Three Heuristic Procedures for the Stochastic, Two-Machine Flow Shop Problem

Kenneth R. Baker • Dan Trietsch

Abstract. Although the deterministic flow shop model is one of the most widely studied problems in scheduling theory, its stochastic analog has remained a challenge. No computationally efficient optimization procedure exists even for the general two-machine version. In this paper, we describe three heuristic procedures for the stochastic two-machine flow shop problem and report on computational experiments that compare their effectiveness. We focus on heuristic procedures that can be adapted for dispatching without the need for computer simulation or computer-based search. We find that all three are capable of quickly generating solutions close to the best known sequences, which were obtained by extensive search.

1 Introduction

The analysis of stochastic flow shop problems has not penetrated very far and remains challenging. Analytic results for the stochastic flow shop have essentially been limited to the expected completion time of all jobs—the *makespan*—as a performance measure, and much of the work addresses only the two-machine problem. In the deterministic case, the minimization of the makespan for the two-machine model is well solved, and a vast literature exists on multi-machine models and on other performance measures. In the stochastic case, the two-machine makespan problem is inherently more complex than its deterministic counterpart. This paper presents an experimental comparison of three heuristic procedures for minimizing the expected makespan in the stochastic, two-machine case.

After introducing the two-machine flow shop model, we review the few theoretical results that exist for the stochastic case and thereby lay the groundwork for the heuristic procedures that are the focus of the study. Then we describe our computational experiments and review our findings. In short, we find that all three heuristics perform effectively and that none dominates the others. We also identify conditions under which each heuristic tends to perform systematically well.

The heuristic procdures that are central to our study exhibit a distinctive feature. They allow a schedule to be created by comparing the numerical traits of the jobs in the schedule. That is, they use job characteristics to construct a complete sequence. This type of heuristic should be distinguished from well-known general search methods such as neighborhood search, simulated annealing, and genetic algorithms. Those general-purpose procedures require comparisons among candidate sequences on the basis of the performance they achieve. For the expected makespan in the stochastic flow shop, that essentially means running a computer simulation of the schedule with a large number of trials for each candidate sequence, so that differences among sequences can be reliably estimated. In fact, the search procedure itself, in addition to the evaluation of schedules, requires extensive computer time (and specialized software). In sharp contrast, the heuristic procedures in our study can be implemented without computer support, making them very practical.

In the two-machine flow shop, we face the problem of scheduling n jobs that are simultaneously available at time zero. Each job requires processing first at Machine A and then at Machine B. Processing on Machine B cannot begin on a job until its processing on Machine A has completed and the preceding job has completed its processing on Machine B. For job j, we use a_j and b_j to denote processing times on Machines A and B, respectively. Assuming for the moment that the jobs are processed in numbered order, we have for job j:

Completion time on Machine A: $C_{aj} = \sum_{i=1}^{j} a_i$

Completion time on Machine B: $C_{bj} = \max\{C_{aj}, C_{b,j-1}\} + b_j$

The makespan is then C_{bn}.

The optimal schedule is a permutation schedule (in which the same sequence occurs on both machines). In the deterministic case, the optimal job sequence may be determined by Johnson's Rule (Johnson, 1954): job i precedes job j in an optimal sequence if $\min\{a_i, b_j\} \leq \min\{a_j, b_i\}$. This condition can be translated into an efficient sequencing algorithm that is computationally equivalent to sorting the jobs. Thus, it lends itself readily to implementation as a dispatching rule.

2 Stochastic counterpart models

In the stochastic case, we use A_j and B_j to denote (random) processing times on the two machines, but we retain a_j and b_j to represent their expected values. We assume that the probability distributions of the processing times are known and that the processing times are independent random variables. Usually, the problem is specified by assuming a particular family of distributions, such as the normal or the exponential distribution, and then specifying the parameters of that family as they apply to each processing time. The objective in the stochastic problem is to minimize the expected value of the makespan. We refer to this problem as the *stochastic counterpart* of the two-machine flow shop problem.

As in the deterministic case, we can limit our attention to permutation schedules (Ku and Niu, 1986). However, for general distributions without special conditions on processing times, only one full solution is known for the stochastic counterpart, given by Makino (1965).

Theorem 1 In the two-job, two-machine stochastic flow shop problem, job 1 precedes job 2 in an optimal sequence if $\mathrm{E}(\min\{A_1, B_2\}) \leq \mathrm{E}(\min\{A_2, B_1\})$.

Theorem 1 generalizes Johnson's Rule for two jobs in the deterministic case. Although the theorem holds for any joint processing time distributions, the calculations become more tractable if we assume independent processing time distributions. With independent distributions, we can express the cumulative distribution function (cdf) of the minimum of two random variables X and Y by using the formula

$$F_{min}(t) = 1 - (1 - F_X(t))(1 - F_Y(t)), \tag{1}$$

where $F_X(t)$, $F_Y(t)$ and $F_{min}(t)$ denote the cdfs of X, Y and their minimum. However, this formula does not lead to a closed-form calculation of the minimum for all distributions, so Theorem 1 is difficult to apply in general. Furthermore, the restriction to two jobs is difficult to overcome. For instance, Elmaghraby and Thoney (1999) studied a case in which processing times have two possible realizations each, with equal probabilities. Using a branch and bound approach, eight-job instances took dozens of hours to solve (and would remain intractable even with more modern computers).

To make further progress with the analysis, we must make a more specific assumption about the distribution of processing times. To start, consider the case of exponential distributions for processing times. (The exponential is not necessarily very practical as a model for job processing times, but it is significant as a boundary case, and it sometimes provides us with general insights.) Suppose we have two exponential random variables, one with a mean of a and the other with a mean of b. Equivalently, we say that one has a *processing rate* (or *completion rate*) of $1/a$ and the other has a rate of $1/b$. The minimum of the two exponential random variables is an exponential random variable with processing rate of $(1/a + 1/b)$. Therefore, the condition in Theorem 1 can be rewritten for the exponential case. The condition for job 1 to precede job 2,

$$\mathrm{E}(\min\{A_1, B_2\}) \leq \mathrm{E}(\min\{A_2, B_1\}), \tag{2}$$

implies the reverse ordering of processing rates, or

$$1/a_1 + 1/b_2 \geq 1/a_2 + 1/b_1 \tag{3}$$

Algebraic rearrangement yields:

$$1/a_1 - 1/b_1 \geq 1/a_2 - 1/b_2 \tag{4}$$

In words, the job with the larger difference in processing rates should come first.

Example 1 Consider the scheduling of two jobs with independent exponentially-distributed processing times. The mean of each processing time is given in the following table.

Machine	Job 1	2
1	4	5
2	5	8

The processing rates associated with these means are 0.25 and 0.2 for job 1 and 0.2 and 0.125 for job 2. Since the difference for job 2 (0.0875) exceeds the difference for job 1 (0.05), job 2 should be scheduled first. This sequence reverses the solution of the deterministic counterpart and demonstrates that in a stochastic problem, applying Johnson's Rule to the mean values does not necessarily produce an optimal sequence. For this example, the expected makespan is 19.333 for the sequence (2-1) and 19.5 for (1-2). The standard deviations are 10.424 and 10.548, respectively. The difference in both measures is on the order of 1%, and the optimal sequence is advantageous on both counts.

Example 2 Consider the scheduling of two jobs with independent exponentially-distributed processing times. The mean of each processing time is given in the following table.

Machine	Job 1	2
1	1	1
2	1	$1+\sqrt{7}$

In Example 2, the calculations yield an expected makespan of 5.861 for the sequence (2-1) and 6.146 for (1-2). The standard deviations are 3.801 and 3.942, respectively. For the optimal sequence, the advantage in the mean is about 4.8%, and the advantage in the standard deviation is about 3.7%. The parameters of this example illustrate the maximum percentage improvement of the correct sorting rule for a two-job problem with exponential processing times. Johnson's Rule, when applied to the means, results in a tie, but if we reduce a_1 infinitesimally, the sequence (1-2) becomes the deterministic optimum, whereas the sequence (2-1) remains the optimal solution.

Sorting by the difference in mean processing rates $(1/a_j - 1/b_j)$ involves a transitive sequencing relation and lends itself readily to dispatching. The idea can also be extended to n jobs. In the exponential case, this sorting rule, known as Talwar's Rule, also turns out to be optimal. Talwar (1967) originally conjectured optimality, although the first proof was obtained by Cunningham and Dutta (1973). Ku and Niu (1986) showed that Talwar's rule is not only optimal for the stochastic case but also yields a stochastically minimal makespan.

Theorem 2 (Talwar's Rule) In the stochastic two-machine flow shop problem with exponential processing times, the makespan is minimized stochastically by sequencing the jobs in nonincreasing order of $(1/a_j - 1/b_j)$.

To summarize the state of the art in stochastic flow shop problems, we have two theoretical results, namely Theorems 1 and 2. Theorem 1 is a distribution-free result, but it holds only for the two-job problem. Theorem 2 holds for the exponential distribution and provides us with a transitive rule for sequencing the jobs optimally.[1] For practical solutions to a stochastic flow shop problem in which the processing times are not exponential, we shall have to rely on a heuristic scheduling procedure unless we are willing to wait for new theory to be developed.

3 Heuristic procedures

When the problem involves distributions other than the exponential, heuristic procedures become relevant because no result as strong as Theorem 2 has been developed for any other family of processing time distributions. One plausible heuristic is to follow Talwar's Rule and sort the jobs by nonincreasing differences in processing rates. This approach makes sense if we believe that the analysis of the exponential case captures relevant information about the problem in general. (As

[1]Nevertheless, the computational effort required to calculate the expected makespan, even when we know the optimal sequence, grows exponentially with the problem size. Even in the case of exponential distributions, where the memoryless property facilitates the computations, the number of subcases to be evaluated is of the order 2^{n-2}.

mentioned earlier, however, the exponential is a boundary case. It is a one-parameter distribution with standard deviation equal to its mean and is therefore not necessarily a good model for the kinds of stochastic behavior found in practice.) Thus, our first heuristic solution procedure might be called *Talwar's Heuristic*: sequence the jobs in nonincreasing order of the difference between their two processing rates.

Another heuristic, of course, is to solve the deterministic counterpart. This approach suppresses the randomness in the problem and replaces the stochastic processing times with deterministic equivalents. A general method for solving a stochastic scheduling problem is to replace its stochastic variables by their means and proceed with deterministic analysis to find a sequence. This approach is commonly found in practice, especially in applications where data are scarce and distribution information is difficult to obtain. We refer to this heuristic procedure as *Johnson's Heuristic*. Portougal and Trietsch (2006) showed that Johnson's Heuristic is asymptotically optimal under mild regularity conditions—that is, as n grows large, the relative difference between the heuristic makespan and the optimal makespan tends to zero. However, little is known about how the heuristic performs for problems of small to moderate sizes.

In a sense, both Talwar's Heuristic and Johnson's Heuristic presume that the mean of the distribution represents all the information we need to know about the probability model in order to make a sound decision about job sequence. Neither approach recognizes variability apart from the estimate of the mean in order to influence job sequence. By contrast, a heuristic procedure that looks beyond mean-value information seems desirable. Such an approach, based on the properties of an adjacent pairwise interchange (API), has been developed by Portougal and Trietsch. Their API procedure starts with the Johnson sequence and employs an API neighborhood search. In principle, however, any other sequence could be used to initialize the procedure. We call a pair of adjacent jobs *stable* if they satisfy Theorem 1. We call a sequence stable if every pair of adjacent jobs is stable. The API procedure checks all adjacent job pairs in the initial sequence, and if an unstable pair is found, the two jobs are interchanged. After the interchange, the upstream (earlier) job may not form a stable pair with its new upstream neighbor, in which case we interchange those jobs, too. After each interchange, the job that was moved downstream is stable with respect to its upstream neighbor, but the one moved upstream may have to be interchanged repeatedly until it reaches a stable position, possibly as the first job. Then, the heuristic resumes its check of all pairs and stops when all adjacent pairs are stable.

The API procedure requires $O(n^2)$ tests and always results in a stable sequence. However, the optimal sequence may not be stable, and more than one stable sequence can exist (see Baker and Trietsch, 2009). The heuristic is guaranteed only to converge to *some* stable sequence, but that does not mean it will locate the best one.

The key step in the API procedure tests Theorem 1 as it applies to any pair of adjacent jobs. Equivalently, the condition in (2) must be evaluated. However, as we pointed out earlier, the formulas in that evaluation might be intractable for many types of probability distributions. In those cases, we can approximate the desired evaluation by assuming a particular family of distributions for all processing times. If we were to assume an exponential distribution for each processing time, we would generate the sequence of Talwar's Heuristic. Instead, we adopt a more flexible approach and evaluate (2) by assuming the form of a normal distribution. The normal distribution is a two-parameter distribution, unlike the exponential, so we can match its mean and standard deviation with those of the distributions in the problem.

Next, we describe the special-case calculation of (2) for the normal distribution. In general, for any random variables X and Y,

$$\min\{X, Y\} = X - \max\{X - Y, 0\}$$

and

$$\mathrm{E}[\min\{X, Y\}] = \mathrm{E}[X] - \mathrm{E}[\max\{X - Y, 0\}].$$

Define $W = X - Y$ so that

$$\mathrm{E}[\min\{X, Y\}] = \mathrm{E}[X] - \mathrm{E}[W^+].$$

Suppose that X and Y are independent normal random variables. Then, W is normal with mean $\mu_{xy} = \mu_x - \mu_y$ and variance $\sigma_{xy}^2 = \sigma_x^2 + \sigma_y^2$. In addition, $\mathrm{E}[W^+] = \sigma_{xy}[\varphi(z_{xy}) + z_{xy}\Phi(z_{xy})]$, where φ and Φ denote the density function and the cumulative distribution function of the standard normal and $z_{xy} = \mu_{xy}/\sigma_{xy}$. The expected minimum (see Baker and Trietsch (2009), pp. 468-69 for details) is then given by

$$\mathrm{E}[\min\{X, Y\}] = \mu_x - \sigma_{xy}[\varphi(z_{xy}) + z_{xy}\Phi(z_{xy})]$$

Therefore, the condition for job 1 to precede job 2 takes the following form.

$$\mu_{x1} - \sigma_{x1y2}[\varphi(z_{x1y2}) + z_{x1y2}\Phi(z_{x1y2})] \leq \mu_{x2} - \sigma_{x2y1}[\varphi(z_{x2y1}) + z_{x2y1}\Phi(z_{x2y1})] \qquad (5)$$

This form of the API procedure uses the properties of normal distributions, but it can be used as a heuristic when the distributions are non-normal, as long as we know their means and variances. Implementing the API procedure based on the calculations in (5) gives rise to what we call the *API Heuristic*. The stable sequence that it produces also lends itself readily to dispatching.

Example 3 Consider the six-job, two-machine flow shop problem shown in Table 1, in which the jobs have normally-distributed processing times on the two machines, A and B.

Table 1. Example problem.

Job	1	2	3	4	5	6
mean A	12.3	17.2	17.6	13.9	16.4	12.7
s.d. A	1.6	1.6	1.2	2.0	2.4	1.5
mean B	14.7	13.1	17.6	12.2	14.1	12.3
s.d. B	1.7	1.4	1.6	2.5	1.4	2.1

We illustrate the algorithm by starting with the sequence 1-2-3-4-5-6. Using (5), Table 2 records the differences between E[min{A_i, B_j}] and E[min{A_j, B_i}] for these jobs, with job i corresponding to a row and j to a column. For instance, the calculations for E[min{A_1, B_2}] yield $\mu_{x1} = 12.3$, $\sigma_{x1y2}^2 = 1.6^2 + 1.4^2 = 2.13^2$, so $z_{x1y2} = (12.3 - 13.1)/2.13 = -0.3763$, leading to E[min{$A_1$, B_2}] = $12.3 - 2.13[0.3717 + (-0.3763) \times 0.3534] = 11.792$. Similarly, E[min{$A_2$, B_1}] = 14.530. The first entry in the table is the difference $11.792 - 14.530 = -2.74$. For the algorithm, we need to know only the sign of this calculation. In this case, the negative value indicates that the first two jobs are stable in the sequence 1-2.

Table 2. Differences between E[min{A_i, B_j}] and E[min{A_j, B_i}] in the example.

	1	2	3	4	5	6
1		-2.74	-2.33	-2.14	-2.12	-1.22
2	2.74		3.39	-0.33	1.09	0.20
3	2.33	-3.39		-1.64	-1.67	-0.40
4	2.14	0.33	1.64		1.01	0.50
5	2.12	-1.09	1.67	-1.01		-0.25
6	1.22	-0.20	0.40	-0.50	0.25	

Suppose now that we start arbitrarily with the sequence 1-2-3-4-5-6. As indicated above, jobs 1 and 2 are stable in this sequence. However, jobs 2 and 3 are not, so we interchange them, observing that jobs 1 and 3 are stable, so we obtain the sequence 1-3-2-4-5-6. Next, jobs 4 and 5 are not stable, so we interchange them and then interchange jobs 2 and 5, yielding the sequence 1-3-5-2-4-6. Next, jobs 4 and 6 are not stable, so we interchange them and then interchange jobs 2 and 6 yielding the stable sequence 1-3-5-6-2-4.

For the parameters of Example 3, Johnson's Heuristic yields the sequence 1-3-5-2-6-4, and Talwar's Heuristic yields the sequence 1-3-6-5-4-2. Thus, the three heuristics produce three different sequences. In other instances, they may produce two different sequences or they may all produce the same sequence.

4 Experimental methodology

We devised a set of experiments to compare the performance of the heuristic procedures introduced in the previous section. The experiments consisted of a set of stochastic flow shop problems in which the objective is to find the minimum expected makespan. To specify an instance of the problem, we had to specify the number of jobs and the (two) processing time distributions for each job. For simplicity, we assumed that all 2n processing time distributions were members of the same family, such as exponential, uniform, or lognormal. The exponential distribution is specified by a single parameter (its mean value) and corresponds to the case in which Talwar's Heuristic produces an optimal schedule. The uniform distribution is specified by

two parameters (e.g., a mean value and a range) and allows us to influence the extent to which processing time distributions overlap. Finally, the lognormal distribution is the basis for most of the experimental work, for two reasons. First, the lognormal can also be specified by two parameters (a mean value and a standard deviation), allowing us to explore the effects of each. Second, the lognormal has considerable practical value as a model for actual processing times. Among other traits, a lognormal variate is nonnegative, and its parameters can represent both high and low coefficients of variation.

The first step in generating a test problem was to obtain the parameters (the mean and, as needed, the standard deviation) for each of the 2*n* processing time distributions in the problem. These parameters were drawn randomly as samples from a pre-specified interval. For example, we might obtain the mean processing times for six jobs by sampling 12 times from the interval between 10 and 20 and the standard deviations by sampling 12 times from the interval between 5 and 10.

Once the parameters of the processing time distributions were determined, the next step was to construct the job sequence using one of the heuristic methods.

- To employ Talwar's Heuristic, the difference in processing rates was calculated for each job. Then a sequence was constructed in which these differences appeared in nonincreasing order.

- To employ Johnson's Heuristic, the mean values were treated as deterministic processing times. Then the desired sequence was constructed using Johnson's Rule.

- To employ the API Heuristic, the values of the mean and standard deviation for each processing time were used in the pairwise comparisons of (5), and a stable sequence was constructed. The process was initialized arbitrarily with the jobs in numerical order.

For comparison purposes, we also evaluated the sequence with jobs in numerical order. We can think of this sequence as "random" in that it uses no information about the processing time distributions in ordering the jobs and represents the performance we might expect from a naïve scheduler.

The next step was to estimate the mean value of the makespan for each of the four sequences. This estimate was based on a simulation containing 100,000 trials. In addition, we used a fifth algorithm to search for an optimal or nearly-optimal benchmark sequence: the Evolutionary Solver.[2] To initialize the Evolutionary Solver, we used the best schedule found by the other four methods. Although optimality could not be guaranteed, no computationally practical method for finding the optimal solution is known, and we were mostly interested in comparing the three main heuristic methods. Since each sequence was evaluated by extensive simulation, the Evolutionary Solver can require significant computing time to converge in large problems; hence, its practical value in such instances may be somewhat limited, whereas our heuristics can be implemented virtually by hand.

This procedure was repeated (by sampling again for the parameters of the processing time distributions) ten times. For the ten replications, we recorded the average deviation between the heuristic procedure's estimated mean and the best value found (referred to as the "average suboptimality"), as well as the number of times (out of 10) that each heuristic produced the best value.

In the remainder of the paper, we present several tables describing summaries of our various experiments. Here, we describe the notation in our tables.

- The top row of the table, labeled *Jobs*, shows the problem size and the family of distributions from which processing times were sampled.
- The second row, labeled *Means*, shows the interval of values from which the various mean values were sampled.
- The third row, labeled *Ranges* in the case of uniform distributions, shows the range of the individual processing time distributions. (For example, a mean of 10 and a range of 2 would indicate that the samples are drawn from U[9,11].) This row is omitted in the exponential case because the mean value describes the variability in the distribution. In the case of the

[2]The Evolutionary Solver is a proprietary genetic algorithm which appears to be particularly effective at solving sequencing problems. It is available as an Excel add-in, part of Frontline Systems' software package, Risk Solver Platform (see www.solver.com).

lognormal distribution, this row is labeled σ for the standard deviation of the distribution or *S.D.'s* for the interval of standard deviations sampled.

- The remaining rows are labeled *R*, *T*, *J*, and *A*, corresponding to Random, Talwar's Heuristic, Johnson's Heuristic, and the API Heuristic, respectively. Across each row, the table contains four pairs of numbers, each pair giving our measure of average suboptimality and the number of times (out of 10) the procedure generated the best value.

5 Experimental Results

To illustrate the type of data we compiled, we review one of our validation experiments, involving 6-job problems. In this experiment, we used the uniform distribution as a model for processing times. We drew mean processing times as integers from the interval (40, 60) and we took the range of the processing times to be 1. Thus, except for the possibility that the means of two processing time distributions might match, the distributions did not overlap. Equivalently, during the simulation, if one processing time was longer than another on one simulated outcome, it had to be longer on all simulated outcomes. As a consequence, Johnson's Heuristic—applied to the mean processing times—should always produce an optimal sequence. We see that this is indeed the case in the first portion of Table 3.

Table 3. Basic results for six-job problems with uniform processing times.

Jobs	n = 6	Uniform						
Means	40-60		40-60		40-60		40-60	
Ranges	1		10		20		30	
R	3.42%	1	2.76%	0	3.32%	0	2.57%	0
T	0.00%	9	0.00%	7	0.08%	2	0.15%	3
J	0.00%	10	0.00%	8	0.04%	5	0.14%	5
A	0.00%	9	0.00%	7	0.09%	2	0.16%	3

In the table, we show the average suboptimality of the estimated mean makespan over ten replications, measured as the deviation from the best value found. For the case in which the range of processing times was equal to 1, Johnson's Heuristic (J) produced the best expected makespan each time (10 out of 10), as expected, so its average deviation was zero. Talwar's Heuristic (T) produced the best value 9 times out of 10, and its average deviation was 0.0003%, shown in the table to two decimal places. The API Heuristic (A) also produced the best value 9 times out of 10, and its average deviation was 0.0005%. The random sequence (R) produced the best value once and was on average about 3.4% worse than the best.

Having verified that our tests confirmed Johnson's Heuristic as the best when the processing time distributions did not overlap, we proceeded to increase the range of the processing time distributions, first to 10 and then to 20 and 30, repeating the experiment each time. The rest of Table 3 shows the results. For a range of 10, Johnson's Heuristic produced an average deviation of 0.002%, but as the range of the distribution was increased, performance declined. This pattern suggests that Johnson's Heuristic is optimal when the processing time distributions do not overlap, but it becomes less effective as overlap increases. As the table shows, Talwar's Heuristic and the API Heuristic exhibited somewhat similar behavior, but were perhaps slightly less likely than Johnson's Heuristic to produce the best sequence.

For a different perspective, we took an alternative parametric approach using a fixed range but sampling means from different intervals, as summarized in Table 4. With the range held fixed, the potential for overlap declines as we increase the dispersion of the mean values. Thus, we again see that Johnson's Heuristic fared best (with average suboptimality of 0.0002%) when means were drawn from the interval (20, 80), in which case the distributions were unlikely to overlap, but its performance was not quite as good when there was more overlap. Again, the pattern roughly held for the other heuristics. This general picture also emerged in experiments with more jobs.

Table 4. Results for uniform processing times and means from different intervals.

Jobs	n = 6	Uniform						
Means	45-55		40-60		30-70		20-80	
Ranges	20		20		20		20	
R	1.95%	0	2.36%	0	9.04%	1	8.15%	0
T	0.11%	0	0.08%	2	0.00%	7	0.09%	8
J	0.03%	3	0.04%	6	0.04%	9	0.00%	9
A	0.26%	0	0.08%	2	0.00%	7	0.09%	8

Table 4 also suggests that the performance of the Random sequence becomes relatively worse as the dispersion among the mean values increases. In the worst case, the estimated deviation for the Random sequence was more than 18%, a reminder that intelligent heuristics can sometimes provide a substantial improvement over naïve scheduling in the stochastic flow shop.

In another validation experiment, we used the exponential distribution as a model for processing times. We drew mean processing times as integers from the interval (10, 20) and in subsequent experiments, (10, 30), (10, 40), and (10, 50). In these cases, of course, all distributions overlap, a condition we expect to be unfavorable for Johnson's Heuristic. Moreover, from Theorem 2, Talwar's Heuristic is known to be optimal for the exponential case. Table 5 shows our results, which essentially confirm these expectations. (We suspect that the two instances in which Talwar's Heuristic was slightly suboptimal simply reflect that fact that a finite simulation cannot perfectly reproduce the exponential distribution.) In these experiments, the API Heuristic produces better results on average than Johnson's Heuristic.

Table 5. Results for six-job problems with exponential processing times.

Jobs	n = 6	Exponential						
Means	10-20		10-30		10-40		10-50	
R	3.28%	0	3.85%	0	7.96%	0	7.50%	0
T	0.00%	10	0.00%	10	0.00%	9	0.00%	9
J	0.28%	3	0.15%	4	0.15%	2	0.20%	3
A	0.10%	6	0.03%	7	0.11%	2	0.16%	4

Thus, our validation experiments confirmed the two cases in which one of the heuristic procedures is known to be optimal (Johnson's Heuristic when the distributions do not overlap; Talwar's Heuristic when the distributions are exponentials). Otherwise, the relative performance is mixed. Because that is the case, we might anticipate that some parametric cases will favor one heuristic, whereas other cases will favor another. Our task is to explore the performance in various cases, rather than to find an overall "best heuristic."

Another observation from these preliminary experiments is that the average suboptimality for the three heuristic procedures is quite small—typically under 1%. The two-machine stochastic flow shop problem therefore seems to present a case in which heuristic procedures can provide very good solutions.

6 Main Results

For our main experiments, we used a lognormal distribution for processing times and tested a variety of 6-job and 10-job problems, representing small and medium size instances. (We did not experiment with large instances because the Johnson Heuristic is asymptotically optimal. In addition, it would be very difficult to generate benchmark solutions for large instances.) We encountered no evidence that problem size was an important factor in the relative performance of the various heuristic methods, so we report only the 10-job results. Table 6 summarizes results for cases in which all processing time distributions have means drawn from the interval (40, 60), and in each instance have the same standard deviation. For relatively small standard deviations, such as $\sigma = 1$, Johnson's Heuristic performs best, probably because very little overlap exists. As σ increases, the performance of Johnson's Heuristic deteriorates, whereas Talwar's Heuristic improves. Thus, the two heuristics show complementary performance. The API Heuristic is robust in the sense that it provides good makespans at both extremes.

Table 6. Basic results for 10-job problems with lognormal processing times.

Jobs	n = 10	Lognormal								
Means	40-60		40-60		40-60		40-60		40-60	
σ	1		5		10		20		40	
R	2.98%	0	2.77%	0	3.51%	0	3.22%	0	1.80%	0
T	0.34%	3	0.15%	2	0.10%	2	0.08%	4	0.07%	4
J	0.00%	9	0.03%	1	0.10%	1	0.20%	1	0.42%	0
A	0.00%	10	0.00%	6	0.00%	8	0.07%	4	0.09%	3

Table 7 summarizes results for cases in which the standard deviation of the lognormal distribution was kept at $\sigma = 10$ and the means were drawn from increasingly wider intervals. When the means were widely dispersed (e.g., from 10 to 90) there was less overlap in the distributions,

and Johnson's Heuristic performed well. On the other end of the spectrum, with little dispersion, Talwar's Heuristic performed well. The API Heuristic performed well throughout, again demonstrating robust response to the experimental conditions.

Table 7. Results for lognormal processing times with constant standard deviations.

Jobs	n = 10	Lognormal								
Means	45-55		40-60		30-70		20-80		10-90	
σ	10		10		10		10		10	
R	1.77%	0	3.22%	0	5.78%	0	8.66%	0	8.14%	0
T	0.04%	4	0.05%	2	0.06%	3	0.12%	3	0.06%	3
J	0.15%	3	0.13%	3	0.07%	0	0.04%	2	0.01%	5
A	0.02%	3	0.03%	4	0.01%	7	0.00%	6	0.01%	5

Table 8 displays the results for a set of experiments in which increasing levels of diversity characterized both the dispersion of the means and the size of the standard deviations of the processing time distributions. Across this spectrum, the results are mixed, and the API Heuristic seldom outperforms either of the other two heuristics. Average suboptimalities are larger and the frequencies of identifying the best solution are lower. Apparently, greater variability in the data leads to diminished performance among the heuristic procedures.

Table 8. Results for lognormal processing times and increasing variability.

Jobs	n = 10	Lognormal								
Means	45-55		40-60		30-70		20-80		10-90	
S.D.s	10-20		10-30		10-40		10-50		10-60	
R	1.80%	0	2.79%	0	4.34%	0	5.05%	0	5.67%	0
T	0.07%	1	0.17%	1	0.10%	4	0.23%	3	0.05%	4
J	0.20%	1	0.15%	3	0.38%	1	0.28%	1	0.15%	2
A	0.14%	1	0.29%	1	0.52%	0	0.29%	1	0.20%	4

Table 9 shows a different perspective on the same data set. Here, we show the worst-case deviation in each set of ten replications, along with the estimated standard deviation of the makespan, averaged over the ten replications. Both measures represent ways of quantifying "risk" in making decisions under uncertain conditions. Talwar's Heuristic tends to exhibit the best worst-case performance, but in general the worst case is seldom above 1%. The average standard deviations are all quite close. Thus, the three heuristics do not appear to have substantially different properties when we consider decision-making under risk.

Table 9. Worst-case deviations and estimated makespan standard deviations

Jobs	n = 10	Lognormal								
Means	45-55		40-60		30-70		20-80		10-90	
S.D.s	10-20		10-30		10-40		10-50		10-60	
R	2.82%	49.6	4.60%	68.5	11.00%	91.2	12.18%	106.4	17.59%	133.1
T	0.14%	48.3	0.29%	68.0	0.47%	89.9	0.95%	104.0	0.15%	135.9
J	0.37%	48.2	0.36%	67.7	1.14%	91.2	0.81%	107.8	0.63%	132.6
A	0.26%	49.6	0.54%	68.9	2.25%	92.4	0.74%	108.5	0.64%	124.7

7 Secondary Results

Talwar's Heuristic and Johnson's Heuristic both construct a job sequence from scratch. The API Heuristic is initialized with a sequence and attempts to improve on it. (However, it does not measure improvement directly and therefore does not entail the same computing requirements as a search procedure.) In our experiments, the initial sequence was arbitrary, but we might wonder how the API Heuristic would perform if it were initialized with a sequence that was already a good one. To explore this question, we also tested a form in which Talwar's Heuristic and the API Heuristic were used in tandem and another form in which Johnson's Heuristic and the API Heuristic were used in tandem. The initial sequence does not make any difference if only one stable sequence exists, but that condition does not always hold.

We found that different initial sequences could occasionally affect the performance of the API Heuristic, but not always for the better. In the experiments with lognormal distributions, different initialization led to a difference in the solution value for fewer than 10% of the test

problems, and the difference was not always an improvement. We concluded that the overall performance of the API Heuristic was not very sensitive to initialization.

A related question is whether the API Heuristic can improve on Talwar's Heuristic when the two procedures are used in tandem, and similarly for Johnson's Heuristic. We found that the API Heuristic improved on Talwar's Heuristic about 40% of the time in the lognormal test data, but it converged to a worse solution about 30% of the time. Meanwhile, the API Heuristic improved on Johnson's Heuristic about 20% of the time but converged to a worse solution about 50% of the time. The tandem structures were most effective when the dispersion among the mean processing times was small.

Talwar's Heuristic and Johnson's Heuristic are computationally efficient, in that their complexity is $O(n\log n)$. The API Heuristic is slightly more demanding, at $O(n^2)$. For a given problem, we can almost imagine implementing the heuristic procedures with hand calculations. However, the Evolutionary Solver is quite different. It requires explicit evaluation of the expected makespan (by simulation), together with a computationally-intensive search procedure. We might wonder how effective the Evolutionary Solver is by comparison.

We initialized the Evolutionary Solver with the best sequence found by the other heuristics. This design meant that the Evolutionary Solver always produced the best sequence in the comparisons, but not necessarily the unique best sequence. In fact, the Evolutionary Solver was able to improve on all three heuristics in 32% of the lognormal instances, but we hasten to add that we used the Evolutionary Solver in a restrictive way. For each test problem, we invoked the Evolutionary Solver just once, with a limit on the time it was allowed to search without finding an improvement, so our results may underestimate the power of the Evolutionary Solver. Nevertheless, its relative performance suggests that the best of the three basic heuristics may produce an optimal solution very frequently.

8 Summary and conclusions

We compared three heuristic methods for solving the stochastic two-machine flow shop problem. These three methods—Johnson's Heuristic, Talwar's Heuristic, and the API Heuristic—are structured around theoretical results in the analysis of flow shops, and each is relatively easy to implement in practice. These heuristic methods require virtually no specialized computer software. Specifically, they do not require computer simulation to estimate a performance metric, nor do they require computationally-intensive search procedures. Nevertheless, they consistently generate good results, almost always within 1% of the best performance we could find with an advanced search procedure. Our experimental evidence also suggests that there may be a diffence of only a few percent in performance between an arbitrary sequence and the best sequence. That is, this is not the type of problem in which we are likely to find ways of improving on arbitrary schedules by 50% or more.

We found that none of the three heuristic procedures dominates the others, and we were able to identify specific conditions under which one or the other performed comparatively well.

- We observed that Johnson's Heuristic is best when little or no overlap exists among the job processing time distributions. This might be the case if processing time distributions have small variances or means that are widely dispersed. When overlap exists, Johnson's Heuristic deteriorates compared to the other heuristics, but its performance is consistently better than that of an arbitrary sequence.

- We observed that Talwar's Heuristic is best when the distributions are all from the exponential family, a theoretical property that has already been noted in the literature. The more general form of this property, empirically speaking, is that the heuristic performs relatively well when the processing time distributions exhibit substantial amounts of overlap.

- We observed that the API Heuristic seems to have robust performance in many situations. It performs nearly as well as Johnson's Heuristic when little or no overlap exists, but it does not deteriorate as much when overlap is present. However, its performance was consistently worse than the other two heuristic methods when variance in the data set was large.

We might wonder about anchoring our quantitative results on the best solution found. Because we have no known optimizing procedure for the stochastic flow shop, we would have to combine enumeration and simulation to estimate the optimum, an approach that is computationally burdensome. One alternative might be to anchor the results on the solution to the deterministic

counterpart, which provides a valid lower bound (Portugal and Trietsch, 2006). However, the makespan of the deterministic counterpart is insensitive to variability and may thus be a misleading benchmark. By including the Evolutionary Solver in our comparisons, we attempted to create a good proxy for the optimal solution.

Our heuristics can all be used in a dispatching mode. Talwar's Heuristic is easiest in this sense, because it prioritizes all jobs in a transitive manner. Johnson's Heuristic can also be implemented as a two-class, transitive rule, after classifying the jobs according to which processing time is larger. To use the API Heuristic dynamically, we can start with an initial static sequence and when a new job arrives, it can be fitted in the earliest stable position among available jobs. Based on the insights from our experiments, we recommend using the Johnson Heuristic unless the coefficient of variation of typical jobs is very high, a case for which Talwar's Heuristic may be superior. We prefer the API Heuristic, however, if it is not clear which of the other two is more appropriate. (That would be the case when we don't know whether typical jobs exhibit high or low variation.)

We thus have three good heuristic rules for dealing with the two-machine flow shop model. One plausible step in building on these results would be to investigate computer-based solution methods. For example, one of the reviewers suggested a simulation-based procedure based directly on Makino's result. Such an approach is computationally more burdensome than the heuristics we studied, but it extends to more complicated cases in which processing times are correlated. Another plausible direction would be to develop algorithms for three-machine and *m*-machine stochastic flow shop problems, building on the insights from our experiments.

References

Baker, K.R., Trietsch, D. (2009). *Principles of Sequencing and Scheduling*. Hoboken: Wiley.

Campbell, H.G., Dudek, R.A., Smith, M.L. (1970). A Heuristic Algorithm for the *n* Job, *m* Machine Sequencing Problem. *Management Science*, 16, 630-637.

Cunningham, A.A., Dutta, S.K. (1973). Scheduling Jobs with Exponentially Distributed Processing Times on Two Machines of a Flow Shop. *Naval Research Logistics Quarterly*, 16, 69-81.

Elmaghraby, S.E., Thoney, K.A. (1999). The Two-Machine Stochastic Flowshop Problem with Arbitrary Processing Time Distributions, *IIE Transactions*; 31, 467-477.

Johnson, S.M. (1954). Optimal Two- and Three-Stage Production Schedules with Setup Times Included. *Naval Research Logistics Quarterly*, 1, 61-68.

Ku, P.-S., Niu, S.-C. (1986). On Johnson's Two-Machine Flow Shop with Random Processing Times. *Operations Research*, 34, 130-136.

Makino, T. (1965). On a Scheduling Problem. *Journal of the Operations Research Society Japan*, 8, 32-44.

Portougal, V., Trietsch, D. (2006). Johnson's Problem with Stochastic Processing Times and Optimal Service Level. *European Journal of Operational Research*, 169, 751-760.

Talwar, P.P. (1967). A Note on Sequencing Problems with Uncertain Job Times. *Journal of the Operations Research Society Japan*, 9, 93-97.