An Enhanced TSP-Based Heuristic for Makespan Minimization in a Flow Shop with Setup Times

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

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

December 1997
Revised August 1998
Accepted September 1998

Abstract

This paper presents an enhanced heuristic for minimizing the makespan of the flow shop scheduling problem with sequence-dependent setup times. The procedure transforms an instance of the problem into an instance of the traveling salesman problem by introducing a cost function that penalizes for both large setup times and bad fitness of schedule. This hybrid cost function is an improvement over earlier approaches that penalized for setup times only, ignoring the flow shop aspect of the problem. To establish good parameter values, each component of the heuristic was evaluated computationally over a wide range of problem instances. In the testing stage, an experimental comparison with a greedy randomized adaptive search procedure revealed the conditions and data attributes where the proposed procedure works best.

Keywords: flow shop scheduling, sequence-dependent setup times, heuristics, traveling salesman problem

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 $C_{\max}$ (makespan) of all jobs. 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 setup time 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 type 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 [10], we proposed a greedy randomized adaptive search procedure (`GRASP()`) for this problem and compared it to Simons' heuristics [11] `SETUP()` and `TOTAL()`. `GRASP()` is an insertion-based heuristic. `SETUP()` and `TOTAL()` are fundamentally based on transforming the SDST instance into a related instance of the asymmetric traveling salesman problem (ATSP). Our empirical evaluation indicated that `GRASP()` found better schedules when the setup to processing time ratio was small (about 0.1). However, when the setup times were allowed larger variations (average ratio 1.0), `SETUP()` proved superior. We also observed that it produced better results when the number of machines was small. This stemmed from the fact that the larger the setups and the smaller the number of machines, the more the problem resembles a TSP so a TSP-based heuristic such as `SETUP()` should do better than an insertion-based heuristic such as `GRASP()`.

Despite the attractiveness of the former, some shortcomings exist in Simons' work. The first is that the cost function (for scheduling two jobs together) includes a penalty term for setup times only, ignoring the flow shop aspect of the problem. In particular, there might be pairs of jobs that cause significant blocking and/or machine idle time when they are scheduled together even though their setup times are small. In addition, Simons made no effort to improve the solution through local search. As such, the objective of this paper is to present and evaluate an improved TSP-based heuristic (`HYBRID()`) for the SDST flow shop. The first aspect of this work involved the development of the hybrid cost function to capture the dual nature of the problem. This was followed by the implementation of local search procedures for obtaining local minima. Also included in the algorithm is a parameter variation step that allows us to diversify the search and explore more of the feasible region.

The proposed heuristic `HYBRID()` clearly dominates `SETUP()` since `SETUP()` is in fact a special case of the former. In the computational testing, `HYBRID()` is also seen, in general, to outperform `GRASP()` over a wide variety of randomly generated instances, especially on those where the similarities with the TSP are greater.

The rest of the paper is organized as follows. The most relevant work on flow shop scheduling 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 heuristic, 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 a comprehensive review of the diversity of problems on machine scheduling research involving setup times, see Allahverdi et al. [1]. Here we review the research most relevant to our work.

2.1 Heuristics

In addressing the makespan minimization of the SDST flow shop, Simons [11] described four heuristics and compared them with three benchmarks that represent generally practiced approaches to scheduling in this environment. Experimental results for problems with up to 15 machines and 15 jobs were presented. His findings indicated that two of the proposed heuristics – `SETUP()` and `TOTAL()` – produced substantially better results than the other methods tested.

In [10], we proposed a GRASP for this problem and compared it to Simons' heuristics `SETUP()` and `TOTAL()`. In this paper, we provide an enhanced version of Simons' heuristics by correcting some of their shortcomings and by adding a local search phase. A full description is given in Section 4.

2.2 Exact Optimization

Other approaches to the m-machine problem have focused on exact optimization schemes based on branch and bound [8, 12, 13] and branch and cut [7, 9]. All other work has been restricted to the 1- and 2-machine case. In general, the largest instances addressed successfully are reported in [8] where several 6-machine, 20-job problems were solved to optimality within 30 minutes using branch and bound. This approach was seen to outperform branch and cut on problems with 2 to 6 machines and up to 20 jobs.

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., $s_{ijk} \neq s_{ikj}$. 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 [6] 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	30	20	10	5
2	15	25	20	25

s_{1jk}	1	2	3	4
0	2	4	5	3
1	–	10	7	3
2	6	–	12	8
3	7	11	–	6
4	5	7	2	–

s_{2jk}	1	2	3	4
0	2	6	1	3
1	–	2	7	11
2	9	–	3	7
3	8	5	–	10
4	8	4	10	–

A schedule $S = (4, 2, 1, 3)$ is shown in Figure 1. The corresponding makespan is 109, which is optimal. □

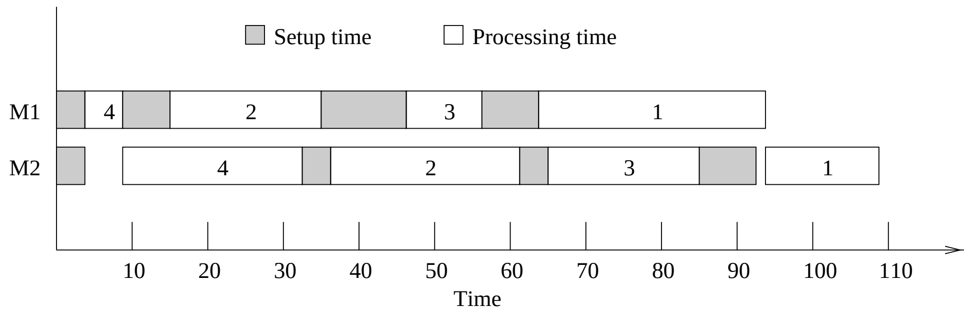

Figure 1: *Example of a* 2×4 *SDST flowshop*

3.1 Notation

In the reminder of the paper, when we refer to the SDST 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.

4 Description of Heuristic

Simons' [11] main idea was first to transform an instance of the SDST flow shop into an instance of the ATSP by computing an appropriate cost matrix, and then to solve the latter by applying a well-known heuristic for the ATSP. In the first of two phases of his heuristics, an instance of the ATSP is built as follows. Every job is identified with a "city." Procedure `TOTAL()` computes the entries in the distance (cost) matrix as the sum of both the processing and setup times over all the machines. Procedure `SETUP()` considers the sum of setup times only. In the second phase, a feasible tour is obtained by invoking a heuristic for the ATSP. This heuristic uses the well-known Vogel's approximation method (VAM) for obtaining good initial solutions to transportation problems with a slight modification to eliminate the possibility of subtours. The ATSP solution maps back into a feasible schedule for the SDST flow shop.

Although this approach seems suitable, given the strong similarities between the SDST flow shop and the ATSP, Simons' work was limited in two respects, as mentioned above. First, the cost function ignores completely the flow shop aspect of the problem, and second, no attempt was made to achieve local optimality.

4.1 Construction Phase

Our `HYBRID()` heuristic is based on Simons' [11] idea of exploiting the embedded ATSP structure to derive good schedules. We attempt to improve on his approach by introducing a cost function that balances setup times and schedule fitness for each pair of jobs. Let C_{jk} be the cost of scheduling job j right before job k. We express this measure as

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

where $\theta \in [0,1]$, and R_{jk} and S_{jk} are the costs of scheduling jobs j and k together, from the flow shop and the setup time perspective, respectively. The setup cost component is simply

$$S_{jk} \quad = \quad \sum_{i \in I} s_{ijk}$$

such that when $\theta = 0$, the cost measure is reduced to Simons' measure for his `SETUP()` heuristic.

We now develop the cost R_{jk} following an idea similar to the one used by Stinson and Smith [14] for $F|prmu|C_{\max}$. Let t_{ij} denote the completion of job j on machine i. Assume that job k immediately succeeds job j. The completion time of job k on any machine can then be recursively determined as follows:

$$t_{ik} \quad = \quad \max\{t_{ij} + s_{ijk}, t_{i-1,k}\} + p_{ik}$$

The relationship between $t_{ij} + s_{ijk}$ and $t_{i-1,k}$ plays a key role here. If $t_{ij} + s_{ijk} > t_{i-1,k}$, then job k will arrive at machine i before job j has been released from machine i; hence job k will be *blocked* in the queue at machine i for $t_{ij} + s_{ijk} - t_{i-1,k}$ time units. On the other hand, if $t_{ij} + s_{ijk} < t_{i-1,k}$, then machine i will be idle for $t_{i-1,k} - (t_{ij} + s_{ijk})$ time units while waiting for job k to arrive. The ideal situation occurs when $t_{ij} + s_{ijk} = t_{i-1,k}$, in which case neither job k is blocked nor does machine i experience any idleness.

Now, let us take this rationale a step further by considering the set of circumstances which would have to take place if $t_{ij} + s_{ijk}$ were to equal $t_{i-1,k}$ for the entire period where both j and k are scheduled together. This will occur only when $p_{ij} + s_{ijk} = s_{i-1,jk} + p_{i-1,k}$ for every machine $i = 2, \ldots, m$. Although we would seldom, if ever, expect such an ideal set of circumstances to arise in practice, the closer the values of $p_{ij} + s_{ijk}$ and $s_{i-1,jk} + p_{i-1,k}$ are matched for all machines, the tighter jobs j and k will tend to fit together within the schedule. With this in mind, we now define a residual factor, r_{ijk}, as

$$r_{ijk} \quad = \quad p_{ij} + s_{ijk} - (s_{i-1,jk} + p_{i-1,k}) \qquad i \in I \setminus \{1\},\ j,k \in J_0$$

where $p_{i0} = 0$ for $i \in I$. For any pair (j,k), $j \neq k$, we may compute $m-1$ such residuals which are then combined in several different ways to yield the overall cost, R_{jk}. The choices considered are listed in Table 1.

Function	Description
$R^1_{jk} = \sum_{i=2}^{m} \lvert r_{ijk} \rvert$	Sum of the absolute residuals
$R^2_{jk} = \sum_{i=2}^{m} [r_{ijk}]^+$	Sum of positive residuals only, where $[r_{ijk}]^+ = r_{ijk}$ if $r_{ijk} > 0$; 0 otherwise
$R^3_{jk} = \sum_{i=2}^{m} [r_{ijk}]^-$	Sum of negative residuals only, where $[r_{ijk}]^- = -r_{ijk}$ if $r_{ijk} < 0$; 0 otherwise
$R^4_{jk} = \sum_{i=2}^{m} 2[r_{ijk}]^+ + [r_{ijk}]^-$	Sum of absolute residuals with positive residuals weighted double
$R^5_{jk} = \sum_{i=2}^{m} [r_{ijk}]^+ + 2[r_{ijk}]^-$	Sum of absolute residuals with negative residuals weighted double

Table 1: Residual functions investigated

With R^1 each residual, regardless of its direction of error, is equally weighted. Rules R^2 and R^4 penalize more for positive residuals (blocking) whereas R^3 and R^5 penalize more for negative residuals (idle time). It is important to note that the sign of each r_{ijk} value is significant. A positive r_{ijk} implies that a degree of blocking for job k at machine i is likely to occur. On the other hand, a negative r_{ijk} implies idleness at machine i. This motivates the choices of rules R^2 through R^5. In fact, preliminary computational experience showed that all of these rules were very helpful; however, for some instances it was observed that rules R^2 and R^4 (which assess a greater penalty for positive residuals) were uniformly dominated by the others (see Section 5.1). This indicates that machine idleness is of greater concern than job blocking when constructing a schedule. One explanation for this is that a negative residual at some machine i has a carryover effect on the remaining downstream machines.

In general, for a given value of $\theta \in [0, 1]$ and a given residual cost rule, there is an associated cost matrix C. To devise a heuristic based on these parameters, let $\Theta = \{\theta_1, \ldots, \theta_p\}$ be a (finite) discretization of $[0, 1]$, where p is the size of the discretization, and let $R = \{R^1, R^2, R^3, R^4, R^5\}$ be the set of cost functions (as defined in Table 1). The construction phase of procedure `HYBRID()` is shown in Figure 2. A local search phase is then applied to this schedule in an attempt to find a local optimum with respect to neighborhood arguments. This is discussed in Section 4.2.

Computational complexity: The computation of the cost matrix performed in Step 3 takes $O(mn^2)$ time. The application of Vogel's modified method to a $(n+1)$-city problem is $O(n^2)$ so the overall procedure has worst-case complexity of $O(|R||\Theta|mn^2)$. When $|R| = O(1)$ this brings the complexity down to $O(|\Theta|mn^2)$. In this regard, preliminary testing has shown that any discretization with $|\Theta| > 10$ provides no better solutions than a discretization with $|\Theta| = 10$.

```
Procedure HYBRID_phase1()
Input: An instance of the SDST flowshop, a discretization
$\Theta$ of the weight range, and a set $R$ of residual cost functions.
Output: A feasible schedule $S$.

0:  Initialize best schedule $S_{\text{best}} = \emptyset$, $C_{\max}(S_{\text{best}}) = \infty$
1:  for each $R^l \in R$ do
2:      for each $\theta \in \Theta$ do
3:          Compute $(n+1) \times (n+1)$ cost matrix as
            $C_{jk} = \theta R^i_{jk} + (1-\theta) S_{jk}$
4:          Apply modified VAM to $(C_{jk})$ to obtain a tour $S$
5:          If $C_{\max}(S) < C_{\max}(S_{\text{best}})$ then $S_{\text{best}} \leftarrow S$
6:  Output $S_{\text{best}}$
7:  Stop
```

Figure 2: Pseudocode of HYBRID() phase 1

Hence, we take $\Theta = \{0.0, 0.1, \ldots, 1.0\}$ giving a time complexity of $O(mn^2)$.

4.2 Local Search

Neighborhoods can be defined in a number of different ways, each having different computational implications. Consider, for instance, a 2-opt neighborhood definition that consists of exchanging two edges in a given tour or sequence of jobs. For this neighborhood, a move in a TSP takes $O(1)$ time to evaluate whereas a move in the SDST flow shop takes $O(mn^2)$. One of the most common neighborhoods for scheduling problems is the 2-job exchange which has been used by Widmer and Hertz [16] and by Taillard [15] for $F|prmu|C_{\max}$. We considered the 2-job exchange as well. In addition, in [7] we generalized the 1-job reinsertion neighborhood proposed by Taillard [15] for $F|prmu|C_{\max}$ to develop an L-job string reinsertion procedure (that is, remove a string of L jobs and reinsert it in a different place in the schedule). This was motivated by the presence of the sequence-dependent setup times, which suggest that subsets (or strings) of consecutive jobs might fit together in a given schedule. We tried both procedures for our problem and found that the string reinsertion uniformly outperformed the 2-job exchange, just as Taillard found the 1-job reinsertion performed better than the 2-job exchange for the regular flow shop.

In general the neighborhood definition is different for each value of the string size L; that is, a local optima with respect to $L = 1$, for instance, may not be local optima with respect to $L = 2$. Thus in practice, one can apply or combine several of these neighborhoods for different values of L, depending on the particular trade-off value between quality of solution desired and time available. For instance, `HYBRID()` is a deterministic heuristic that runs very quickly. This makes a local search effort more affordable.

5 Experimental Evaluation

All procedures 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()`.

	p_{ij}	s_{ijk}
Class A	[10,100]	[1,10]
Class D	[20,100]	[20,40]
Class C	[50,100]	[1,50]

Table 2: Data class attributes

To conduct our experiments we used randomly generated data drawn from classes A, D, and C, where both processing and setup times are generated according to a uniform distribution in the intervals shown in Table 2. Class D is most representative of real world instances, having a setup/processing time ratio between 20% and 40%. Classes A and C, account for a smaller (1–10%) and a larger (1–50%) ratio variation, respectively, and are intended to assess algorithmic performance in best- and worst-case scenarios.

For a given combination of $(m \times n) \in \{2, 4, 6, 8, 10\} \times \{20, 50, 100\}$ and for each data class we generated 20 random instances. In total then, each of the experiments described below was tested on 5 machine settings $\times$ 3 job settings $\times$ 3 data classes $\times$ 20 replications $=$ 900 instances.

5.1 Experiment 1: Evaluation of Cost Function

The purpose of this preliminary experiment was to evaluate the impact of the cost function

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

as the parameter $\theta \in [0, 1]$ and the residual cost component R_{jk} varied. Recall from Section 4 that five residual functions were proposed. Therefore, by taking all combinations of a particular residual cost function and an element of a partition for θ, we formed "different" algorithms.

The first issue addressed involved the determination of acceptable ranges for the weight parameter θ. For this purpose we ran the algorithm using residual cost function R^1 for $\theta \in \{0.0, 0.2, \dots, 1.0\}$. For each run, we tallied the number of times a given choice of θ found the best (or tied for best) solution. Tables 3, 4, and 5 display the results for data sets A, D, and C, respectively.

	Values of θ							
$m \times n$	0.0	0.2	0.4	0.6	0.8	1.0	F	Best range for θ
2×20	3	9	3	3	1	2	$(*)$ 5.92	[0.0,0.6]
50	3	10	7	0	0	0	$(*)$ 30.08	[0.0,0.4]
100	11	6	3	0	0	0	$(*)$ 57.97	[0.0,0.4]
6×20	2	6	2	7	6	7	$(*)$ 3.27	[0.2,1.0]
50	0	6	2	2	6	4	$(*)$ 9.66	[0.2,1.0]
100	0	7	5	4	1	3	$(*)$ 7.58	[0.2,1.0]
10×20	2	8	7	7	3	4	$(*)$ 6.47	[0.2,1.0]
50	1	3	10	3	2	4	$(*)$ 7.41	[0.2,1.0]
100	0	3	5	4	4	4	$(*)$ 5.91	[0.2,1.0]
$(*)$ F test significant at $\alpha = 99\%$								

Table 3: Effect of parameter θ on class A instances

	Values of θ							
$m \times n$	0.0	0.2	0.4	0.6	0.8	1.0	F	Best range for θ
2×20	7	5	5	3	1	0	$(*)$ 17.78	[0.0,0.6]
50	13	3	2	2	0	0	$(*)$ 55.30	[0.0,0.4]
100	14	6	0	0	0	0	$(*)$ 263.27	[0.0,0.2]
6×20	2	5	3	6	2	5	2.35	[0.0,1.0]
50	4	1	3	5	3	4	0.80	[0.0,1.0]
100	7	8	4	0	1	0	$(*)$ 7.78	[0.0,0.4]
10×20	3	4	5	4	4	5	1.73	[0.0,1.0]
50	2	5	4	2	3	4	2.29	[0.0,1.0]
100	0	6	7	4	3	0	$(*)$ 3.28	[0.2,0.8]
$(*)$ F test significant at $\alpha = 99\%$								

Table 4: Effect of parameter θ on class D instances

In these tables, each row shows, for a particular $m \times n$ combination, the number of times a given choice of θ found the best, or tied for best, solution. In addition, the Friedman test (non-parametric statistical test equivalent to classical ANOVA [2]) was applied to determine whether there is a significant difference among the values of θ used. The observed F value is shown in the second to last column. If the test was significant at $\alpha = 99\%$ (greater than $F_{0.99,5,95} = 3.241$), it is indicated with an asterisk $(*)$. The last column shows the best range for θ obtained from the experiments

The following observations were made:

- For data set A, the range [0.2,1.0] provided the best results, except for the 2-machine instances. This finding can be explained by noting that in data class A, the setup time variation is small, which lessens the importance of the setup time contribution S_{jk} in the cost function (1). However, for the 2-machine instances, ranges starting from 0.0 should be considered because strong similarities with the ATSP still exist despite the small setup

	Values of θ							
$m \times n$	0.0	0.2	0.4	0.6	0.8	1.0	F	Best range for θ
2×20	13	5	1	1	0	0	(∗) 74.82	[0.0,0.2]
50	15	6	0	0	0	0	(∗) 229.48	[0.0,0.2]
100	17	3	0	0	0	0	(∗) 315.17	[0.0,0.2]
6×20	1	9	10	4	0	0	(∗) 12.99	[0.2,0.6]
50	9	8	3	0	0	0	(∗) 53.99	[0.0,0.4]
100	13	7	0	0	0	0	(∗) 241.78	[0.0,0.2]
10×20	4	4	6	5	1	0	(∗) 3.80	[0.0,0.6]
50	6	6	6	2	0	0	(∗) 29.64	[0.0,0.4]
100	9	10	1	0	0	0	(∗) 103.69	[0.0,0.2]
(∗) F test significant at $\alpha = 99\%$								

Table 5: Effect of parameter θ on class C instances

times.

- For data set D, practically the complete [0,1] range for θ yielded good results, except again for the 2-machine instances (same explanation as before) and the large 100-job instances. Looking at the 100-job instances, the acceptable ranges for θ shift towards 1.0 as the number of machines becomes large. This behavior is expected since increasing the number of machines makes the setup time contribution less representative and the residual cost component more important.

- For data set C, ranges less than 0.4 or 0.6 worked best for all instances. This is an intuitive result since this is the data set with the largest setup time variations. Therefore, one would expect the setup time component of the cost function to play a more decisive role.

- A final remark about these results is that by allowing parameter θ to vary in the [0,1] range, we were able to obtain a wide variety of schedules. In almost all cases, these schedules were better than those obtained when θ was fixed at zero. Recall that the special case of $\theta = 0$ represents Simons' `SETUP()` heuristic.

The second component of this experiment involved the evaluation of algorithmic performance for different values of the residual cost function R^l, $l = 1, \ldots, 5$. To make the assessment, we re-ran the algorithm fixing θ at 0.2, 0.6, and 1.0. Note that this was only done for machine and job size combinations that were found acceptable for a specific θ value in the first round of the experiments. For example, because a value $\theta = 1$ was found unacceptable for all 2-machine class D instances in the first round of experiments (Table 4), we did not run the 2-machine instances for $\theta = 1$ for data set D. Tables 6 and 7 show the results for data sets D and C, respectively. Results for data set A (not included) were similar to those for D.

		Residual cost functions					
θ	$m \times n$	R^1	R^2	R^3	R^4	R^5	F
0.2	2×20	5	7	6	1	3	2.56
	50	5	$\otimes$2	6	3	4	$(*)$ 5.33
	100	$(\star)$ 12	3	4	1	0	$(*)$ 9.56
	6×20	2	3	4	5	6	1.16
	50	3	5	7	2	3	0.60
	100	6	7	5	1	1	3.19
	10×20	3	1	4	3	9	1.12
	50	5	3	9	0	3	3.33
	100	4	7	5	3	1	1.84
0.6	2×20	5	4	7	2	4	1.89
	6×20	6	5	4	1	4	1.25
	50	6	3	4	3	4	0.86
	10×20	4	3	4	4	5	2.93
	50	3	7	4	5	2	0.72
	100	5	5	4	2	4	0.38
1.0	6×20	4	$\otimes$1	5	4	7	$(*)$ 3.67
	50	6	4	1	3	6	0.52
	10×20	5	3	1	5	6	1.96
	50	7	1	3	6	3	0.57
	100	6	1	4	3	6	1.56
$(*)$ F test significant at $\alpha = 99\%$							

Table 6: Effect of residual cost functions on class D instances

In essence, the information displayed is the same as before. The difference is that this time, the Friedman test is significant at $\alpha = 99\%$ if $F > F_{0.99,4,76} = 3.604$. For those cases where this test was significant (which supports the alternate hypothesis that not all choices of R^l perform equally well) a Wilcoxon test (non-parametric pairwise test [2]) was done between all possible pairs to determine which of the choices were dominant (marked with a star $(\star)$) or dominated (marked with an $\otimes$) in the statistical sense. The main observation is that in practically all cases tested, there was no statistical evidence that any specific rule performed better or worse than the others. Consequently, it would be advantageous to use all of these functions in the overall procedure since each accounts for approximately 20% of the best schedules found. Results for data set A were similar.

For data set C, the results were a bit different. As can be seen from Table 7, most of the tests were significant. R^1 and R^3 were the best choices for half the instances and R^1 was the best choice for the other half. What this result says is that when setup time variations are large (data set C), penalizing equally for positive and negative deviations (rule R^1) suffices since now the setup time component S_{jk} in eq. (1) plays a more significant role. The cases were R^3 was also found to be a good choice means that having machine idle time (negative residuals) causes

θ	$m \times n$	R^1	R^2	R^3	R^4	R^5	F
		Residual cost functions					
0.2	2×20	$(\star)$ 9	2	$(\star)$ 5	1	4	$(*)$ 4.24
	50	$(\star)$ 16	1	2	0	1	$(*)$ 12.44
	100	$(\star)$ 20	0	0	0	0	$(*)$ 21.43
	6×20	6	4	7	0	3	2.79
	50	$(\star)$ 10	0	$(\star)$ 8	2	0	$(*)$ 4.91
	100	$(\star)$ 13	4	2	1	0	$(*)$ 13.79
	10×20	4	6	7	2	2	1.75
	50	7	3	6	$\otimes$1	3	$(*)$ 4.71
	100	$(\star)$ 9	2	$(\star)$ 9	0	0	$(*)$ 13.53

$(*)$ F test significant at $\alpha = 99\%$

Table 7: Effect of residual cost functions on class C instances

a larger disruption in the schedule than having a blocked machine (positive residuals). This can be explained by noting that when idle time is incurred at a machine, there is a carryover effect on the downstream machines. This becomes an important issue when setup time variations are relatively large.

5.2 Experiment 2: Evaluation of Discretization Size and Local Search

In the second set of experiments, the computational burden in `HYBRID()` was increased in an attempt to uncover better solutions. This allowed us to evaluate the trade-off between solution quality and resource usage as measured by CPU time.

In the first phase, we ran `HYBRID()` for three different discretizations of the weight interval: $\Theta^1 = \{0.0, 0.5, 1.0\}$, $\Theta^2 = \{0.0, 0.2, 0.4, \ldots, 1.0\}$, and $\Theta^3 = \{0.0, 0.1, 0.2, \ldots, 1.0\}$. Note that $|\Theta^1| = 3$, $|\Theta^2| = 6$, and $|\Theta^3| = 11$. No local search was performed.

Results for data set D are shown in Table 8. For each category of problem sizes defined by $n \times m$, we give the number of best solutions (or tied for best) found under each discretization and the average CPU time (sec). It can be seen that the finer discretization (Θ^3 compared to Θ^2) produces nearly twice as many superior solutions. This came at the expense of about a 50% increase in CPU time. Nevertheless, the largest instances (10×100) took on average less than four minutes under discretization Θ^3, which is still quite reasonable. Results for data sets A and C (not included) were similar.

The purpose of the second component of experiment 2 was to determine which local search strategy worked best within the `HYBRID()` framework. We used the L-job string reinsertion procedure (denoted by LS) and defined four different implementation strategies, namely

S1: Apply LS($L = 1$)

S2: Apply LS($L = 1$) then LS($L = 2$) and stop

		Discretization		
$m \times n$	Statistic	Θ^1	Θ^2	Θ^3
2×20	Number of best	8	8	20
	Average time (sec)	1	1	2
6×20	Number of best	6	12	20
	Average time (sec)	1	2	3
10×20	Number of best	4	12	20
	Average time (sec)	1	2	4
2×50	Number of best	7	11	20
	Average time (sec)	5	12	22
6×50	Number of best	3	9	20
	Average time (sec)	6	14	27
10×50	Number of best	3	8	20
	Average time (sec)	7	16	31
2×100	Number of best	14	16	20
	Average time (sec)	38	89	175
6×100	Number of best	7	11	20
	Average time (sec)	41	98	196
10×100	Number of best	1	12	20
	Average time (sec)	45	109	220

Table 8: Evaluation of different discretization sizes on class D instances

S3: Apply LS($L = 1$) then LS($L = 2$) then LS($L = 3$) and stop

S4: Apply S3 as many times as necessary until no improvement is possible

Strategy k, $k = 1, 2, 3$, will deliver a local optimum with respect to its respective neighborhood. Strategy 4 delivers a local optimum with respect to all three neighborhoods. The strategies are listed in increasing order of computational effort. The question of interest is whether or not the extra effort pays off in terms of solution quality. Note that in this experiment the local search phase was applied only to the most promising schedule obtained from the construction phase. Of course, it is possible to apply local search to each schedule constructed but this would significantly increase the computational burden. Empirical evidence suggests that such a strategy is rarely justified (see [3, 4]).

Results of this experiment for data set D are shown in Table 9. In each cell we give the number of times a given strategy found the best solution (Nbest), the average relative percentage gap with respect to a lower bound (described in [8]), and the average CPU time. The first thing to notice is that in most cases strategy S4 found solutions that were about 50% better than those found by the other strategies. In 85 out of 180 instances, for example, S4 proved superior to S3. The performance of S4 was even better for the larger instances (in terms of the number of jobs). In terms of relative gap, strategy S4 gives a relative average improvement of 2–10% over S3. This improvement comes at a cost of about a 50% increase in CPU time. The largest

		Strategy			
$m \times n$	Statistic	S1	S2	S3	S4
2×20	Nbest	6	10	13	20
	Average gap (%)	2.7	2.5	2.4	2.2
	Average time (sec)	1	1	1	2
6×20	Nbest	6	9	13	20
	Average gap (%)	10.0	9.8	9.7	9.5
	Average time (sec)	2	2	2	3
10×20	Nbest	12	17	17	20
	Average gap (%)	12.8	12.5	12.5	12.5
	Average time (sec)	3	3	3	3
2×50	Nbest	1	3	5	20
	Average gap (%)	2.5	2.2	2.1	1.9
	Average time (sec)	12	13	14	16
6×50	Nbest	4	6	12	20
	Average gap (%)	8.0	7.8	7.6	7.4
	Average time (sec)	17	19	20	23
10×50	Nbest	8	11	12	20
	Average gap (%)	10.9	10.8	10.7	10.5
	Average time (sec)	23	25	27	31
2×100	Nbest	1	4	6	20
	Average gap (%)	1.9	1.8	1.7	1.6
	Average time (sec)	88	93	96	107
6×100	Nbest	2	5	5	20
	Average gap (%)	6.6	6.3	6.2	6.0
	Average time (sec)	115	126	134	161
10×100	Nbest	7	10	12	20
	Average gap (%)	9.4	9.3	9.2	9.0
	Average time (sec)	156	165	173	206

Table 9: Evaluation of local search strategies on class D instances

average CPU time was associated with the 10×100 instances, as expected, and was less than 4 minutes. The same performance was observed for problem instances from data sets A and C (results not shown).

5.3 Experiment 3: `HYBRID()` vs. `GRASP()`

The purpose of the third experiment was to compare `HYBRID()` with `GRASP()`. GRASP is a heuristic methodology that has proven effective in finding high quality solutions to difficult combinatorial optimization problems, including those involving single- and multi-machine scheduling [3, 5]. A general description of the GRASP methodology is given by Feo and Resende [4]; for a narrower description with respect to the SDST flow shop, see [10].

When comparing two algorithms, it is important to provide a level playing field. `GRASP()`

		Set A	Set D	Set C
`HYBRID()`	Θ	$\{0.0, 0.5, 1.0\}$	$\{0.0, 0.5, 1.0\}$	$\{0.0, 0.2, 0.4\}$
	R	R^1, R^2, R^3	R^1, R^2, R^3	R^1, R^3
`GRASP()`	N	7	7	5
	K	1	1	1
	λ	2	2	2

Table 10: Parameter values used for `HYBRID()` and `GRASP()`

is a randomized heuristic that requires the user to specify the maximum number of iterations. The higher this value is set, the more (and probably better) schedules that may be found. In contrast, `HYBRID()` is a deterministic heuristic, where the degree of diversification determines the maximum number of schedules uncovered. In an attempt to make the comparison as fair as possible, the local search strategy was set to S3 for both heuristics. Table 10 shows the settings used for the analysis. The `GRASP()` parameters are: N (maximum number of iterations) K (iterations between invoking the post-processor), and λ (restricted candidate list size). Under these settings, both heuristics will construct the same number of schedules in phase 1 and will apply local search to each. Recall that for a fixed size of Θ and R in `HYBRID()`, and a fixed size of N in `GRASP()` the computational complexity of the construction phase is the same for both heuristics ($O(mn^2)$). We have observed that in practice both procedures use about the same amount of CPU time, with `HYBRID()` taking about 15–20% more time than `GRASP()` on class A, D instances and `GRASP()` taking about 15–20% more time on class C instances.

	Data Set A	$n = 20$		$n = 50$		$n = 100$	
m	Statistic	H vs. G		H vs. G		H vs. G	
2	Nbest	14	13	13	13	13	7
	Average gap (%)	1.4	1.4	0.9	0.9	0.5	0.7
	Wilcoxon test					H best	
4	Nbest	13	8	8	14	5	16
	Average gap (%)	4.1	4.1	2.4	2.4	1.8	1.7
	Wilcoxon test						
6	Nbest	11	12	7	14	4	16
	Average gap (%)	5.6	5.6	4.8	4.6	3.0	2.8
	Wilcoxon test					G best	
8	Nbest	14	7	9	11	2	18
	Average gap (%)	9.2	9.4	6.4	6.2	4.8	4.3
	Wilcoxon test					G best	
10	Nbest	2	18	5	15	3	17
	Average gap (%)	11.1	10.7	7.2	6.7	6.1	5.6
	Wilcoxon test	G best		G best		G best	

Table 11: Heuristic evaluation for data class A

Tables 11, 12, and 13 display the results for data sets A, D, and C, respectively, in terms of

	Data Set D	$n = 20$		$n = 50$		$n = 100$	
m	Statistic	H vs. G		H vs. G		H vs. G	
2	Nbest	14	7	17	3	19	1
	Average gap (%)	1.8	1.9	1.8	2.0	1.6	2.1
	Wilcoxon test			H best		H best	
4	Nbest	8	13	13	7	13	7
	Average gap (%)	6.0	5.9	4.1	4.1	4.1	4.2
	Wilcoxon test						
6	Nbest	8	12	11	9	9	11
	Average gap (%)	8.8	8.6	7.0	6.9	5.9	5.9
	Wilcoxon test						
8	Nbest	10	11	7	13	5	15
	Average gap (%)	10.2	10.2	7.5	7.4	7.3	7.2
	Wilcoxon test			G best			
10	Nbest	13	9	5	16	4	16
	Average gap (%)	11.9	11.9	10.2	9.9	8.6	8.4
	Wilcoxon test			G best		G best	

Table 12: Heuristic evaluation for data class D

the number of times a given heuristic found the best solution (Nbest) and the average relative gap (Average gap). The third line in each cell indicates which of either of the heuristics was found to be statistically superior after performing the Wilcoxon test at a 99% level of confidence. When the test was not significant (i.e., no heuristic was found to be better than the other) the table entry is blank. For a fixed value of n, it can be observed that `HYBRID()` tends to do better when the number of machines is small. As m gets larger, `GRASP()` generally dominates. For example, for data set D with $n = 50$ and $m = 2$, `HYBRID()` was found to be statistically superior to `GRASP()`. When m took on the values 4, 6, the Wilcoxon test did not find any significant difference between the heuristics. When $m = 8$, 10, `GRASP()` dominated. Similar behavior was observed for data sets A and C.

When comparing performance among the different data classes, it was observed that `GRASP()` tended to do better when setup time fluctuations were small. For the problem instances in class A, for example, `GRASP()` was found to be statistically superior in 5 cases, and `HYBRID()` in 1 case. When we examined class D, `GRASP()` proved superior in only 3 cases, and `HYBRID()` in 2 cases. Finally, for data set C, `HYBRID()` was found to be superior in 10 cases, clearly dominating `GRASP()`.

6 Discussion and Conclusions

The first observation that can be made from the computational results is that `HYBRID()` generally outperforms `GRASP()` when the number of machines is small. Another favorable scenario

	Data Set C	$n = 20$		$n = 50$		$n = 100$	
m	Statistic	H vs. G		H vs. G		H vs. G	
2	Nbest	15	5	20	0	20	0
	Average gap (%)	6.9	7.4	6.0	7.2	4.9	7.0
	Wilcoxon test			H best		H best	
4	Nbest	11	9	20	0	20	0
	Average gap (%)	12.3	12.5	13.3	14.5	12.0	14.0
	Wilcoxon test			H best		H best	
6	Nbest	12	8	19	1	20	0
	Average gap (%)	16.2	16.5	16.0	17.0	15.8	17.5
	Wilcoxon test			H best		H best	
8	Nbest	11	9	18	2	20	0
	Average gap (%)	17.4	17.3	19.1	19.8	18.3	20.1
	Wilcoxon test			H best		H best	
10	Nbest	14	6	16	4	20	0
	Average gap (%)	18.8	19.1	20.9	21.5	20.7	21.9
	Wilcoxon test			H best		H best	

Table 13: Heuristic evaluation for data class C

for `HYBRID()` is when setup time fluctuations are large (data set C). This stems from the fact that the fewer the number of machines and/or the larger the magnitude of the setup times, the more the problem resembles an ATSP so a TSP-based procedure should do well. Recall that in `HYBRID()` the distance between jobs has a setup time cost component which is computed as the sum of the setup times between jobs over all machines. In the extreme case where there is only one machine, the problem reduces to an instance of the ATSP. As more machines are added, the weighted cost function becomes less representative of the "distance" between the jobs.

The maximum number of machines in the problem for `HYBRID()` to do better than `GRASP()` depends not only on the number of jobs, but on the magnitude of the setup times as well. For data class D, a threshold value of $m = 4$ was observed for the 50- and 100-job instances. However, for data class C (larger setup times), `HYBRID()` was found to outperform `GRASP()` with respect to both makespan (especially for the 50- and 100-job data sets) and CPU time. This implies a threshold value of $m > 10$.

Another way to explain the better performance of `HYBRID()` on the larger instances is as follows. An insertion-based heuristic like `GRASP()` includes a makespan estimation routine that has setup costs as part of its performance measure; this is the only explicit treatment of setups in the heuristic. Because the job insertion decision is made one job at a time, while the sequence-dependent setup time is dictated by the interrelationships of an entire sequence of jobs, a TSP-based heuristic should to do better when the number of machines is small. In such cases, the similarities between the SDST flow shop and the ATSP are strongest.

In terms of solution quality, `HYBRID()` delivered average optimality gaps of 1.6% for the

2-machine, 100-job instances in data set D. As the number of machines increases, the average gap increased as well up to around 8.6% for the 10-machine instances. Consequently, looking at data class A (C), smaller (larger) setup time variations resulted in smaller (larger) optimality gaps.

An advantage of `GRASP()`, of course, is that by increasing the iteration counter, more and perhaps better solutions can be found. The scheduler must make this trade-off in light of the specific time constraints he or she faces. When the ultimate goal is to develop exact optimization methods, however, both procedures can be combined to yield better upper bounds. This is critical in speeding convergence. In [8], both heuristics are used within a branch-and-bound scheme with notable success.

In summary, the hybrid heuristic did a good job in exploiting the similarities between the SDST flow shop and the TSP. The overall results are promising. In terms of future work, it would be worthwhile to investigate alternate cost functions and to develop other non-TSP-based algorithms. In fact, the proposed cost function can be used for developing an effective diversification strategy within memory-based meta-heuristic schemes such as tabu search.

7 Acknowledgments

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

We would also like to thank two anonymous referees for their suggestions, which helped improve the presentation of this work.

References

[1] A. Allahverdi, J. N. D. Gupta, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 1998. Forthcoming.

[2] W. Conover. *Practical Nonparametric Statistics.* John Wiley & Sons, New York, 1980.

[3] T. A. Feo, J. F. Bard, and K. Venkatraman. A GRASP for a difficult single machine scheduling problem. *Computers & Operations Research*, 18(8):635–643, 1991.

[4] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[5] M. Laguna and J. L. González-Velarde. A search heuristic for just-in-time scheduling in parallel machines. *Journal of Intelligent Manufacturing*, 2:253–260, 1991.

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

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

[8] R. Z. Ríos-Mercado and J. F. Bard. A branch-and-bound algorithm for the permutation flow shop scheduling problem with sequence-dependent setup times. *IIE Transactions*, 1998. Forthcoming.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] J. P. Stinson and A. W. Smith. A heuristic programming procedure for sequencing the static flowshop. *International Journal of Production Research*, 20(6):753–764, 1982.

[15] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74, 1990.

[16] M. Widmer and A. Hertz. A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research*, 41(2):186–193, 1989.