \leq t \leq n} \left\{ \sum_{j=k+1}^{t-1} (p_{u[j]} + s_{u[j][j+1]}) + \sum_{i=u}^{v} p_{i[t]} + \sum_{j=t+1}^{n} (p_{v[j]} + s_{v[j-1][j]}) \right\} \\ & + \sum_{i=v+1}^{m} p_{i[n]} \end{aligned}$$

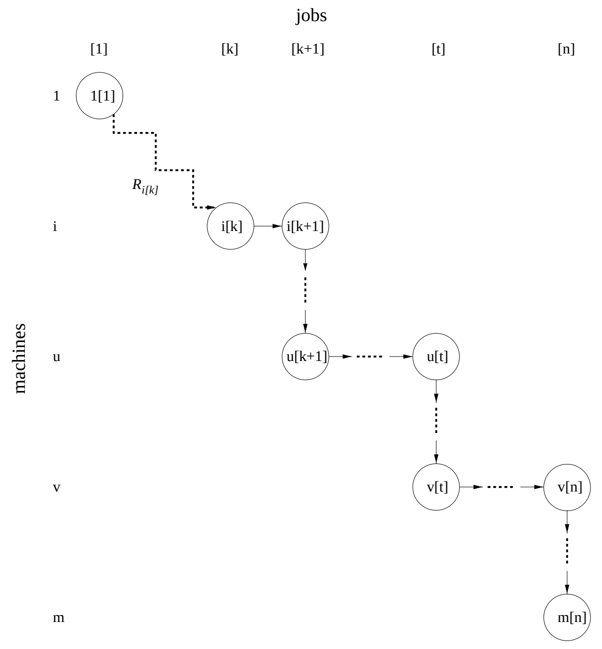

Figure 4: Critical path illustration for developing GLB

$$
\begin{aligned}
\geq\ & \max_{1\leq i\leq u}\left\{r_{i[k]}+\min_{h\in U_k}\left\{s_{i[k]h}+\sum_{q=i}^{u-1}p_{qh}\right\}\right\}\\
&+\max_{k+1\leq t\leq n}\left\{\sum_{j=k+1}^{t-1}p_{u[j]}+\sum_{i=u}^{v}p_{i[t]}+\sum_{j=t+1}^{n}p_{v[j]}\right\}\\
&+\sum_{j=k+2}^{n}\min\left\{s_{u[j-1][j]},s_{v[j-1][j]}\right\}+\min_{h\in U_k}\sum_{i=v+1}^{m}p_{ih} \qquad (1)
\end{aligned}
$$

Note that inequality (1) is valid for an arbitrary subsequence $\bar{S}_k$, but in fact, the main problem is to find the subsequence $\bar{S}_k^*$ in U_k that minimizes the left-hand side $C_{mn}(\bar{S}_k)$ in (1). As can be seen, minimizing the right-hand side (RHS) of (1) yields a lower bound on $C_{m[n]}(\bar{S}_k^*)$.

Now let us take a closer look at the minimization of the RHS of (1). The first and fourth terms do not depend on a particular subsequence, thus they can just be precomputed with no optimization involved. In the second term of the RHS of (1), we note that

$$\sum_{j=k+1}^{t-1}p_{u[j]}+\sum_{i=u}^{v}p_{i[t]}+\sum_{j=t+1}^{n}p_{v[j]} \;=\; \sum_{j=k+1}^{t}\sum_{i=u}^{v-1}p_{i[j]}+\sum_{j=t}^{n}\sum_{i=u+1}^{v}p_{i[j]}-\sum_{i=u+1}^{v-1}p_i(\bar{S}_k).$$

Let

$$Z_{uv}(\bar{S}_k) \;=\; \max_{k+1\leq t\leq n}\left\{\sum_{j=k+1}^{t}\left(\sum_{i=u}^{v-1}p_{i[j]}\right)+\sum_{j=t}^{n}\left(\sum_{i=u+1}^{v}p_{i[j]}\right)\right\}$$

The problem of minimizing $Z_{uv}(\bar{S}_k)$ is reduced to a solvable 2-machine flow shop (using Johnson's algorithm) with processing times

$$p'_{1j} = \sum_{i=u}^{v-1} p_{ij} \qquad\qquad p'_{2j} = \sum_{i=u+1}^{v} p_{ij}$$

Let $Z^*_{uv}(\bar{S}_k)$ denote its minimum value.

For the third term of the RHS of (1), let $s^{uv}_{[j-1][j]} = \min\{s_{u[j-1][j]}, s_{v[j-1][j]}\}$. It can be seen that the problem of minimizing $\sum_{j=k+2}^{n} s^{uv}_{[j-1][j]}$ corresponds to finding a shortest tour of an ATSP on $n-k$ vertices. Let $S^*_{uv}(\bar{S}_k)$ denote the length of this optimal tour.

We can thus establish the following generalized lower bound $g_{uv}(\bar{S}_k)$ on $C_{m[n]}(\bar{S}^*_k)$

$$\begin{aligned} g_{uv}(\bar{S}_k) \quad = \quad & \max_{1\le i\le u} \left\{ r_{i[k]} + \min_{h\in U_k} \left\{ s_{i[k]h} + \sum_{q=i}^{u-1} p_{qh} \right\} \right\} - \sum_{i=u+1}^{v-1} p_i(\bar{S}_k) + \min_{h\in U_k} \sum_{i=v+1}^{m} p_{ih} \\ & + Z^*_{uv}(\bar{S}_k) + S^*_{uv}(\bar{S}_k) \end{aligned}$$

for any $1 \le u < v \le m$. Note that the optimal sequence of the jobs in the embedded 2-machine flow shop (for given u, v) has to be determined only once for FS($\emptyset$), the original problem, since it does not change if some jobs are removed nor it is influenced by the fact that machine v is not available until $C_{v[k]}$.

In summary, for a given pair of machines (u, v), we have derived a generalized lower bound g_{uv} which may be computed for any two–machine combination. If $W = \{(u_1, v_1), \ldots, (u_w, v_w)\}$ is a set of machine pairs, then the corresponding overall lower bound GLB(W) is defined by

$$\text{GLB}(W) \quad = \quad \max\{g_{u_1,v_1}, \ldots, g_{u_w,v_w}\}\,.$$

Note that there are $m(m-1)/2$ possible pairs (u, v); however, the load for computing GLB based on all pairs is too heavy. Therefore, we only consider the following subsets of machine pairs $W_0 = \{(1,2), (2,3), \ldots, (m-1, m)\}$, $W_1 = \{(1, m), (2, m), \ldots, (m-1, m)\}$, and $W_2 = W_0 \cup W_1$, which contains $O(m)$ pairs. Our empirical work (Section 5) has shown that GLB(W_1) provides better results than GLB(W_0) and is faster to compute than GLB(W_2).

Machine-Based Lower Bounds: Above, we have developed a family of lower bounds g_{uv} for $1 \le u < v \le m$. Consider now the case $u = v$; that is, there is only one bottleneck machine and the capacity of all other machines is relaxed. Thus it is possible to find m additional lower bounds g_u, $1 \le u \le m$.

Again, let the sequence of the first k jobs fixed be $S_k = ([1], [2], \ldots, [k])$ and the set of remaining $n-k$ (unscheduled) jobs be U_k. Let $\bar{S}_k = ([k+1], [k+2], \ldots, [n])$ denote an arbitrary sequence of jobs in U_k. Then by proceeding in a similar fashion as we did in deriving GLB, we consider the paths: $(R_{i[k]}, i[k+1], \ldots, u[k+1], \ldots, u[n], \ldots, m[n])$ for $1 \le i \le u$ in G (see Figure 5). Thus we have

$$\begin{aligned} C_{m[n]}(\bar{S}_k) &\geq \max_{1\leq i\leq u}\left\{ r_{i[k]} + s_{i[k][k+1]} + \sum_{q=i}^{u-1} p_{q[k+1]} \right\} \\ &\quad + p_{u[k+1]} + \sum_{j=k+2}^{n} (p_{u[j]} + s_{u[j-1][j]}) + \sum_{i=u+1}^{m} p_{i[n]} \\ &\geq \max_{1\leq i\leq u}\left\{ r_{i[k]} + \min_{h\in U_k}\left\{ s_{i[k]h} + \sum_{q=i}^{u-1} p_{qh} \right\}\right\} \\ &\quad + p_u(\bar{S}_k) + \sum_{j=k+2}^{n} s_{u[j-1][j]} + \min_{h\in U_k} \sum_{i=u+1}^{m} p_{ih} \qquad (2) \end{aligned}$$

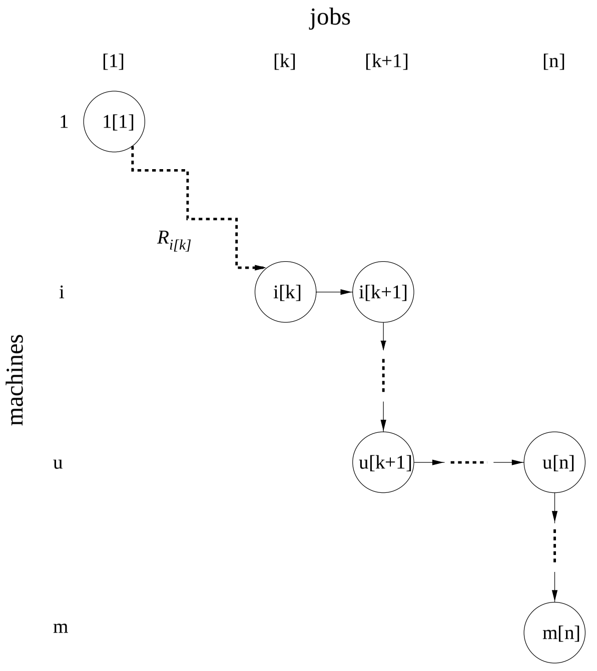

Figure 5: Critical path illustration for deriving MBLB

Since $p_u(\bar{S}_k)$ is constant for any sequence (just like the first and fourth terms in the RHS of (2)), the problem of minimizing the RHS corresponds to finding a sequence that minimizes $\sum_{j=k+2}^{n} s_{u[j-1][j]}$, which is equivalent to finding the shortest tour in an ATSP on $n-k$ vertices. Let $S_u^*(\bar{S}_k)$ be the optimal tour length for this ATSP. Then

$$g_u(\bar{S}_k) = \max_{1\leq i\leq u}\left\{ r_{i[k]} + \min_{h\in U_k}\left\{ s_{i[k]h} + \sum_{q=i}^{u-1} p_{qh} \right\}\right\} + p_u(\bar{S}_k) + S_u^*(\bar{S}_k) + \min_{h\in U_k} \sum_{i=u+1}^{m} p_{ih} \quad (3)$$

for $1 \leq u \leq m$ is a valid lower bound on $C_{\max}$.

The fact that the setup time between jobs $[k]$ and $[k+1]$, $s_{u[k][k+1]}$, is not considered in the computation of $g_u(\bar{S}_k)$ allows us to use the first term on the RHS of (3) as a lower bound for the starting time of job $[k+1]$ on machine u. It might be advantageous, however, to include this setup time ($s_{u[k][k+1]}$) in the computations to improve the lower bound S_u^* of the related ATSP. The trade-off is that by doing so, we no longer can use the first term on the RHS of (3). This alternate bound is expressed as

$$g_u'(\bar{S}_k) \quad = \quad C_{u[k]} + p_u(\bar{S}_k) + L_u^*(\bar{S}_k) + \min_{h \in U_k} \sum_{i=u+1}^{m} p_{ih}$$

where L_u^* is the optimal tour length $\sum_{j=k+1}^{n} s_{u[j-1][j]}$.

ATSP Lower Bounds: In deriving the GLB and MBLB, we have to deal with solving an ATSP at some point. The ATSP itself is an $\mathcal{NP}$-hard problem; however, since we are only interested in a lower bound, any valid lower bound for the ATSP will suffice.

In our work, we used the assignment problem (AP) lower bound, which is obtained by relaxing the connectivity (subtour elimination) constraints for the ATSP. It has been documented (Balas and Toth [2]) that the AP bound is very sharp for the ATSP. (This is not necessarily true for the symmetric TSP.)

4.3 Search Strategy

The search strategy we use selects the subproblem with the best bound; e.g., the smallest lower bound in the case of a minimization problem. This approach 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 than any other node. As shown in [6], this strategy has the characteristic that, if other parts of a branch-and-bound algorithm are not changed, the number of partial problems decomposed before termination is minimized.

Another well known strategy is depth-first search, which is mostly used in situations where it is important to find feasible solutions quickly. However, we do not consider it since feasibility is not an issue.

4.4 Dominance Rule

We now establish some conditions under which all completions of a partial schedule S_k (associated with subproblem P_k) can be eliminated because a schedule at least as good exists among the completions of another partial schedule S_j (corresponding to subproblem P_j). Let $J(S_j)$ and $J(S_k)$ denote the index sets of jobs corresponding to S_j and S_k, respectively; $l(S)$ denote the index of the last scheduled job in schedule S; and $C_i(S)$ denote the completion time of the last scheduled job in S on machine i. Then P_j dominates P_k if for any completion $S_k\bar{S}_k$ of S_k there exists a completion $S_j\bar{S}_j$ of S_j such that $C_{\max}(S_j\bar{S}_j) \leq C_{\max}(S_k\bar{S}_k)$. This is stated formally in the following theorem.

Theorem 1 *If $J(S_j) = J(S_k)$, $l(S_j) = l(S_k)$, and $C_i(S_j) \leq C_i(S_k)$ for all $i \in I$, then P_j dominates P_k.*

The proof can be found in [16] along with a number of other dominance rules for special cases. In terms of computational effort, determining whether a given subproblem P_k is dominated implies: (a) searching for another subproblem (at the same level), and (b) checking conditions of Theorem 1. Step (a) can be done in $O(\log T)$ time, where $T = O(2^d)$ is the size of search tree up to depth d (if done efficiently, there is no need to search the whole tree). Operation (b) takes $O(m)$ time. At level d, there are potentially $O(2^d)$ nodes, thus the worst-case complexity to determine whether a given subproblem (at depth d) is dominated is $O(md2^d)$.

Despite this worst-case complexity, the implementation of this dominance rule has had a strong positive impact in the performance of BABAS. Computational results are provided in Section 5.

4.5 Upper Bounds

It is well known that branch-and-bound computations can be reduced by using a heuristic to find a good solution to act as an upper bound prior to the application of the enumeration algorithm, as well as at certain nodes of the search tree. With this in mind we have adapted the GRASP developed in [19] and the hybrid heuristic described in [20] to handle partial schedules.

In our basic algorithm, we apply both heuristics with extensive local search at the root node to obtain a high quality feasible solution. Once the algorithm is started, an attempt is made to find a better feasible solution every time UB_FREQ nodes are generated, where UB_FREQ is a user-specified parameter. In our experiments, we set this parameter to 50, which means that the heuristic will be applied once every 50 nodes. Note that setting UB_FREQ = 1 implies that the heuristic is applied to every node in the enumeration tree. At the intermediate stages, we do not do a full local search but try to balance the computational load. Once BABAS satisfies the stopping criteria, if the best feasible solution is not optimal, we apply an extensive local search to ensure that a local minimum has been obtained.

4.6 Partial Enumeration

Partial enumeration is a truncated branch-and-bound procedure similar to what is called beam search [12]. Instead of waiting to discard a portion of the tree that is *guaranteed* not to contain the optimum, we may discard parts of the tree that are not *likely* to contain the optimum. One essential is to have a good measure of what "likely" means.

The way we handle the partial enumeration is as follows. During the branching process, every potential child is evaluated with respect to a *valuation function h*. Those potential subproblems whose valuation function do not meet a certain pre-established criterion are discarded.

We implemented this idea by ranking the potential children by increasing value of h and then discarding the worst ρn nodes, where $\rho \in [0,1]$ is a user-specified parameter. The larger the value of ρ, the more nodes that will be eliminated from consideration. The case $\rho = 0$ coincides with regular branch and bound.

4.6.1 A Valuation Function

To develop a valuation function h we make use of the following cost function C_{jk} for each pair of jobs $j, k \in J$:

$$C_{jk} = \theta R_{jk} + (1-\theta) S_{jk}$$

where $\theta \in [0,1]$ is a weight factor, R_{jk} is a term that penalizes a "bad" fit from the flow shop perspective, and S_{jk} is a term that penalizes large setup times. This cost measure was introduced in [20] where it was used to develop a TSP-based hybrid heuristic for the SDST flow shop with very good results. A detailed description on how to estimate R_{jk} and S_{jk} is given in that work.

Let P_j be the node from which branching is being considered with corresponding partial schedule S_j. Let $l(S_j)$ be the index of the last scheduled job in S_j. Then, for every $k \in U_j$, we compute $h(k) = C_{l(S_j),k}$ and then discard the worst ρn potential subproblems (in terms of $h(k)$).

Although it is likely that the nodes excluded by this procedure will not be in an optimal solution, no theoretical guarantee can be established. We should also point out the trade-off between higher confidence in the quality of the solution and smaller computational effort when ρ is set to smaller and larger values, respectively.

5 Computational Experience

All routines were written in C++ and run on a Sun Sparcstation 10 using the CC compiler version 2.0.1, with the optimization flag set to -O. CPU times were obtained through the C function `clock()`. To conduct our experiments we used randomly generated data. It has been documented [5] that the main feature in real-world data for this type of problem is the relationship between processing and setup times. In practice, setup times are about 20-40% of the processing times. Because the experiments are expensive, we generated one class of random data sets as follows: $p_{ij} \in [20, 100]$ and $s_{ijk} \in [20, 40]$.

5.1 Experiment 1: Lower Bounds

The lower bounding procedures developed in Section 4.2 were compared within the branch-and-bound enumeration framework. In our first experiment, the generalized lower bound (GLB) was evaluated for three different subsets of machine pairs.

$$\begin{aligned} W_0 &= \{(1,2),(2,3),\ldots,(m-1,m)\} \\ W_1 &= \{(1,m),(2,m),\ldots,(m-1,m)\} \\ W_2 &= W_0 \cup W_1 \end{aligned}$$

It is evident that GLB(W_2) will dominate the other two; however, it requires more computational effort.

	$m=4$			$m=6$		
	W_0	W_1	W_2	W_0	W_1	W_2
Average relative gap (%)	0.8	0.3	0.3	1.3	0.3	0.4
Average number of evaluated nodes (1000)	10.1	9.2	8.7	11.0	9.3	9.0
Average CPU time (min)	10.8	9.2	9.3	15.0	11.8	12.1
Optimal solutions found (%)	60	60	60	20	70	60

Table 1: Evaluation of GLB for 10-job instances

Table 1 shows the average results for 10-job problems with machine parameter $m = 4, 6$. Note that when $m = 2$, $W_0 = W_1 = W_2 = \{(1,2)\}$. The averages are taken over 10 instances with a stopping limit of 15 CPU minutes. The dominance rule is in effect as well. Each column shows the statistics for GLB based on W_0, W_1, and W_2, respectively. The relative gap is computed as

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

As can be seen, the quality of GLB(W_0) is inferior to the other two since a larger number of nodes has to be evaluated, resulting in larger execution times. In addition, under GLB(W_0), fewer optimal solutions are found in the allotted time (only 20% in the 6-machine instances as opposed to 60% using W_1 and W_2). When comparing GLB(W_1) and GLB(W_2), similar performance is observed in almost every statistic. In fact, GLB(W_1) was found to be slightly better than GLB(W_2). This implies that the extra effort used by GLB(W_2) (the dominant bound) is not paying off.

We now compare GLB(W_1) with the machine-based lower bound (MBLB). A stopping limit of 15 CPU minutes was similarly imposed. Table 2 shows the results of this comparison for 15-job instances. It can be seen from the table that the GLB is actually better at the root node; however, as branching takes place, the MBLB makes more progress providing, in almost all cases, a tighter bound. There were even some instances that were solved to optimality under the MBLB alone.

One possible explanation for this result is that the MBLB, for a given machine, takes into account all the involved setup times, whereas the GLB, in its attempt to reduce the problem

	$m = 2$		$m = 4$		$m = 6$	
	GLB(W_1)	MBLB	GLB(W_1)	MBLB	GLB(W_1)	MBLB
Average relative gap at root (%)	2.7	6.6	6.4	12.1	8.8	14.8
Average relative gap at termination (%)	2.2	3.1	4.1	2.9	5.3	3.1
Times best bound found (%)	40	60	30	80	0	100
Optimal solutions found (%)	30	60	0	50	0	10

Table 2: Lower bound comparison for 15-job instances

to a 2-machine case, loses valuable setup time information (recall that for a given machine pair (u, v), GLB uses $\min\{s_{ujk}, s_{vjk}\}$ to represent the setup time between jobs j and k). Because the MBLB procedure was uniformly better than the GLB scheme, we use it in the remainder of the experiments.

5.2 Experiment 2: Dominance Elimination Criterion

We now evaluate the effectiveness of the dominance rule. Table 3 shows the average statistics over 10 instances for number of machines $m = 2, 4, 6$. Each instance was run with a CPU time limit of 30 minutes and optimality gap tolerance of 0.0. The results for the algorithm with and without the dominance rule in effect are indicated by DR and NDR, respectively. As we can see, the implementation of the dominance rule has a significant impact on the overall algorithmic performance resulting in a considerably smaller number of nodes to be evaluated, and a factor of 2 reduction in CPU time. In fact, when the dominance rule was in effect, the algorithm found optimal solutions to all instances, as opposed to only 80% when the rule was not in effect.

	$m = 2$		$m = 4$		$m = 6$	
	NDR	DR	NDR	DR	NDR	DR
Average relative gap (%)	0.7	0.0	0.0	0.0	0.1	0.0
Average number of evaluated nodes	16063	8529	5074	2985	10879	7924
Average CPU time (min)	18.3	5.8	4.8	2.3	14.2	8.4
Optimal solutions found (%)	50	100	100	100	90	100

Table 3: Evaluation of dominance rule for 10-job instances

5.3 Experiment 3: Partial Enumeration

In this experiment, we illustrate the effect of doing partial versus complete enumeration. We ran the partial search strategy for $\rho = 0$ (normal enumeration), $\rho = 0.5$ (truncating 50% of the potential children), and $\rho = 0.8$ (truncating 80% of the potential children) for 10, 6×20 instances, with a stopping criterion of 30 minutes and relative gap fathoming tolerance of 1.0%.

The overall results are displayed in Table 4. Results for a particular instance are by row. For each value of ρ we tabulate upper bound (UB), relative gap percentage (Gap) and CPU time (Time) rounded to the nearest minute. It should be noted that the relative gap for the truncated versions ($\rho \in \{0.5, 0.8\}$) shown in the table do not correspond to a true optimality gap of the original problem, but to the best lower bound of the restricted problem (i.e., the one not considering the truncated nodes). As can be seen, increasing the value of ρ results in a larger number of truncated nodes, hence a quicker execution of the procedure. We can also observe that the quality of the solution decreases with the size of ρ. A good compromise seems to be around $\rho = 0.5$, but one must keep in mind that once ρ assumes a value greater than zero, the algorithm can no longer be guaranteed to provide an optimal solution to the original problem.

	$\rho = 0$			$\rho = 0.5$			$\rho = 0.8$		
Instance	UB	Gap	Time	UB	Gap	Time	UB	Gap	Time
fs6x20.1	2022	2.8	30	2020	1.8	30	2029	1.0	1
fs6x20.2	2108	4.4	30	2111	3.2	30	2114	1.0	1
fs6x20.3	2100	5.3	30	2093	4.1	30	2106	1.0	1
fs6x20.4	1967	5.5	30	1966	3.5	30	1972	1.0	1
fs6x20.5	2095	1.5	30	2094	1.0	10	2096	1.0	1
fs6x20.6	2058	6.5	30	2057	5.3	30	2070	1.0	2
fs6x20.7	2088	5.6	30	2082	3.9	30	2088	1.0	2
fs6x20.8	2129	8.1	30	2129	6.8	30	2124	1.0	8
fs6x20.9	2106	3.7	30	2106	2.3	30	2109	1.0	1
fs6x20.10	2142	6.1	30	2130	4.2	30	2144	1.0	2

Table 4: Partial enumeration evaluation for 6-machine, 20-job instances

5.4 Experiment 4: BABAS Overall Performance

Here we show the results when the full algorithm is applied to instances of the SDST flow shop. We use the MBLB procedure, dominance elimination rule, and a relative gap fathoming tolerance of 1%. Maximum CPU time is set at 30 minutes.

Table 5 displays the summary statistics which were calculated from 10 problem instances for each $m \times n$ combination. As can be seen, all 10-job instances were solved (within 1%) in an average time of less than 5 minutes, a notable improvement when compared to previous published research on this problem, where the size of the largest instances solved optimally was a 6-machine, 8-job problem. In fact, BABAS was able to solve 43% of the 15-job instances, and 23% of the 20-job instances. Most of the instances solved corresponded to the 2-machine case. This is to be expected since the fathoming rules (lower bound and dominance) become less powerful as the number of machines increases.

Size	Optimality gap (%)			Time (sec)			Instances
$m \times n$	best	average	worst	best	average	worst	solved (%)
2×10	0.3	0.9	1.0	1	235	560	100
4	0.8	0.9	1.0	2	68	222	100
6	0.9	1.0	1.0	29	265	450	100
2×15	0.0	1.0	2.6	3	725	1800	70
4	0.9	2.2	4.5	7	1074	1800	50
6	1.0	2.9	4.5	38	1624	1800	10
2×20	0.5	1.0	1.6	7	1298	1800	70
4	2.4	4.2	5.1	1800	1800	1800	0
6	1.5	5.0	8.1	1800	1800	1800	0

Table 5: BABAS evaluation

Finally, Table 6 shows the algorithmic performance when BABAS is applied to 100-job instances. Thirty percent of the 2-machine instances were solved and 70% finished with a relative gap of 1.3% or better. In general, the average relative gap from the start to the end of the algorithm improved by 2.0%, 0.9%, and 1.6% for the 2-, 4-, and 6-machine instances, respectively. We also observed that the lower bound and the dominance test was less powerful than the 20 or fewer job cases.

Size	Optimality gap at root (%)			Optimality gap at end (%)			Average	Instances
$m \times n$	best	average	worst	best	average	worst	time (min)	solved (%)
2×100	1.2	3.4	8.4	0.6	1.4	2.1	28.1	30
4	3.3	5.1	6.5	2.3	4.2	5.7	30.0	0
6	5.0	7.6	9.4	4.3	6.0	7.2	30.0	0

Table 6: BABAS evaluation on 100-job instances

6 Summary

We have presented and evaluated a branch-and-bound scheme for the SDST flow shop scheduling problem. Our implementation includes both lower and upper bounding procedures, and a dominance elimination criterion. The empirical results demonstrate the relative effectiveness of the machine-based lower bound procedure and the dominance rule. Significantly better performance over previously published work (LP-based methods) was also obtained. We were able to solve (within 1% optimality gap) 100%, 43%, and 23% of the 10-, 15-, and 20-job instances tested. In addition, for the 100-job instances, our algorithm delivered average relative gaps of 1.4%, 4.2%, and 6.0% when applied to the 2-, 4-, and 6-machine cases, respectively. A salient feature of our algorithm is that it permits partial enumeration search, which can be used to obtain approximate solutions with reasonable computational effort.

7 Acknowledgments

The research of Roger Ríos-Mercado was partly supported by the Mexican National Council of Science and Technology (CONACYT) and by an E. D. Farmer Fellowship and a David Bruton Fellowship from The University of Texas at Austin. Jonathan Bard was supported by a grant from the Texas Higher Education Coordinating Boards' Advanced Research Program.

We thank Matthew Saltzman for allowing us to use his C implementation of the dense shortest augmenting path algorithm to solve AP.

This paper also benefit from the comments and suggestions of two anonymous referees.

References

[1] A. Allahverdi, J. N. D. Gupta, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27(2):219–239, 1999.

[2] E. Balas and P. Toth. Branch and bound methods. 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 361–401. John Wiley & Sons, Chichester, 1990.

[3] J. Carlier and I. Rebai. Two branch and bound algorithms for the permutation flow shop problem. *European Journal of Operational Research*, 90(2):238–251, 1996.

[4] F. Della Croce, V. Narayan, and R. Tadei. Two-machine total completion time flow shop problem. *European Journal of Operational Research*, 90(2):227–237, 1996.

[5] 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.

[6] T. Ibaraki. Enumerative approaches to combinatorial optimization: Part I. *Annals of Operations Research*, 10(1–4):1–340, 1987.

[7] T. Ibaraki. Enumerative approaches to combinatorial optimization: Part II. *Annals of Operations Research*, 11(1–4):341–602, 1987.

[8] E. Ignall and L. Schrage. Application of the branch and bound technique to some flow-shop scheduling problems. *Operations Research*, 13(3):400–412, 1965.

[9] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[10] B. J. Lageweg, J. K. Lenstra, and A. H. G. Rinnooy Kan. A general bounding scheme for the permutation flow-shop problem. *Operations Research*, 26(1):53–67, 1978.

[11] Z. A. Lomnicki. A "branch-and-bound" algorithm for the exact solution of the three-machine scheduling problem. *Operational Research Quarterly*, 16(1):89–100, 1965.

[12] T. E. Morton and D. W. Pentico. *Heuristic Scheduling Systems.* Wiley, New York, 1993.

[13] E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1):160–175, 1996.

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

[15] C. N. Potts. An adaptive branching rule for the permutation flow-shop problem. *European Journal of Operational Research*, 5(1):19–25, 1980.

[16] R. Z. Ríos-Mercado. *Optimization of the Flowshop Scheduling Problem with Setup Times.* PhD thesis, University of Texas, Austin, August 1997.

[17] 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, Austin, TX 78712–1063, July 1996.

[18] R. Z. Ríos-Mercado and J. F. Bard. Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. *Computers & Operations Research*, 25(5):351–366, 1998.

[19] R. Z. Ríos-Mercado and J. F. Bard. Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, 110(1):76–98, 1998.

[20] R. Z. Ríos-Mercado and J. F. Bard. An enhanced TSP-based heuristic for makespan minimization in a flow shop with setup times. *Journal of Heuristics*, 1999. Forthcoming.

[21] J. V. Simons Jr. Heuristics in flow shop scheduling with sequence dependent setup times. *Omega*, 20(2):215–225, 1992.

[22] 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.

[23] 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.