Setting Optimal Due Dates in a Basic Safe Scheduling Model

By

Kenneth R. Baker

Tuck School of Business
Dartmouth College
Hanover, NH 03755

ken.baker@dartmouth.edu

July, 2013

Abstract

We examine a basic stochastic sequencing model with due dates as decisions. The objective is to make due dates as tight as possible while meeting service level constraints. We develop an optimizing procedure using branch and bound, and we evaluate the performance of heuristic procedures as well.

Keywords: safe scheduling, stochastic sequencing, due-date setting, service levels

Setting Optimal Due Dates in a Basic Safe-Scheduling Model

1. Introduction

Safe scheduling identifies a class of stochastic scheduling problems in which service levels play a key role. This perspective augments traditional scheduling models by recognizing the role of safety time. The standard approach to stochastic scheduling focuses on expected values and ignores service levels, whereas safe-scheduling models recognize the role of service levels explicitly (Baker and Trietsch, 2009b). The safe-scheduling perspective gives rise to a set of new and interesting scheduling models and thereby expands both the theoretical and practical dimensions of the field. In this paper, we examine a basic model in safe scheduling and describe how to produce optimal solutions. With that benchmark in hand, we also look at three heuristic approaches and evaluate their effectiveness.

The safe-scheduling model builds on the basic single-machine sequencing problem (see, for example, Baker and Trietsch, 2009a). The key parameters in the model include the processing time for job j (p_j) and the due date (d_j). In the actual schedule, job j completes at time C_j, and the set of completion times leads to a measure of performance to be optimized. Two types of models relate to due dates. In the traditional scenario, due dates are given, and a performance measure (such as total tardiness) captures the effectiveness of the schedule at meeting the given due dates. The model may also permit jobs to be rejected. In the other scenario, due dates are internal decisions, possibly used in negotiations with customers or incorporated into the schedule in order to pace the progress of the various jobs. In this case, all jobs are accommodated, and no job can be rejected. The model described in this paper falls into this latter category, with due dates as decisions.

Our model is also stochastic, in the sense that the processing time p_j is treated as a random sample from a known distribution. Specifically, we assume that the processing time for job j follows a normal distribution with mean μ_j and standard deviation σ_j and that the p_j are independent random variables. We use the normal distribution because it is familiar and plausible for many scheduling applications. Later, we comment on more general forms of the model.

As a consequence of sequencing decisions, job j will have a random completion time C_j, and the corresponding service level SL_j is defined as follows:

$$SL_j = \mathrm{P}(C_j \le d_j)$$

We assume that a required service level b_j is given for job j. In other words, we require that $SL_j \ge b_j$ or

$$\mathrm{P}(C_j \le d_j) \ge b_j$$

When this inequality is satisfied, we say that job j is *stochastically on time*; otherwise, the job is *stochastically tardy*. Thus, we prescribe constraints that require each job to be stochastically on time. To reflect practical service level goals, we assume that b_j is relatively high (say, 0.8 or more),

although the analysis is consistent with service levels of at least 0.5. For the purposes of exposition, we assume equal service levels for all jobs, or $b_j = b$, but it is straightforward to generalize our analysis to unequal service levels.

In essence, we face a sequencing problem in which we select due dates subject to a set of service-level constraints. As an objective function, we strive for efficient due dates—that is, due dates that are as tight as possible. Operationally, we wish to minimize the sum of the due dates, D, where

$$D = \sum_{j=1}^{n} d_j \qquad (1)$$

To state the optimization problem precisely, we define the binary variable x_{jk} as follows

x_{jk} = 1 if job j is sequenced in position k

= 0 otherwise

Then we want to solve the following optimization problem in the variables d_j and x_{jk}:

$$\text{Min} \quad \sum_{j=1}^{n} d_j$$

$$\text{s.t.} \quad \sum_{u=1}^{n} x_{uk} = 1, \quad \text{for all positions } k,\ 1 \le k \le n$$

$$\sum_{u=1}^{n} x_{ju} = 1, \quad \text{for all jobs } j,\ 1 \le j \le n$$

$$P\left(\sum_{u=1}^{k}\sum_{j=1}^{n} p_j x_{ju} - \sum_{j=1}^{n} d_j x_{jk} \le 0\right) \ge b, \quad \text{for all positions } k,\ 1 \le k \le n$$

In this formulation, the "assignment constraints" ensure that the x_{jk} variables correspond to a valid job sequence based on which we can express the service level requirements. We can view this formulation as a nonlinear, integer stochastic program in which the parameters p_j are independent (normal) random variables. This model represents the basic safe-scheduling problem with due dates as decisions, but until now, no method has been given in the literature, short of complete enumeration, that will produce optimal solutions.

2. Literature Review

The safe-scheduling model considered in this paper features due-date assignments and stochastic processing times. We review the literature as it relates to the single-machine model, tracing the highlights of these themes in the subsections that follow. First, we discuss articles on due-date assignment in deterministic models; then we discuss relevant articles on stochastic scheduling.

2.1. Due-Date Assignment

Considerations of assigning due dates to jobs emerged from the early work in job shop scheduling, particularly Conway (1965). Although many articles followed in that tradition, we highlight work that addressed the single-machine model. The basic problem in this segment of the literature is to assign due dates to the jobs and to sequence those jobs so that a performance measure related to due dates is optimized. In the deterministic setting, due dates can be assigned so that they are achievable, and the objective is usually to make them as tight as possible. The optimal due dates for this problem were characterized by Baker and Bertrand (1981), who also examined heuristic rules for assigning due dates, such as those based on constant, slack-based, or total-work leadtimes. Alternatively, Seidmann, et al. (1981) solved a specialized variation of the deterministic single-machine model with earliness and tardiness penalties, based on a cost structure that penalizes loose due dates if they exceed customers' reasonable and expected lead time. The common due-date version of the same problem, studied by Panwalkar, et al. (1982) proved to be more difficult to solve, except when expected lead times are zero. The literature on heuristic rules for due-date assignment was later reviewed by Cheng and Gupta (1989). Surveys of the common due-date assignment problem have been compiled by Baker and Scudder (1990) and Gordon, et al. (2002). As indicated below, however, very little of the work on due-date assignment has dealt with stochastic models.

2.2. Stochastic Processing Times

The stochastic counterpart of a deterministic sequencing problem is typically defined by treating processing times as uncertain and then attempting to minimize the expected value of the deterministic criterion. Occasionally, it is possible to solve the stochastic counterpart by replacing processing time distributions by their means and simply invoking deterministic optimization results. This approach works for the minimization of expected total weighted completion time, which is minimized by sequencing the jobs in order of shortest weighted expected processing time, or SWEPT (Rothkopf, 1966). Not only is SWEPT optimal, but the optimal value of expected total weighted completion time can be computed by substituting expected values for processing times and calculating the total weighted completion time for the resulting deterministic model. For the minimization of expected maximum tardiness it is optimal to sequence the jobs in order of earliest due date, or EDD (Crabill and Maxwell, 1969). However, substituting expected values for processing times and calculating the deterministic objective function under EDD does not necessarily produce the value of the objective in the stochastic counterpart. In fact, suppressing uncertainty seldom leads to the optimal solution of stochastic sequencing problems; problems that are readily solvable in the deterministic case may be quite difficult to solve when it comes to their stochastic counterpart. A case in point is minimizing the expected number of tardy jobs (or, equivalently, the number of stochastically tardy jobs) when due dates are given, a problem first posed by Balut (1973). Kise and Ibaraki (1973) showed that even this relatively basic stochastic problem is NP-Hard, and later Jang (2002) offered a myopic heuristic procedure for the problem and its weighted version. Because of the computational difficulties this problem presents, other formulations have drawn attention. For example, Banerjee (1965) showed that EDD sequencing minimizes the maximum probability of being late. Equivalently, EDD maximizes the minimum service level, but this criterion does not have a direct deterministic counterpart. Elsewhere, several papers have examined the case of a common due date. Sarin, et al. (1991) and Seo, et al. (2005) studied the problem of minimizing the

expected number of tardy jobs when all jobs have the same due date. Pinedo (1983) and De, et al. (1991) studied the common due-date problem when the due date follows an exponential distribution. Other papers that deal with stochastic sequencing in the single machine case all adopt an expected cost criterion, usually using something more tractable than the number of stochastically tardy jobs. In our formulation, the number of stochastically tardy jobs is not the criterion; instead, all jobs are constrained to be stochastically on time, and the criterion is to make the due dates as tight as possible. This new variation of stochastic sequencing has not been studied in the literature.

An alternative direction for stochastic sequencing is represented by techniques of robust scheduling. As originally introduced by Daniels and Kouvelis (1995), robust scheduling aimed at finding the best worst-case schedule for a given criterion. This approach, which is essentially equivalent to maximizing the minimum payoff in decision theory, requires no distribution information and assumes only that possible realizations for each stochastic element can be identified but not their relative likelihoods. On the other hand, β-robust scheduling, due to Daniels and Carrillo (1997), does use distribution information in maximizing the probability that a given level of schedule performance will be achieved. A more recent application is discussed by Wu, et al. (2009). Nevertheless, the criterion to be optimized in these models is a probability; the performance measure itself may not be optimized. By contrast, the model we examine in this paper optimizes a specific criterion (due-date tightness), subject to the requirement that all jobs must be stochastically on time.

In stochastic scheduling it is rare to find an optimal solution that holds for any choice of the processing time distribution. The early results mentioned above are among the few such examples that have been discovered. Some additional results have been derived for exponential distributions, but the exponential distribution—which has just one parameter—seldom serves as a good model for stochastic behavior found in practice. Several papers have addressed stochastic scheduling problems and have used the normal distribution as an appropriate model for processing times. Examples in the literature on single-machine problems include Balut (1973), Sarin, et al. (1991), Fredendall and Soroush (1994), Cai and Zhou (1997), Soroush (1999), Jang (2002), Seo, et al. (1995), Portougal and Trietsch (2006), and Wu, et al. (2009). The model we examine in this paper follows in the same tradition.

3. Analysis of the Model

The assumption of normal processing times leads to a convenient result: in any sequence, the completion time of job *j* follows a normal distribution because the sum of normal random variables is also normally distributed. Using notation, let B_j denote the "before set," or the set of jobs preceding job *j* in the schedule. Then C_j follows a normal distribution with mean $E[C_j] = \sum_{i \in B_j} \mu_i + \mu_j$ and variance $var[C_j] = \sum_{i \in B_j} {\sigma_i}^2 + {\sigma_j}^2$. To streamline the notation, we write $E[C_j] = \mu_{B_j} + \mu_j$ and $var[C_j] = \sigma_{B_j}^2 + \sigma_j^2$. Once we know the properties of the random variable C_j, we can determine the optimal choice of d_j.

To simplify the service level requirement in the normal case, let z represent the standard normal variate at which the cumulative distribution function equals b. In standard notation, $\Phi(z) = b$. Then the appropriate choice for the due date of job j is

$$d_j = \mathrm{E}[C_j] + z(\mathrm{var}[C_j])^{1/2} = \mu_{B_j} + \mu_j + z\,(\sigma_{B_j}^2 + \sigma_j^2)^{1/2} \qquad (2)$$

In this expression the due date d_j depends on the previous jobs in sequence via the set B_j, and our objective is the sum in (1).

Finding a job sequence to minimize the sum of due dates in (1) is a challenging sequencing problem. It is possible, of course, to find an optimum by enumerating all possible job sequences and then selecting the sequence with minimal sum of due dates. We refer to this as Algorithm E. (See the appendix for details.)

However, enumerative methods are ultimately limited by computing considerations and the size of the solution space. We can compute optimal solutions more efficiently with the use of a branch and bound (B&B) algorithm, and we can enhance that algorithm by exploiting a basic dominance condition. Next, we explore these two aspects of the model.

3.1 Lower Bound

Suppose we have a partial sequence of the jobs, denoted by π, and we wish to compute a lower bound on the value of D that can be obtained by completing the sequence. The component of D generated by the jobs in π has presumably been computed already. Let π' denote the set of unscheduled jobs. In the set π', we take the set of means μ_j in smallest-first order and take the set of standard deviations σ_j in smallest-first order and treat these values as if they were paired. Then we calculate the component of D generated by these fictitious unscheduled jobs and add it to the component for the partial sequence. This total is a lower bound on the value that could be achieved by completing the partial sequence in the best possible way. (A formal proof follows a standard pairwise interchange argument.)

To express this lower bound calculation mathematically, let p denote the number of scheduled jobs, i.e., the number of jobs in π. For the remaining $(n - p)$ jobs, we sort their μ_j values and use bracketed subscripts to denote their order in nondecreasing sequence. Thus $\mu_{[1]}$ represents the smallest value among the means of unscheduled jobs, $\mu_{[2]}$ represents the next smallest, and so on, up to $\mu_{[n-p]}$. Separately, sort the σ_j values and let $\sigma_{[1]}$ represent the smallest value among unscheduled jobs, $\sigma_{[2]}$ represent the next smallest, and so on, up to $\sigma_{[n-p]}$. Then, following the logic of (2), we can compute the due date for job $[j]$ in the lower bound as follows, where $1 \le j \le (n - p)$, using the formula

$$d_{[j]} = \mu_\pi + \sum_{i=1}^{j} \mu_{[i]} + z\,(\sigma_\pi^2 + \sum_{i=1}^{j} \sigma_{[j]}^2)^{1/2}$$

Thus, in a B&B algorithm, if we ever encounter a partial sequence for which the lower bound on D is greater than or equal to the value of D for a known sequence, we know that the partial sequence can never lead to a full sequence with a better value than the known sequence. This is the lower-

bounding principle that enables us to curtail an enumerative search. The resulting procedure is called Algorithm B.

3.2 Dominance

Job *j* is said to dominate job *k* if an optimal sequence exists in which job *j* precedes job *k*. If we confirm a dominance condition of this type, then in searching for an optimal sequence we need not pursue any partial sequence in which job *k* precedes job *j*. Dominance conditions can reduce the search effort required to find an optimal schedule, but the extent to which dominance conditions apply depends on the specific data in a given problem instance. For that reason, it may be difficult to predict the extent of the improvement.

In our model, a relatively simple dominance condition holds.

Property 1. For two jobs *j* and *k*, if $\mu_j \leq \mu_k$ and $\sigma_j \leq \sigma_k$ then job *j* dominates job *k*. In other words, there exists an optimal sequence in which job *j* precedes job *k*.

Property 1 is straightforward to prove by means of an adjacent pairwise interchange argument. Thus, if we are augmenting a partial sequence and we notice that job *j* dominates job *k* while neither appears in the partial sequence, then we need not consider the augmented partial sequence constructed by appending job *k* next. Although we may encounter quite a few dominance properties in randomly-generated instances, it is also possible that no dominance conditions hold. That would be the case if the means and standard deviations were ordered in opposite directions. In such a situation, dominance properties would not help reduce the computational burden, and the computational effort involved in testing dominance conditions would be counterproductive. We refer to the algorithm that checks dominance conditions as Algorithm D.

Finally, we can implement both lower bounds and dominance conditions in the search. We refer to this approach as Algorithm BD. This combined algorithm involves the most testing of conditions but can thereby eliminate the most partial sequences among the various algorithms. In the next section we describe computational results for four versions of a solution procedure: Algorithm E, Algorithm B, Algorithm D, and Algorithm BD. (See the appendix for details on the four algorithmic variations.)

4. Computational Results for Optimization Methods

For experimental purposes, we generated a set of test problems for various values of *n*, the number of jobs. The mean processing times were sampled from a uniform distribution between 10 and 100. Then, for each job *j*, the standard deviation was sampled from a uniform distribution between $0.10\mu_j$ and $0.25\mu_j$. In other words, the mean was between 4 and 10 times the standard deviation, virtually ensuring that the individual processing times would be positive. For each value of *n*, we created 100 randomly-generated problem instances. In addition, we set the required service level at 95% (corresponding to *z* =

1.645). The computational effort is mildly sensitive to the choice of service level, but that turns out to be a second-order effect.

For practical considerations, the algorithms were coded in VBA to maintain an easy interface with Excel and run on a laptop computer using a 2.9 GHz processor. For Algorithm E, the search ultimately generated all *n*! feasible sequences, building up from partial sequences. If we think of each full or partial sequence as a node in the search tree, the number of nodes can be expressed by the formula $\sum_{k=0}^{n} k! \binom{n}{k}$. The code solved problem sizes up to *n* = 12 in roughly 45 minutes of cpu time. Larger problems could not be solved within an hour using Algorithm E.

Table 1 summarizes the computational experience for the four algorithms on relatively small problem sizes. All run times in this table (given in seconds) are quite small, but the node count provides a useful perspective. Each figure represents the average over 100 instances with the same parametric features. Algorithm E examines all full and partial sequences using a simple tree-search procedure, so even with a large number of nodes, its run times were relatively small. Each of the other algorithms invests in some additional computational overhead to eliminate nodes in the search tree. Algorithm BD, which combines the elimination conditions of Algorithms B and D, obviously achieved the smallest number of nodes, but its added computational overhead sometimes offset the improvement in node count. In larger problems, however, we might expect that trading off additional overhead for a reduction in node count will be productive.

	Average CPU time (seconds)				**Average Number of Nodes Generated**			
n	***Algo E***	***Algo B***	***Algo D***	***Algo BD***	***Algo E***	***Algo B***	***Algo D***	***Algo BD***
5	0.0002	0.0006	0.0002	0.0002	326	66	20	18
6	0.0004	0.0011	0.0004	0.0004	1955	147	36	27
7	0.0005	0.0027	0.0005	0.0005	13700	301	58	40
8	0.0006	0.0047	0.0005	0.0009	109601	591	88	55
9	0.0010	0.0072	0.0008	0.0011	986410	977	154	76
10	0.0012	0.0102	0.0011	0.0015	9864101	1955	267	114

Table 1. Computation effort for small problem sizes.

Next, we tested Algorithms B, D, and BD on larger problems. Our primary goal was to compare the algorithms. In particular, we wanted to discover how large a problem could be solved within an hour of computation time, a benchmark that has been used frequently in computational comparisons to determine the practical capabilities of an algorithm. A summary of our results appears in Table 2, where all times are shown in seconds.

	Average time			***Maximum time***		
n	***Algo B***	***Algo D***	***Algo BD***	***Algo B***	***Algo D***	***Algo BD***

15	0.21	0.03	0.005	1.1	0.49	0.05
20	8.95	1.1	0.05	51.14	11.14	0.42
25	784.76	157.18	0.88	(2)	3175.05	16.11
30			14.48			455.31
32			99.20			2332.32
35			662.61			(4)

Table 2. Computation effort for larger problem sizes.

Table 2 summarizes results for problem sizes of $15 \le n \le 35$. In this set of runs, the main entries represent average cpu time and the maximum cpu time observed in the 100 test problems. The results in parentheses show the number of test problems that were unsolved after an hour of cpu time. However, all instances were solved to optimality, and the run times that exceeded one hour were included in the average value shown on the left-hand side of the table. The table reveals that each of the algorithms eventually reaches a computational "knee of the curve" in terms of problem size. Algorithm B solved 25-job problems in an average of about 13 minutes, with a maximum that in two cases exceeded one hour. Algorithm D was faster, with solution times that averaged less than three minutes. Algorithm BD was much more efficient, as we anticipated, but it eventually encountered combinatorial demands as the problem size grew to 35 jobs. At a problem size of n = 35, Algorithm BD was able to solve 96 out of the 100 test problems in less than an hour, averaging about 11 minutes. Nevertheless, for problems of moderate size, such as these, Algorithm BD is the best performer among optimizing algorithms.

5. Computational Results for Heuristic Methods

For problems containing a large number of jobs (in this case, 35 or more), optimization procedures may require prohibitive amounts of time to find a solution. For that reason, we are also interested in evaluating the effectiveness of heuristic procedures that do not require combinatorial amounts of effort. Here we study some simple yet powerful heuristics for the problem.

A simple and straightforward heuristic procedure is to sort the jobs. In this problem, a reasonable approach is to sort on the basis of shortest expected processing time (SEPT), or in other words, nondecreasing values of μ_j, breaking ties according to smallest σ_j. After the jobs are ordered, the optimal due date for each job can be computed using equation (2). This heuristic is easy to implement and has the appealing property that it generates optimal solutions in the deterministic counterpart. (Furthermore, when means and variances are agreeable in the stochastic case, SEPT produces an optimal solution.) SEPT is a static rule: the relative desirability of one job over another applies globally.

We might observe, however, that SEPT uses information about means but ignores information about standard deviations, except for breaking ties. An alternative sorting mechanism

would be one that takes both means and standard deviations into account. A simple way to do so is to sort on the sum ($\mu_j + \sigma_j$). We use the acronym SMSD (sum of mean and standard deviation) for this heuristic. SMSD is also a static rule because the jobs need to be sorted only once, and that can be done before the schedule begins.

A slightly more complex heuristic procedure is to order the jobs by earliest due date (EDD), referring to the optimal due date for the job. However, the optimal due date for job *j* depends on when job *j* appears in the schedule—that is, it depends on which jobs appear earlier. Suppose a partial schedule π has already been implemented and we want to determine which job to schedule next. For each unscheduled job *j*, we calculate its due date from (2) assuming it comes next; that is:

$$d_j = \mu_\pi + \mu_j + z\,(\sigma_\pi{}^2 + \sigma_j{}^2)^{1/2}$$

Under the EDD rule, we then choose the job with smallest d_j as the next job in the schedule. To begin the schedule, we take $\mu_\pi = \sigma_\pi{}^2 = 0$. This version of the EDD rule (a greedy algorithm) is a dynamic rule because the preference between job *j* and job *k* may depend on π. In other words, the preference may depend on when the pair of jobs is considered for scheduling.

Following Baker and Trietsch (2009b), it is possible to show that the EDD rule is asymptotically optimal—that is, as *n* increases and μ_π and σ_π grow large, the EDD rule produces a solution that is negligibly different from the optimum. Therefore, we might expect that EDD performs quite well as the problem size grows. In fact, as discussed below, EDD seems to produce optimal solutions nearly all the time, even for problem sizes as small as *n* = 3.

Table 3 contains a three-job example to demonstrate that EDD is not necessarily optimal. In this example, we take $b_j = 0.8413$, so that $z = 1$.

Job *j*	1	2	3
Mean μ_j	11.0	10.1	10.0
s.d. σ_j	0.05	1.00	1.20

Table 3. A three-job instance in which EDD is suboptimal.

At time zero, we compare $\mu_j + z\sigma_j$ for the three jobs. The minimum occurs for job 1. At its mean completion time (11), we select the next job according to $\mu_j + z(\sigma_1^2 + \sigma_j^2)^{1/2}$ for $j \neq 1$. The minimum occurs for job 2. Finally, job 3 comes last, and the full sequence is 1-2-3. The sum of due dates for that sequence is 65.81, whereas the optimal sequence (2-3-1) achieves a value of 65.42.

The three heuristic procedures, SEPT, SMSD, and EDD, were tested on the same instances used in Table 2. The quality of the heuristic solutions is summarized in Table 4. Two summary measures are of interest: (1) the number of times the heuristic produced an optimal solution (out of

100 instances) and (2) the extent of suboptimality, as measured by the maximum value (expressed as a percent above optimal) observed in the 100 instances. As the table shows, both static rules tended to produce fewer optima as the problem size increased, and neither was able to produce an optimal solution to any of the 35-job problems. Meanwhile, the EDD procedure achieved optimality in all but two of the 600 test problems. Although we know that the EDD rule cannot guarantee an optimal solution, the frequency with which it generates optimal solutions in the test data is remarkable in comparison to most other scheduling heuristics.

	Optima			***Maximum***		
n	***SEPT***	***SMSD***	***EDD***	***SEPT***	***SMSD***	***EDD***
15	14	69	100	0.07%	<0.01%	0.00%
20	4	35	99	0.04%	0.01%	<0.01%
25	6	13	100	0.04%	0.01%	0.00%
30	3	2	100	0.02%	0.03%	0.00%
32	1	3	99	0.09%	0.07%	<0.01%
35	0	0	100	0.07%	0.07%	0.00%

Table 4. Summary results for the heuristic procedures.

Although the EDD rule produced an optimal solution in almost every one of our test problems, its ability to produce an optimal solution may depend on features of the problem data. To illustrate this point, an additional set of test problems was created in which no pair of jobs satisfied the dominance condition. In other words, when one job had a smaller mean than another, it also had a larger standard deviation. (Under these conditions, Property 1 never holds, so no dominance exists.) In this set, the EDD rule produced an optimal solution only slightly less often than in the previous dataset, as shown in Table 5. As in Table 4, the dataset consists of 100 instances for each problem size.

	Optima			***Maximum***		
n	***SEPT***	***SMSD***	***EDD***	***SEPT***	***SMSD***	***EDD***
3	66	90	100	0.64%	0.06%	0.00%
6	53	66	100	0.22%	0.04%	0.00%
10	39	41	95	0.14%	0.05%	<0.01%
15	21	16	92	0.07%	0.05%	<0.01%
20	20	10	95	0.05%	0.04%	<0.01%

Table 5. Summary results for the heuristic procedures on test problems without dominance.

6. Summary and Conclusions

We have analyzed the basic safe-scheduling problem with due dates as decisions. In this problem, we seek to assign due dates and sequence the jobs so that the sum of the assigned due dates is as small as possible, given the service levels to be met for each job, when processing times follow normal distributions. We described an optimization approach for this problem that is capable of systematically generating all partial and full sequences, but then we incorporated lower bounds and a dominance condition to reduce the search effort. Our computational experiments indicated that the resulting algorithm (Algorithm BD) can solve nearly all problems of up to 35 jobs in less than an hour of cpu time.

We also examined three heuristic solution procedures. Somewhat surprisingly, the dynamic version of the EDD rule delivered optimal solutions in the vast majority of the test cases we examined, although it can sometimes be suboptimal. Even static rules reliably generate solutions well within 1% of optimality, but the minimal extra effort required to implement EDD seems well worth the small computational price it requires by comparison. (The average run times for the EDD rule were less than 0.001 second.)

The normal distribution has been used in previous scheduling research, as mentioned earlier, but we might wonder whether our analysis can somehow be extended to other distributions. The key to answering this question is the observation that completion times are described by convolutions of the processing time distribution. The normal distribution is particularly convenient because sums of normal distributions (that is, completion time distributions) are normally distributed. In place of the normal distribution, we could have assumed that processing times follow lognormal distributions. The lognormal distribution is sometimes offered as a more practical representation of uncertain processing times; indeed, it may be the most useful standard distribution model for that purpose. In addition, the lognormal supports its own central limit theorem (Trietsch, et al., 2012). In other words, sums of lognormal random variables are not lognormal, but they approach a lognormal distribution. Therefore, the type of analysis in (2) could be adapted to work—at least as an approximation—for the lognormal as well. We have used the normal distribution in this research because no approximations were needed and the distribution itself is familiar to a wide audience.

Safe-scheduling comprises a set of models that represent variations on the deterministic sequencing problems that have traditionally drawn most of the attention in the research literature. Now that stochastic scheduling itself is attracting more attention, safe-scheduling models may serve as useful building blocks for a more solid understanding of the scheduling problem in nondeterministic settings. The analysis in this paper is intended to further this line of research and expand our knowledge about stochastic scheduling.

References

Baker, K. and G. Scudder (1990) Sequencing with earliness and tardiness penalties: a review. *Operations Research* 38, 22-36.

Baker, K. and J. Bertrand (1991) A comparison of due date selection rules. *AIIE Transactions* 13, 123-131.

Baker, K. and D. Trietsch (2009a) *Principles of Sequencing and Scheduling*, John Wiley & Sons, New York.

Baker, K. and D. Trietsch (2009b) Safe scheduling: setting due dates in single-machine problems. *European Journal of Operational Research* 196, 69-77.

Banerjee, B. (1965) Single facility sequencing with random execution times. *Operations Research* 13, 358-364.

Balut, S. (1973) Scheduling to minimize the number of late jobs when set-up and processing times are uncertain. *Management Science* 19, 1283–1288.

Cai, X. and S. Zhou (2007) Scheduling stochastic jobs with asymmetric earliness and tardiness penalties. *Naval Research Logistics,* 44, 531-557.

Cheng, T. and M. Gupta (1989) Survey of scheduling research involving due date determination decisions. *European Journal of Operational Research* 38, 156–166.

Conway, R. (1965) Priority dispatching and job lateness in a job shop. *Journal of Industrial Engineering* 16, 228-237.

Crabill, T. and W. Maxwell (1969) Single machine sequencing with random processing times and random due-dates. *Naval Research Logistics Quarterly* 16, 549-555.

Daniels, R. and P. Kouvelis (1995) Robust scheduling to hedge against processing time uncertainty in single-stage production. *Management Science* 41, 363-376.

Daniels, R. and J. Carrillo (1997) β-Robust scheduling for single-machine systems with uncertain processing times. *IIE Transactions* 29, 977-985.

De, P., J. Ghosh and G. Wells (1991). On the minimization of the weighted number of tardy jobs with random processing times and deadline. *Computers & Operations Research* 18, 457-463.

Fredendall, L. and H. Soroush (1994). The stochastic single machine scheduling problem with earliness and tardiness costs. *European Journal of Operational Research* 77, 287-302.

Gordon, V., J-M. Proth and C. Chu (2002) A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research* 139, 1–25.

Jang, W. (2002) Dynamic scheduling of stochastic jobs on a single machine. *European Journal of Operational Research* 138, 518–530.

Kise, H. and T. Ibaraki (1983) On Balut's algorithm and NP-completeness for a chance constrained scheduling problem. *Management Science* 29, 384-388.

Panwalkar, S., M. Smith and A. Seidmann (1982) Common due date assignment to minimize total penalty for the one machine scheduling problem. *Operations Research* 30, 391–399.

Pinedo, M. (1983) Stochastic scheduling with release dates and due dates. *Operations Research* 31, 559–572.

Portougal, V. and D. Trietsch (2006) Setting due dates in a stochastic single machine environment. *Computers & Operations Research* 33, 1681-1694.

Rothkopf, M. (1966). Scheduling with random service times. *Management Science* 12, 707-713.

Sarin, S. E. Erdel and G. Steiner (1991) Sequencing jobs on a single machine with a common due date and stochastic processing times. *European Journal of Operational Research* 27, 188–198.

Seidmann, A., S. Panwalkar and M. Smith (1981) Optimal assignment of due-dates for a single processor scheduling problem. I*nternational Journal of Production Research* 19,393–399.

Seo, D., C Klein, and W. Jang (2005) Single machine stochastic scheduling to minimize the expected number of tardy jobs using mathematical programming models. *Computers & Industrial Engineering* 48, 153-161.

Soroush, H. (1999) Sequencing and due-date determination in the stochastic single machine problem with earliness and tardiness costs. *European Journal of Operations Research* 113, 450–468.

Trietsch, D., L. Mazmanyan, V. Ohanyan and K. Baker (2012) Modeling activity times by the Parkinson distribution with a lognormal core: Theory and validation. *European Journal of Operational Research* 216, 386–396.

Wu, C., K. Brown, and J. Beck (2009) Scheduling with uncertain durations: Modeling β-Robust scheduling with constraints. *Computers & Operations Research* 36, 2348–2356.

Appendix. Algorithm Details

Algorithm E is a backtracking algorithm. For example, if there are three jobs in the dataset, the algorithm creates the following partial and full sequences, in order: 1, 12, **123**, 13, **132**, 2, 21, **213**, 23, **231**, 3, 31, **312**, 32, **321**. (Complete sequences are shown in bold.) From any current node representing a partial sequence, the algorithm branches by appending the lowest-indexed job that has not previously been appended to this partial sequence.

Algorithm B is a branch and bound algorithm carried out on the branching tree described in Algorithm E. For any partial sequence that is not a complete sequence, its lower bound is calculated and compared to the value of the best complete sequence found thus far. If the bound is greater than or equal to that value, the partial sequence is fathomed—that is, no further branching from this partial sequence is needed in the search for an optimum.

Algorithm D is a variation of Algorithm E that eliminates branches in which appending the next job would violate the dominance condition. In the three-job example illustrated earlier, suppose job 3 dominates job 1. Then, when branching is carried out from the node corresponding to the partial sequence 2, the partial sequence 21 is eliminated (not constructed) because the completion of that partial sequence would schedule job 1 before job 3, violating the dominance condition.

Algorithm BD adds both dominance conditions and lower bound calculations to Algorithm E. It eliminates branches in which appending the next job would violate the dominance condition. Among the partial sequences that remain, it calculates the lower bound and compares it to the value of the best complete sequence found thus far to determine whether the partial sequence can be fathomed. This combination constructs fewer partial sequences than the other algorithms.