A Branch-and-Bound Algorithm for Flowshop Scheduling with Setup Times

Roger Z. Ríos-Mercado[1]
Graduate Program in Operations Research
University of Texas at Austin
Austin, TX 78712–1063
roger@bajor.me.utexas.edu

Jonathan F. Bard[2]
Graduate Program in Operations Research
University of Texas at Austin
Austin, TX 78712–1063
jbard@mail.utexas.edu

May 1997

[1]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

[2]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-bound enumeration scheme for the makespan minimization of the flowshop scheduling problem with 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: flowshop scheduling, setup times, branch-and-bound, lower bounds, dominance rules, upper bounds

1 Introduction

In this paper, we address the problem of finding a permutation schedule of n jobs in an m-machine flowshop environment that minimizes the maximum completion time $C_{\max}$ of all jobs, also known as the makespan. The jobs are available at time zero and have sequence-dependent setup times on each machine. All parameters, such as processing and setup times, are assumed to be known with certainty. This problem is regarded 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 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 flowshop, 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 flowshop. This includes the development of lower bounding schemes, a dominance elim-

ination 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 flowshop 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

For an excellent review of flowshop scheduling in general, including computational complexity results, see [16]. For an overview on complexity results and optimization and approximation algorithms involving single-machine, parallel machines, open shops, job shops, and flowshop scheduling problems, see [11].

2.1 Minimizing Makespan in Regular Flowshops

The flowshop scheduling problem (with no setups) has been studied extensively over the past 25 years. Several exact optimization schemes, mostly based on branch and bound, have been proposed for $F||C_{\max}$ including those of Lageweg et al. [10], Potts [15] and Carlier and Rebai [2]. The 3-machine case is considered by Ignall and Schrage [8] and Lomnicki [12]. Della Croce et al. [3] present a branch-and-bound approach for the 2-machine case.

2.2 Sequence-Dependent Setup Times

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 [21], and by Stafford and Tseng [22] 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.

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 flowshop 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 (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 its optimality gap) moderate to large size instances. The largest instance solved to optimality (in about 60 minutes of CPU on a Sun workstation) was a 6-machine, 8-job problem.

Other approaches have focused on heuristics [19, 20] and variations of the SDST flowshop. For example, Gupta [5] presents a branch-and-bound algorithm for the case where the objective is to minimize the total machine setup time. No computational results are reported. All other work has been restricted to the 1- and 2-machine case.

3 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 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 flowshop.

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. □

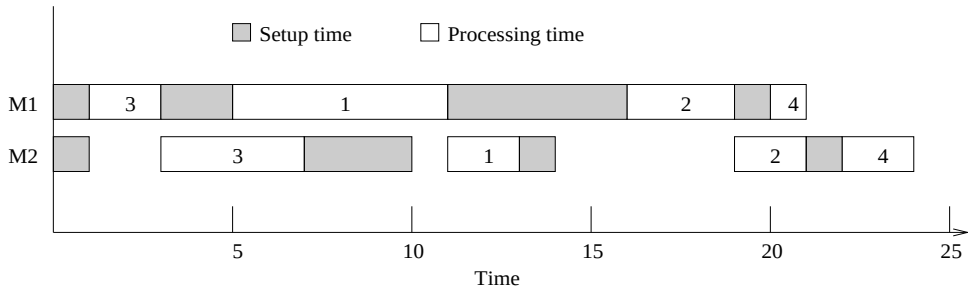

Figure 1: *Example of a* 2×4 *SDST flowshop*

3.1 Notation

In the reminder of the paper, when we refer to the SDST flowshop 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 flowshop 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_k, with associated schedule S_k, where $S_k = ([1], \ldots, [k])$ is an initial partial sequence of k jobs. Let U_k denote the set of unscheduled jobs. Then, for $U_k \neq \emptyset$, an immediate successor of P_k has an associated schedule of the form $S_j = ([1], \ldots, [k], j)$, where $j \in U_k$. 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.

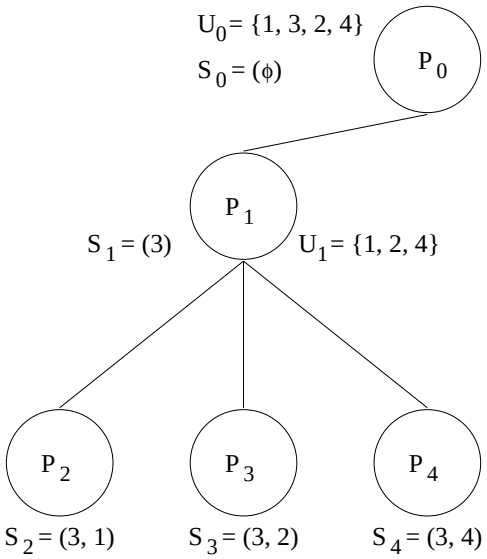

Figure 2: *Illustration of the branching rule for a 4-job instance*

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. The only solvable case among flowshop problems is the 2-machine regular (no setups) flowshop (Johnson [9]). We know that any problem involving three or more bottleneck machines is likely to be $\mathcal{NP}$-hard. We therefore restrict ourselves to choosing at most two machines u and v, $1 \leq u < v \leq m$, to be bottleneck machines. For any given pair (u, v) we now 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.

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 FS(S_k). Let $\bar{S}_k = ([k+1], [k+2], \ldots, [n])$ be an arbitrary sequence of jobs in U_k, and let $p_i(U_k) = \sum_{h \in U_k} p_{ih}$. Thus the completion time $C_{i[n]}$ of job $[n]$ on machine i can be derived as follows.

$$C_{1[n]} = C_{1[k]} + \sum_{h=k+1}^{n} s_{1[h-1][h]} + p_1(U_k)$$

$$
\begin{aligned}
C_{2[n]} &= \max\left\{C_{2[k]} + \sum_{h=k+1}^{n} s_{2[h-1][h]} + p_2(U_k),\ C_{1[k]} + s_{1[k][k+1]} + T_{12}(\bar{S}_k)\right\} \\
&\vdots \\
C_{m[n]} &= \max\Bigg\{C_{m[k]} + \sum_{h=k+1}^{n} s_{m[h-1][h]} + p_m(U_k),\ C_{m-1[k]} + s_{m-1[k][k+1]} + T_{m-1,m}(\bar{S}_k), \\
&\qquad \ldots,\ C_{1[k]} + s_{1[k][k+1]} + T_{1m}(\bar{S}_k)\Bigg\} \qquad (1)
\end{aligned}
$$

where $T_{uv}(\bar{S}_k)$ is the elapsed time from the start of job $[k+1]$ on machine u until the finish of job $[n]$ on machine v. Subproblem 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 $S_k\bar{S}_k$.

The definition of $T_{uv}(\bar{S}_k)$ is consistent with subsequences of $\bar{S}_k$, that is, $T_{uv}(([k+1],\ldots,[j]))$ is the elapsed time from the start of job $[k+1]$ on machine u until the finish of job $[j]$ on machine v, for $k+1 \le j \le n$. Thus $T_{uv}(([k+1],\ldots,[j]))$ can be recursively computed as follows:

$$
\begin{aligned}
T_{uu}(([k+1])) &= p_{u[k+1]} \\
T_{uw}(([k+1])) &= \sum_{i=u}^{w} p_{i[k+1]} && w = u+1,\ldots,v \\
T_{uu}(([k+1],\ldots,[j])) &= p_{u[k+1]} + \sum_{h=k+2}^{j} s_{u[h-1][h]} + p_{u[h]} && j = k+1,\ldots,n \\
T_{uw}(([k+1],\ldots,[j])) &= \max\Big\{T_{uw}(([k+1],\ldots,[j-1])) + s_{w[j-1][j]}, \\
&\qquad T_{u,w-1}(([k+1],\ldots,[j]))\Big\} + p_{w[j]} && j = k+1,\ldots,n, \\
&&& w = u+1,\ldots,v
\end{aligned}
$$

There is an alternate way to look at this recursion. To help understand the computations we introduce the following directed graph G_{uv} (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 $p_{i[j]}$. For each machine i, $i \in \{u, u+1,\ldots,v-1,v\}$, there is a node $(i[k+1])$ that represents the initial or current state (job $[k+1]$ is the first job in $\bar{S}_k$). 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 $i = u, u+1,\ldots,v-1,v$, $j = k+1,\ldots,n-1$. Node $(i[j])$, $i = u, u+1,\ldots,v-1$, $j = k+1,\ldots,n-1$, also has an arc going to node $(i+1,[j])$ with zero weight. Note that nodes corresponding to machine v have only one outgoing arc, and that node $(v[n])$ (target) has no outgoing arcs. The following proposition establishes the relationship between $T_{uv}(\bar{S}_k)$ and the critical path of G_{uv}.

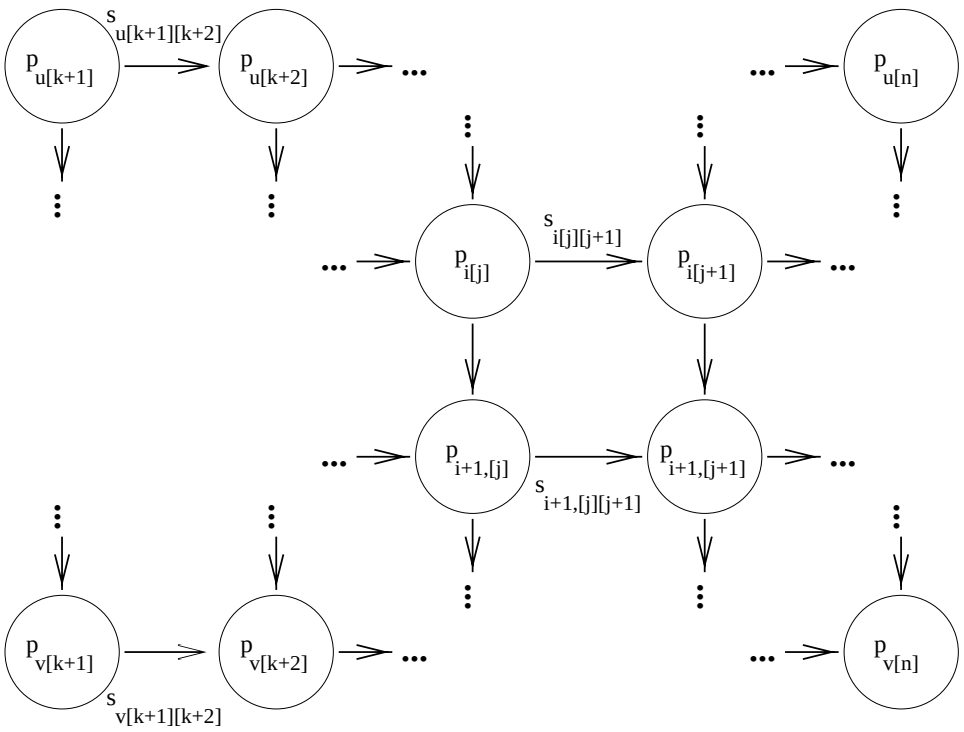

Figure 3: *Directed graph G_{uv} for computation of T_{uv} in a SDST flwoshop*

Proposition 1 *$T_{uv}(\bar{S}_k)$, with $\bar{S}_k = ([k+1], \ldots, [n])$, is determined by the maximum length or critical path from node $(u[k+1])$ to node $(v[n])$.*

Proof: The proof is by induction on $w + j$ (second machine index and job index of last job in subsequence $([k+1], \ldots, [j])$. The trivial case $w + j = u + k + 1$ corresponds to $w = u$ and $j = k + 1$ and is easily verified (only source node $(u[k+1])$ involved with length $T_{uu}(([k+1]))$.

The induction hypothesis assumes that $T_{uw}(([k+1], \ldots, [j]))$ is the maximum length path from node $(u[k+1])$ to node $(w[j])$ holds for $w + j < i + l$. It remains to prove that this result holds for $w + j = i + l$ as well.

Consider $T_{ui}(([k+1], \ldots, [l]))$ given by

$$T_{ui}(([k+1], \ldots, [l])) = \max\left\{T_{ui}(([k+1], \ldots, [l-1])) + s_{i[l-1][l]},\ T_{u,i-1}(([k+1], \ldots, [l]))\right\} + p_{i[j]}$$

Since each of the T_{uv} in the maximization above has $w + j = i + l - 1 < i + 1$, by the induction hypothesis, those represent maximum length paths from source node to node $(i[l-1])$ and $(i-1, [l])$, respectively. Since these are the only two nodes preceding node $(i[l])$, it follows that $T_{ui}(([k+1], \ldots, [l]))$ is the maximum length path from the source to node $(i[l])$ and the result is established. ■

Given the structure of G_{uv}, the length of the critical path from $(u[k+1])$ to $(v[n])$ (or equivalently, $T_{uv}(\bar{S}_k)$) is also given by

$$
\begin{aligned}
T_{uv}(\bar{S}_k) \;=\; & \max_{k<t_u\le t_{u+1}\le\ldots\le t_{v-1}\le t_v\le n}\Bigg\{p_{u[k+1]}+\sum_{h=k+2}^{t_u}\left(s_{u[h-1][h]}+p_{u[h]}\right)\\
& +p_{u+1[t_u]}+\sum_{h=t_u+1}^{t_{u+1}}\left(s_{u+1[h-1][h]}+p_{u+1[h]}\right)\\
& +\ldots\\
& +p_{v-1[t_u]}+\sum_{h=t_{v-1}+1}^{t_v}\left(s_{v-1[h-1][h]}+p_{v-1[h]}\right)\\
& +p_{v[t_v]}+\sum_{h=t_v+1}^{n}\left(s_{v[h-1][h]}+p_{v[h]}\right)\Bigg\} \qquad (2)
\end{aligned}
$$

for $1 \le u < v \le m$, where $\sum_{h=a}^{b}(\cdot) = 0$ for $b < a$. Thus the maximization in (2) consists of finding the $t_u, t_{u+1}, \ldots, t_{v-1}, t_v$ that define the critical path on G_{uv}, where t_i corresponds to the index of the job where the critical path crosses from level i to level $i+1$ on G_{uv}.

Recall that the maximization on the right-hand side of (2) is only used to find the T_{uv} for a given sequence $\bar{S}_k$, but in fact, the main problem is to find the subsequence $\bar{S}_k$ in U_k that minimizes $C_{m[n]}$ in (1). As can be seen from (1), minimizing $T_{uv}(\bar{S}_k)$ yields a lower bound on $C_{m[n]}$.

The minimization of $T_{uv}(\bar{S}_k)$ is as hard as the problem FS(S_k) (minimizing $C_{m[n]}$ in (1)), even for $T_{u,u+1}(\bar{S}_k)$. Hence we consider the minimization of the following lower bound of $T_{uv}(\bar{S}_k)$ by considering the case where $k < t_u = t_u + 1 = \ldots = t_{v-1} = t_v = t \le n$ and excluding all other terms in $T_{uv}(\bar{S}_k)$ (note that this is a valid lower bound since this special case corresponds to a path with length less than or equal to the length of the critical path), i.e.,

$$
\begin{aligned}
T_{uv}(\bar{S}_k) \;\ge\; & \max_{k<t\le n}\Bigg\{p_{u[k+1]}+\sum_{h=k+2}^{t}\left(s_{u[h-1][h]}+p_{u[h]}\right)+p_{u+1[t]}+\ldots\\
& +p_{v-1[t]}+p_{v[t]}+\sum_{h=t_v+1}^{n}\left(s_{v[h-1][h]}+p_{v[h]}\right)\Bigg\}\\
= \; & \max_{k<t\le n}\Bigg\{\sum_{h=k+1}^{t}p_{u[h]}+p_{u+1[t]}+\ldots+p_{v-1[t]}+\sum_{h=t}^{n}p_{v[h]}\\
& +\sum_{h=k+2}^{t}s_{u[h-1][h]}+\sum_{h=t+1}^{n}s_{v[h-1][h]}\Bigg\}\\
= \; & \max_{k<t\le n}\Bigg\{\sum_{h=k+1}^{t}p_{u[h]}+\sum_{h=k+1}^{t}p_{u+1[h]}+\ldots+\sum_{h=k+1}^{t}p_{v-1[h]}\\
& +\sum_{h=t}^{n}p_{u+1[h]}+\ldots+\sum_{h=t}^{n}p_{v-1[h]}+\sum_{h=t}^{n}p_{v[h]}
\end{aligned}
$$

$$
\begin{aligned}
& - \sum_{h=k+1}^{n} p_{u+1[h]} - \ldots - \sum_{h=k+1}^{n} p_{v-1[h]} \\
& + \sum_{h=k+2}^{t} s_{u[h-1][h]} + \sum_{h=t+1}^{n} s_{v[h-1][h]} \Bigg\} \\
= & \max_{k<t\leq n} \left\{ \sum_{h=k+1}^{t} \left(\sum_{i=u}^{v-1} p_{i[h]} \right) + \sum_{h=t}^{n} \left(\sum_{i=u+1}^{v} p_{i[h]} \right) \right. \\
& \left. + \sum_{h=k+2}^{t} s_{u[h-1][h]} + \sum_{h=t+1}^{n} s_{v[h-1][h]} \right\} - \sum_{i=u+1}^{v-1} p_i(\bar{S}_k) \\
\geq & \max_{k<t\leq n} \left\{ \sum_{h=k+1}^{t} \left(\sum_{i=u}^{v-1} p_{i[h]} \right) + \sum_{h=t}^{n} \left(\sum_{i=u+1}^{v} p_{i[h]} \right) \right\} \\
& + \sum_{h=k+2}^{n} s^{uv}_{[h-1][h]} - \sum_{i=u+1}^{v-1} p_i(\bar{S}_k)
\end{aligned}
$$

where $s^{uv}_{[h-1][h]} = \min\{s_{u[h-1][h]}, s_{v[h-1][h]}\}$. Let

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

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

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

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

The problem of minimizing $\sum_{h=k+2}^{n} s^{uv}_{[h-1][h]}$ corresponds to finding a shortest tour of an ATSP on $n-k$ vertices. Let $S^*_{uv}(\bar{S}_k)$ be a lower bound for this ATSP. Then

$$
T_{uv}(\bar{S}_k) \geq Z^*_{uv}(\bar{S}_k) + S^*_{uv}(\bar{S}_k) - \sum_{i=u+1}^{v-1} p_i(\bar{S}_k) \qquad 1 \leq u < v \leq m
$$

Now note the following valid lower bounds for the starting time of job $[k+1]$ on machine u

$$
\begin{aligned}
& C_{u[k]} + \min_{h\in U_k} \left\{ s_{u[k]h} \right\} \\
& C_{u-1[k]} + \min_{h\in U_k} \left\{ s_{u-1[k]h} + p_{u-1,h} \right\} \\
& C_{u-2[k]} + \min_{h\in U_k} \left\{ s_{u-2[k]h} + p_{u-2,h} + p_{u-1,h} \right\} \\
& \vdots
\end{aligned}
$$

$$C_{1[k]} + \min_{h \in U_k} \left\{ s_{1[k]h} + p_{1h} + \ldots p_{u-1,h} \right\}$$

Denote by $T^{min}_{i,u-1}$ the minimum elapsed time (among all unscheduled jobs) from the finish of job $[k]$ on machine i until the finish time of job $[k+1]$ on machine $u-1$, for $i = 1, \ldots, u$, i.e.,

$$T^{min}_{i,u-1} \quad = \quad \min_{h \in U_k} \left\{ s_{i[k]h} + \sum_{q=i}^{u-1} p_{qh} \right\}$$

where the case $i = u$ corresponds to $T^{min}_{u,u-1} = \min_{h \in U_k} \left\{ s_{u[k]h} \right\}$. A lower bound on the starting time of job $[k+1]$ on machine u is then given by

$$\max_{1 \le i \le u} \left\{ C_{i[k]} + T^{min}_{i,u-1} \right\}$$

Note that once the last job $[n]$ has finished on machine v, the remaining time until termination (assuming no idle time) is $\sum_{i=v+1}^{m} p_{i[n]}$. This yields the following lower bound for the elapsed time since the finish of job $[n]$ on machine v until the finish of job $[n]$ on machine m:

$$\min_{h \in U_k} \left\{ \sum_{i=v+1}^{m} p_{ih} \right\}$$

We can thus establish the following generalized lower bound $g_{uv}(\bar{S}_k)$ on $C_{\max}$

$$g_{uv}(\bar{S}_k) \quad = \quad \max_{1 \le i \le u} \left\{ C_{i[k]} + T^{min}_{i,u-1} \right\} + Z^*_{uv}(\bar{S}_k) + S^*_{uv}(\bar{S}_k) - \sum_{i=u+1}^{v-1} p_i(\bar{S}_k) + \min_{h \in U_k} \left\{ \sum_{i=v+1}^{m} p_{ih} \right\}$$

for any $1 \le u < v \le m$. Note that the optimal sequence of the jobs in the embedded 2-machine flowshop (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 various machine pairs (u, v). 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

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

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: In the previous section we developed a family of lower bounds g_{uv} for $1 \le u < v \le m$, based on a pair (u, v) of bottleneck machines. 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, $1 \leq u \leq m$.

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. For an arbitrary sequence of jobs in U_k, $\bar{S}_k = ([k+1], [k+2], \ldots, [n])$, let $T_u(\bar{S}_k)$ be the elapsed time from the starting time of job $[k+1]$ until the finish time of job $[n]$ on machine u. Then $T_u(\bar{S}_k)$ is given by

$$\begin{aligned} T_u(\bar{S}_k) &= p_{u[k+1]} + \sum_{h=k+2}^{n} (s_{u[h-1][h]} + p_{u[h]}) \\ &= p_u(\bar{S}_k) + \sum_{h=k+2}^{n} s_{u[h-1][h]} \end{aligned}$$

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

$$g_u(\bar{S}_k) = \max_{1 \leq i \leq u} \left\{ C_{i[k]} + T_{i,u-1}^{min} \right\} + S_u^*(\bar{S}_k) + \min_{h \in U_k} \left\{ \sum_{i=u+1}^{m} p_{ih} \right\} \tag{3}$$

for $1 \leq u \leq m$ is a valid lower bound on $C_{\max}$, where the first and last terms on the right-hand side are a lower bound on the starting time of job $[k+1]$ on machine u, and a lower bound on the elapsed time between the finish of job $[n]$ on machine u and the finish of job $[n]$ on machine m, respectively, as developed in the previous section.

The fact that the setup time between jobs [k] and $[k+1]$, $s_{u[k][k+1]}$, is not considered in the computation of $T_u(\bar{S}_k)$ allows us to use the first term on the right-hand side 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 right-hand side of (3). This alternate bound is expressed as

$$g'_u(\bar{S}_k) = L'_u(\bar{S}_k) + \min_{h \in U_k} \left\{ \sum_{i=u+1}^{m} p_{ih} \right\}$$

where L'_u is valid lower bound on $\sum_{h=k+1}^{n} s_{u[h-1][h]}$.

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 [1]) 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 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.*

Proof: Let Q be a schedule and $q_i(Q)$ be the elapsed time between the starting of the first job in Q on machine i and the end of operations. Then for a partial schedule S_k, let Q be any schedule formed by the jobs in U_k (set of unscheduled jobs). The makespan of S_kQ can be computed as

$$C_{\max}(S_kQ) \quad = \quad \max_{i\in I}\Big\{C_i(S_k) + s_{i,l(S_k),h} + q_i(Q)\Big\}$$

where h is the job index of the first job in Q. Let P_j be a subproblem such that $J(S_j) = J(S_k)$ (its corresponding partial schedule S_j has the same job indices as those of schedule S_k), $l(S_j) = l(S_k)$ (have the same job scheduled last), and $C_i(S_j) < C_i(S_k)$ for $i \in I$. Since the set of unscheduled jobs is the same for both subproblems, S_jQ is also a valid completion for P_j, and

$$s_{i,l(S_j),h} + q_i(Q) \quad = \quad s_{i,l(S_k),h} + q_i(Q) \qquad i \in I$$

Therefore

$$C_i(S_j) \leq C_i(S_k) \quad i \in I \quad \Rightarrow \quad C_i(S_j) + s_{i,l(S_j),h} + q_i(Q) \leq C_i(S_k) + s_{i,l(S_k),h} + q_i(Q) \quad i \in I$$

$$\Rightarrow \quad \max_{i \in I} \left\{ C_i(S_j) + s_{i,l(S_j),h} + q_i(Q) \right\} \leq \max_{i \in I} \left\{ C_i(S_k) + s_{i,l(S_k),h} + q_i(Q) \right\}$$

$$\Rightarrow \quad C_{\max}(S_j Q) \leq C_{\max}(S_k Q)$$

which shows that P_j dominates P_k. ■

A second dominance rule arises for the special case where there is no idle time between a subsequence of any three particular jobs in a schedule. This is presented in Lemma 1 in Appendix A. Two other special cases, the first related to reversing the job sequence and the second to the independence of processing times and machines, are also discussed in the Appendix.

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 a hybrid heuristic (described in [16]) 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 UPPER_BOUND_LOG nodes are generated, where UPPER_BOUND_LOG is a user-specified parameter. In our experiments, we set this parameter to 50. 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 [13]. 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 flowshop perspective, and S_{jk} is a term that penalizes large setup times. This cost measure was introduced in [16] where it was used to develop a TSP-based hybrid heuristic for the SDST flowshop 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 [4] 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 with the setup times being 20-40% of the processing times: $p_{ij} \in [20, 100]$ 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 settings $m = 4, 6$. 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

	$m = 2$		$m = 4$		$m = 6$	
	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*

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 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

	$m = 2$		$m = 4$		$m = 6$	
	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*

We now evaluate the effectiveness of the dominance rule. Table 3 shows the average statistics over 10 instances for machine sizes $m = 2, 4, 6$. 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.

5.3 Experiment 3: Partial Enumeration

	$\rho = 0$			$\rho = 0.5$			$\rho = 0.8$		
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*

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 ($\rho \in \{0.5, 0.8\}$) do not correspond to a true optimality gap, but to the best lower bound without 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.

5.4 Experiment 4: BABAS Overall Performance

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

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*

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 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*

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.

6 Summary

We have presented and evaluated a branch-and-bound scheme for the SDST flowshop scheduling problem. Our implementation includes both lower and upper bounding procedures, and a dominance elimination criterion. The empirical results indicate the positive impact 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 relatively smaller computational effort.

7 Acknowledgments

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

Appendix A

This appendix contains three lemmas which address special cases of the SDST flowshop. The first presents a dominance rule, the second discusses the reversibility of the schedule, and the third considers specific parameter relationships. To simplify the presentation, the bracket notation for a given schedule will be dropped and we will denote a schedule S by $(1,\ldots,n)$ rather than $([1],\ldots,[n])$.

Lemma 1 *Let $S = (1,2,\ldots,n)$ be a feasible schedule of $F|s_{ijk},\ prmu|C_{\max}$. Let e_{ij} be the earliest completion time of job j on machine i*

$$e_{ij} \quad = \quad \max\{e_{i-1,j},\ e_{i,j-1} + s_{i,j-1,j}\} + p_{ij}$$

for $i = 1,2,\ldots,m,\ j = 1,2,\ldots,n,$ and $e_{i0} = e_{0j} = 0$. Let q_{ij} be the minimum remaining time from the start of job j on machine i to the end of operations on the last machine

$$q_{ij} = \max\{q_{i+1,j},\ q_{i,j+1} + s_{i,j,j+1}\} + p_{ij}$$

for $i = m, m-1,\ldots,1,\ j = n, n-1,\ldots,1,$ and $q_{i,n+1} = q_{m+1,j} = 0$. Let j and $j+1$ be any two adjacent jobs in S ($j = 1,2,...,n-1$) and let $S' = (1,\ldots,j-1,j+1,j,j+2,\ldots,n)$ be the schedule where jobs j and $j+1$ are exchanged (with completion time e'_{ij} and remaining time q'_{ij}).

If all of the following conditions hold for each $i = 1,2,\ldots,m$

(a) $e_{ij} = e_{i,j-1} + s_{i,j-1,j} + p_{ij}$ *(there is no idle time between jobs* $j-1$ *and* j *in* S*)*

(b) $q_{i,j+1} = q_{i,j+2} + s_{i,j+1,j+2} + p_{i,j+1}$ *(there is no idle time between jobs* $j+1$ *and* $j+2$ *in* S*)*

(c) $e'_{i,j+1} = e'_{i,j-1} + s_{i,j-1,j+1} + p_{i,j+1}$ *(there is no idle time between jobs* $j-1$ *and* $j+1$ *in* S'*)*

(d) $q'_{i,j} = q'_{i,j+2} + s_{i,j,j+2} + p_{i,j}$ *(there is no idle time between jobs* j *and* $j+2$ *in* S'*)*

(e) $s_{i,j-1,j} + s_{i,j,j+1} + s_{i,j+1,j+2} > s_{i,j-1,j+1} + s_{i,j+1,j} + s_{i,j,j+2}$

then S' *has a lower makespan than* S*,*

$$C_{\max}(S') < C_{\max}(S).$$

Proof: First notice that both S and S' are identical sequences except for jobs j and $j+1$. This implies that $e_{ik} = e'_{ik}$ for all $k = 1, 2, \ldots, j-1$ and $q_{ik} = q'_{ik}$ for all $k = j+2, j+3, \ldots, n$. Thus, from (e) we obtain

$$e_{i,j-1} + s_{i,j-1,j} + p_{ij} + q_{i,j+2} + s_{i,j+1,j+2} + p_{i,j+1} > e'_{i,j-1} + s_{i,j-1,j+1} + p_{i,j+1} + q'_{i,j+2} + s_{i,j,j+2} + p_{ij}$$

for all i. Conditions (a)-(d) yield

$$e_{ij} + s_{i,j,j+1} + q_{i,j+1} > e'_{i,j+1} + s_{i,j+1,j} + q'_{ij} \quad \text{for all } i$$

In particular, this is valid for the maximum over i

$$\max_i \{e_{ij} + s_{i,j,j+1} + q_{i,j+1}\} > \max_i \left\{e'_{i,j+1} + s_{i,j+1,j} + q'_{ij}\right\}$$

But these expressions correspond to the makespan values of S and S', respectively. That is,

$$C_{\max}(S) > C_{\max}(S').$$

■

An appropriate data structure should keep track of both e_{ij} and q_{ij} for all i and j. This would make it possible to check conditions (a)-(d) in $O(m)$ time.

As seen in Section 4.2, Proposition 1, $T_{uv}(\bar{S}_k)$ (the elapsed time between the first job in $\bar{S}_k$ on machine u and the last job in $\bar{S}_k$ on machine v) can be computed by finding the critical path on graph G_{uv} (Figure 3). Note that $T_{1m}(S)$ is an equivalent form to express the makespan of schedule S, which implies, by Proposition 1, that its makespan is given by the critical path from node $(1,0)$ to node (m,n) in graph G_{1m}.

An interesting property can be obtained when comparing two instances of the SDST flowshop with no initial setup times. Let FS be an instance of $F|s_{ijk},\ prmu|C_{\max}$ with processing

times p_{ij} and setup times s_{ijk}. Let us assume that $s_{i0k} = 0$ for all $i \in I$, and $k \in J$. Let FS$'$ be another instance of the SDST flowshop with processing and setup times given by

$$\begin{aligned} p'_{ij} &= p_{m+1-i,j}, \quad \text{and} \\ s'_{ijk} &= s_{m+1-i,k,j}, \end{aligned}$$

respectively. This basically implies that the first machine in the FS$'$ is identical to the last machine in FS; the second machine in FS$'$ is identical to machine $m-1$ in FS, and so on. The following lemma applies to these two flowshops.

Lemma 2 *Let $S = (1, \ldots, n)$ be a sequence of jobs in FS with corresponding makespan $C_{\max}(S)$. If the jobs in FS$'$ follow the sequence $S' = (n, n-1, \ldots, 1)$ (with makespan $C'_{\max}(S')$), then*

$$C_{\max}(S) = C'_{\max}(S').$$

Proof: Let $S = (1, \ldots, n)$ be a feasible sequence in FS. Then its makespan $C_{\max}(S)$ is given by $T_{1m}(S)$, the length of the critical path in G_{1m}. Let G'_{1m} be the graph associated to FS$'$ under sequence $S' = (n, \ldots, 1)$. By definition of FS$'$, G'_{1m} is obtained from G_{1m} by reversing the sense of all the arcs in G_{1m}. Since the length of the critical path from does not change, it follows that $T_{1m}(S) = T'_{1m}(S')$, where $T'_{1m}(S')$ is the length of the critical path in G'_{1m}, and the proof is complete. ■

Lemma 2 states the following reversibility result: the makespan does not change if the jobs go through the flowshop in the opposite direction in the reverse order.

Another special case of $F|s_{ijk},\ prmu|C_{\max}$ which is of interest is the so-called proportionate flowshop. In this flowshop the processing times of job j on each machine are equal to p_j, that is, $p_{ij} = p_j$, $i = 1, \ldots, m$. Minimizing the makespan in a proportionate permutation flowshop is denoted by $F|p_{ij} = p_j,\ prmu|C_{\max}$. This problem has a very special property when all setup times are equal to a constant $s_{ijk} = s$.

Lemma 3 *For $F|p_{ij} = p_j,\ s_{ijk} = s,\ prmu|C_{\max}$, the makespan is given by*

$$C_{\max} = \sum_{j=1}^{n} p_j + ns + (m-1) \max_j \{p_j\}$$

and is independent of the schedule.

Proof: From Figure 3 we can see that for any sequence of jobs $S = (1, 2, \ldots, n)$ the critical path starts at node $(1, 0)$, stays on machine 1 until it reaches node $(1, k)$, where $k = \arg\max_j \{p_j\}$, stays on job k until it reaches node (m, k), and ends by reaching node (m, n). ■

Similar results on reversibility and proportionate flowshops for $F|prmu|C_{\max}$ are discussed in [14].

References

[1] 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.

[2] 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.

[3] 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.

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

[5] S. K. Gupta. n jobs and m machines job-shop problems with sequence-dependent set-up times. *International Journal of Production Research*, 20(5):643–656, 1982.

[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] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S. S. Graves, A. H. G. Rinnooy Kan, and P. Zipkin, editors, *Handbook in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory*, pages 445–522. North-Holland, New York, 1993.

[12] 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.

[13] T. E. Morton and D. W. Pentico. *Heuristic Scheduling Systems*. John Wiley & Sons, New York, 1993.

[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 Flow Shop Scheduling Problem with Setup Times*. PhD thesis, University of Texas at Austin, Austin, TX 78712–1063, 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 at Austin, 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. Technical Report ORP97–01, Graduate Program in Operations Research, University of Texas at Austin, Austin, TX 78712–1063, May 1997.

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

[20] 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.

[21] 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.

[22] 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.