GENERATING EXPERIMENTAL DATA FOR COMPUTATIONAL TESTING WITH MACHINE SCHEDULING APPLICATIONS

NICHOLAS G. HALL

Fisher College of Business, Department of Management Sciences,
The Ohio State University, Columbus, Ohio 43210-1399, hall.33@osu.edu

MARC E. POSNER

Department of Industrial and Systems Engineering,
The Ohio State University, Columbus, Ohio 43210-1271, posner.1@osu.edu

(Received February 1997; revisions received October 1998, April 2000; accepted June 2000)

The operations research literature provides little guidance about how data should be generated for the computational testing of algorithms or heuristic procedures. We discuss several widely used data generation schemes, and demonstrate that they may introduce biases into computational results. Moreover, such schemes are often not representative of the way data arises in practical situations. We address these deficiencies by describing several principles for data generation and several properties that are desirable in a generation scheme. This enables us to provide specific proposals for the generation of a variety of machine scheduling problems. We present a generation scheme for precedence constraints that achieves a target density which is uniform in the precedence constraint graph. We also present a generation scheme that explicitly considers the correlation of routings in a job shop. We identify several related issues that may influence the design of a data generation scheme. Finally, two case studies illustrate, for specific scheduling problems, how our proposals can be implemented to design a data generation scheme.

The operations research literature includes many contributions that address applications. Much of this work involves the development of solution procedures for newly defined problems or improved procedures for known problems. These procedures may be algorithms that identify optimal solutions, or heuristics that seek approximate solutions. In either case, it is valuable and customary to include computational evaluations of the proposed procedures. The purposes of these evaluations include:

1. Demonstrating the potential of a new procedure in specific situations,
2. Demonstrating that a procedure is practical,
3. Identifying conditions under which a procedure performs well or poorly, and
4. Comparing competing procedures.

Recently, widespread concern has arisen that many published computational experiments are inadequate. Consequently, editors and researchers familiar with computational testing recommend that authors adhere to more rigorous and consistent guidelines when *reporting* computational experiments (Crowder et al. 1978, Florian et al. 1979, Jackson et al. 1991, and Lee et al. 1993). Researchers also suggest methodologies for *designing* computational experiments. Greenberg (1990) discusses general approaches to computational testing and offers suggestions about how much testing should be performed. Several authors discuss what should be measured to determine the performance of a solution procedure. These include Crowder and Saunders (1980), Hoffman and Jackson (1982), Greenberg (1990), and Barr et al. (1995). Ahuja et al. (1993) describe a method to determine the approximate running time of a procedure.

Hooker (1994) makes a persuasive case for more rigorous computational evaluation of algorithms. He suggests that problems be generated in such a way as to demonstrate how algorithmic performance depends on problem characteristics. Hooker (1995) discusses how this approach can alleviate some of the difficulties that arise from a traditional "competitive testing" approach. Both of these papers concentrate on high-level experimental design.

Barr et al. (1995) suggest that algorithms and heuristics should be tested against the best competitive solution procedures. However, this approach may tend to focus attention on data sets where it is easiest to demonstrate improvement. Typically, these are sets for which previous procedures have performed badly. This tendency may even influence the design of new procedures. A variety of problem factors, algorithmic factors, and test environment factors in experimental design are discussed. McGeoch (1996) points out that research often centers around intractable problems, where it is difficult to interpret the quality of solution. Generating instances where the solution is known by construction, but concealed from the solution procedure being tested, may generate unrepresentative instances.

Subject classifications: Simulation, random variable generation: methods for generating random data. Production/scheduling: experimental data for testing algorithms and heuristics.
Area of review: OPTIMIZATION.

The above papers provide an excellent discussion of general issues. However, detailed recommendations for data generation are not given. Usually, the primary goal of a computational experiment is to evaluate procedures. For a procedure that finds an optimal solution, we usually want to evaluate the computational effort needed to do so. For a heuristic procedure, we often want to evaluate how close the solution value is to the optimal value. The same is true for a procedure that finds an infeasible bound.

One method for evaluating a solution procedure is mathematical analysis. The order of computational effort of an optimal procedure describes how solution time grows with the size of the problem instance. However, this measure describes only the worst-case running time, and not the typical performance. For example, the simplex method for linear programming has exponential worst-case running time (Klee and Minty 1972). However, its average running time is polynomial. Also, potentially large constant factors are ignored. Hence, it is not clear whether a problem instance of a particular size is computationally tractable.

Another analytical approach to computational evaluation of heuristics involves deriving bounds on the solution value. However, worst-case performance bounds are rarely attained in practice (Fisher 1980). Probabilistic approaches can be used to describe average case performance. However, these results usually require restrictive assumptions about the way in which the data is generated. An example is the frequent use of independent marginal uniform distributions. Thus, the results derived are often not representative of average performance on other randomly generated or real-world problems.

A second method for computational evaluation involves experiments on real-world or library test problems. Real-world problems can provide an accurate assessment of the eventual practical usefulness of a procedure. Library test problems are useful because the properties of the problems and the performance of other procedures are usually known. However, it is frequently difficult to obtain a sufficient number and variety of either type of problem to assess the performance of a solution procedure (McGeoch 1996). Also, McGeoch notes that data sets may quickly become obsolete. Further, comparisons using library problems may promote methodologies that work well on that particular set of instances. This may favor procedures that work poorly for general problems and disfavor other procedures that have superior performance (Hooker 1995). As we later discuss, there is frequently a need to have sets of problems with specific attributes that are based on specific principles. These types of problems are usually absent from library data sets.

In view of the limitations of the above two methods, most research studies use *randomly generated problem instances* for the testing of solution procedures. By computer simulation, many large and varied data sets can be generated quickly and easily. However, there are no systematic guidelines to generate these problems or to conduct these evaluations. For example, competing procedures are often tested by different authors using different problem generation methods on the same problem class. This leads to a lack of comparability of the results obtained. Moreover, some widely used problem generation approaches are inherently biased. For example, they may produce problem instances with particular characteristics, such as having very few active constraints or being easy to solve.

While many of our suggestions are applicable to most classes of deterministic optimization problems, we address these concerns in the context of machine scheduling problems and offer suggestions for improvement. We develop specific proposals for the generation of several types of scheduling data. Because a generation scheme should depend on the problem being considered and the solution procedures being tested, no single scheme should be used for all situations. Consequently, we identify the relevant issues in test problem generation that arise when studying particular solution procedures for particular classes of scheduling problems. Our proposals are illustrated by the development of detailed test problem generation schemes for two scheduling problems.

This paper is organized as follows: §1 describes several principles of data generation that are independent of the specifics of the experiment, and some desirable properties of a generation scheme. Our notation is presented in §2. In §3, we describe and critique existing schemes for generating data for machine scheduling problems. Frameworks for generating various types of scheduling data are given in §4. Other issues that are relevant to data generation are discussed in §5. §6 contains two case studies that illustrate how to design a data generation scheme for specific scheduling problems. §7 contains a conclusion and some suggestions for future research.

1. PRINCIPLES AND PROPERTIES OF DATA GENERATION

A data generation *principle* is an axiom or perspective from which to approach the design of a generation scheme. The principles we describe are general in that they do not depend on the goals or characteristics of a particular experiment. Some principles to consider when generating test problems are:

1. Purpose: Generate data to satisfy the purposes of the experiment.
2. Comparability: Make computational tests comparable.
3. Unbiasedness: Avoid introducing unintended biases into the data.
4. Reproducibility: Make the generation scheme reproducible.

Some purposes for experimentation are given in the introduction. Currently, most experimental studies do not explicitly consider the purpose of the experiment. There is an infinite variety of possible ways to generate data. Consequently, it is impossible to test even a small representative sample of these. Choices should be made based on the purposes of testing and not on previous approaches. A

wide variety of data sets is needed if the goal is to demonstrate the potential of a procedure. However, much more restricted sets should be used if the objective is to show practicality in specific situations.

While Principle 2 is usually taken for granted, there are numerous studies in the literature where different experimental tests in a paper use data sets with different characteristics. Consequently, comparisons between the outcomes of the tests are not valid.

Regarding Principle 3, we have observed two major sources of unintended biases in data generation procedures. The first is when a generation scheme tries to construct feasible solutions. The process of trying to ensure feasibility can distort the data in peculiar ways. The second is when correlation is built into the generation process.

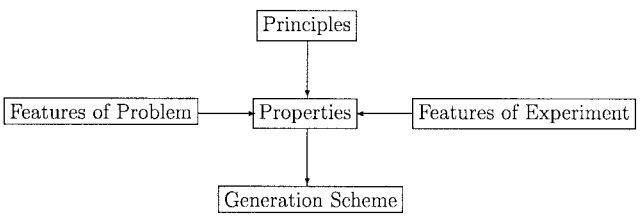
Principle 4 has been widely acknowledged as important. However, the degree of reproducibility falls on a continuum. By creating schemes that can generate a wide variety of data, our methodology provides ways to improve reproducibility.

When constructing a data generation scheme, we should consider the *features of the problem* and the *features of the experiment*. The features of the problem are known problem information. These features include application type, computational complexity, job arrival process, due date setting process, and feasibility considerations. The features of the experiment are decisions that are made about the experiment. They include purposes, publishing intent, implementation decisions, and study size.

The principles and the features of the problem and experiment are inputs to creating the characteristics of the data called *properties* of the experiment. However, depending upon the features of the problem and the experiment, only a subset of the properties may be applicable. A schematic representation of the various components that go into a data generation scheme appears in Figure 1. Properties that are important for the computational evaluation of a solution procedure include:

1. Variety: Creates a wide range of problem instances.
2. Practical relevance: Generates data that models real-world scenarios.
3. Scale invariance: When the scale of the input data changes, other features do not.
4. Size invariance: When the input size changes, other features do not.

Figure 1. Relationships between concepts in a generation scheme.



5. Regularity: Identical types of input are treated in a similar manner.
6. Describability: Easy to describe.
7. Efficiency: Easy and efficient to implement, use, and replicate.
8. Parsimony: Only parameters that may affect the analysis are varied.

Table 1 shows which principles typically lead to which properties. An "X" denotes that a specific principle induces a specific property if the features of the problem and experiment make it applicable.

If the goal is to test the usefulness of a given procedure, then the test data should have variety (Property 1). It is important to test a wide range of problems to discover the strengths and weaknesses of the procedure. For example, consider the problem of minimizing makespan on a single machine in the presence of sequence-dependent setup times. The differences between the values of relaxations of various subproblems are larger when there are large differences in the setup times. If a branch-and-bound approach is used, then the pruning of the tree becomes more efficient. This makes the instance easier to solve. Thus, in accordance with Principle 1, there should be instances with both large and small differences in setup times. Similarly, for comparison, there should be some data sets with large differences and other data sets with small differences. As a second example, Fleischer and Jacobson (1999) provide computational evidence that higher entropy measures are associated with superior finite-time performance of a simulated annealing algorithm.

When several procedures are being compared, testing a wide variety of problems is also desirable to satisfy Principle 2. Any given procedure performs well on some data sets and poorly on others. When an experiment concludes that Procedure A is superior to Procedure B, what is really meant is that Procedure A performs better than B *on average over the data sets generated*. Consequently, as opposed to including only random data sets, an experiment should usually include data sets for which the procedure being tested is likely to perform well and also data sets for which it is likely to perform poorly. For example, as discussed by Potts and Van Wassenhove (1988), there are two competitive procedures for the problem of minimizing the weighted number of late jobs scheduled on a single machine. The first is a dynamic programming algorithm that requires worst-case time of $O(n^2c)$, where c is a parameter measuring average processing time. The second is branch-and-bound, which requires a decision tree of size $O(2^n)$ in the worst case. These two procedures are not comparable over all data sets, because the parameters of the problem instance affect their worst-case running times in different ways. If we increase the average processing time, the effect on the dynamic programming algorithm is proportional, while the effect on the branch-and-bound procedure is negligible. Alternatively, if we increase the number of jobs from n to $n+1$, then the running time of the dynamic programming algorithm increases by a factor

Table 1. Relationships between principles and properties

	Principle			
Property	Purpose	Comparability	Unbiasedness	Reproducibility
Variety	X	X		
Practical Relevance	X			
Scale Invariance		X	X	
Size Invariance		X	X	
Regularity			X	
Describability				X
Efficiency				X
Parsimony				X

of $[(n+1)/n]^2$ in the worst case. However, in a similar situation, the running time of the branch-and-bound algorithm may double in the worst case. If a purpose of the experiment is to select the best procedure for real-world problems, then we can compare the two methods over these types of instances. No experiment can determine which method is superior overall, since there are an infinite number of data sets where each performs better. However, one useful experiment is to determine for which sets each procedure performs better. Then, we can increase the expected processing time to find the point after which the branch-and-bound procedure dominates, and we can increase n to find the point after which the dynamic programming procedure dominates.

Providing a variety of data sets for a particular procedure is easier to achieve when some intuition is available about performance. For example, if a method includes a dominance test for certain variables, then generating data sets that fail these tests creates harder problems, while data sets which satisfy many of the dominance criteria are easier.

If we want results on random problems to predict performance on real-world problems, then the generation scheme should have practical relevance (Property 2). Consequently, we should consider how data arises in real-world settings. Some examples in machine scheduling include: weights that depend on job class, job processing time positively correlated with weight, costs that are grouped by crew seniority in a crew scheduling problem, externally arriving jobs that motivate exponentially distributed interarrival times, interarrival times of internally arriving jobs that have less variance, burstiness of job arrivals, deadlines specified by customer requirements, deadlines determined by the producer based upon processing time, due dates based on an inventory turnover target or on the current jobs in the shop, and due dates specified before job arrival. Similar lists can be developed for other applications.

Usually, the data should be generated so that its features, other than data scale and problem size, do not change with the data scale or problem size, respectively (Properties 3 and 4). In accordance with Principle 2, this enables comparison between experiments with different numbers of variables and different scales. In machine scheduling, for example, as the number of jobs increases, the expected value of total slack (defined in §2) increases, even when the tightness of the constraints remains constant. This makes expected value of total slack less suitable as a problem feature. There are exceptions to Property 3 for certain types of procedures. For example, some dynamic programming algorithms have solution times that depend on the scale of the data. In such situations, it is important to vary the data so that the scale changes.

How a generation scheme should model the growth of problem difficulty as problem size increases is an important issue. If a problem is known to be *NP*-hard, then there does not exist a polynomial time algorithm for that problem, unless $P = NP$ (Garey and Johnson 1979). However, this classification often results from the construction of worst-case instances of the problem which are not representative of average case instances (Lu and Posner 1993). Consequently, the *average* computation time may be a polynomial function of the input size. A similar result has been found empirically for the traveling salesman problem (Lawler et al. 1985, Ch. 10).

A related question is "What constitutes an average case instance?" Although worst-case performance is better defined, average-case performance is more relevant in many computational experiments. For many procedures, it is possible to provide both a data generation scheme representative of worst-case computational effort and an alternative scheme with a slower growth of effort. The first approach is more applicable for computational experiments on extreme performance, while the latter scheme is more applicable for measuring typical performance. Which approach should be used in a particular case depends on the features of the experiment.

In a generation method, certain items should not be given systematically special treatment unless this is required to test a particular hypothesis (Property 5). For example, in a scheduling problem we would not usually want the last due date to depend upon other problem data while other due dates do not. Generating data in this manner can introduce unintended biases, which violates Principle 3.

In accordance with Principle 4, for the purposes of dissemination and documentation, the generation method should be easy to describe and implement (Properties 6 and 7). Regarding parsimony (Property 8), the generation procedure should only create changes in the data that may have an influence on the solution procedures being tested.

For example, the performance of a basic enumeration algorithm such as branch-and-bound is unaffected by changes in costs if dominance rules are not used. Therefore, incorporating changes in costs into the data generation scheme provides no useful information.

It may happen that a conflict exists between satisfying the various properties discussed above. In this case, the purposes of the experiment should resolve the conflict.

2. NOTATION

We need some notation to describe machine scheduling problems. Let $N = \{1, \ldots, n\}$ be the set of jobs. For job $j \in N$, p_j is the processing time, r_j is the release date, w_j is the weight (or value), d_j is the due date, and $\bar{d}_j$ is the deadline. A *finish date*, f_j, denotes either d_j or $\bar{d}_j$. For multiple machine problems, $M = \{1, \ldots, m\}$ is the set of machines and p_{ij} is the processing time of job $j \in N$ on machine $i \in M$. We define the *total slack* to be $\sum_{j \in N}(d_j - r_j - p_j)$.

For precedence constraints, the set of constraints can be represented by an acyclic graph in which each node corresponds to a job. A directed arc from node i to node j means that job i precedes job j, i.e., job j cannot begin processing until job i has completed. We assume that this acyclic graph includes all transitive arcs. For example, if the precedence arcs include (i, j) and (j, k), then the transitive arc (i, k) is also present. Observe that job i is required to precede job j if and only if there exists an arc from node i to node j in the precedence constraint graph. If there is a complete ordering of the jobs, then there are $1 + \cdots + (n-1) = [n(n-1)/2]$ arcs in the precedence constraint graph. We define the *density* of the precedence constraint graph to be (total number of arcs)/$[n(n-1)/2]$. When we want to model only *immediate* predecessor and successor relationships, the graph that represents these relationships is called an immediate precedence graph.

3. A CRITIQUE OF EXISTING GENERATION SCHEMES

We consider several widely used data generation schemes and identify their advantages and disadvantages.

3.1. Release Dates

For problems with release dates, most researchers use a similar scheme (see, for example, Chandra 1979, Dessouky and Deogun 1981, Hariri and Potts 1983, and Posner 1986):

$$p_i \sim U[a_1, b_1],$$

$$r_i \sim U[a_2, b_2],$$

where U denotes the uniform integer distribution. The value of a_1 is usually selected to be one, and a_2 is usually selected to be zero. However, this not always the case. Also, a range of values for b_2 is considered, and this range frequently but not always depends on n.

When this scheme is used, a_2 should be set to zero. The reason is that, if the minimum release date is greater than zero, then a translation of the time axis has zero as the minimum release date. To include other minimum release dates disregards parsimony (Property 8), and adds no new information to the experiment. Similarly, a_1 should usually be set to one.

If b_2 does not depend on n or $\sum p_j$, then the comparability of results for different-size problems is questionable. As n grows, the release dates become closer to each other and become earlier relative to the total processing time. This means that jobs with larger release dates do not have active release date restrictions. This scheme does not have regularity (Property 5).

In the generation scheme of Hariri and Potts (1983), $p_i \sim U[1, 100]$ and $r_i \sim U[0, 50.5nR]$, where $R \in \{0.2, 0.4, 0.6, 0.8, 1.0, 1.25, 1.5, 1.75, 2, 3\}$. For the purpose of comparing the effects of release date tightness to total processing time, we support the use of the expected value of the processing time of a job, which is 50.5 in this case. With this approach, we can change the distribution of the processing times, while retaining the comparability of the tightness over various experiments.

The Hariri and Potts (1983) scheme described above has several of the properties described in §1. These include scale invariance (Property 3), size invariance (Property 4), describability (Property 6), and efficiency (Property 7). One drawback to this scheme is that the interrelease date times are always approximately, but never exactly, exponentially distributed. Therefore, this scheme does not have variety (Property 1). A researcher may want to compare the effects of different arrival distribution patterns, which is not possible using the method of Hariri and Potts (1983).

3.2. Finish Dates

Generating finish dates is more complicated than generating release dates. This is because there are two factors, mean and variance, that contribute to the positioning of due dates and deadlines. Most data generation methods generate the processing times, p_i, independently from a uniform integer distribution, $U(a, b)$, where $0 < a < b$. The due dates are generated from a uniform integer distribution, $U(\alpha h(p_1, \ldots, p_n), \beta h(p_1, \ldots, p_n,))$, where α and β come from a set of small values and h is a functional. Some authors use $h(p_1, \ldots, p_n) = \sum p_i$, while others use the expected value of total processing time. As an example, Potts and Van Wassenhove (1983) use $\alpha \in \{-.2, -.1, \ldots, .9\}$ and $\beta \in \{.7, .8, \ldots, 1.8\}$, where $\alpha < \beta$ for a particular experiment. The actual choices for α and β are based on the selection of a mean and a range for the due dates. According to much of the published research, both the mean and the range are important in determining the computation time of typical branch-and-bound procedures (Potts and Van Wassenhove 1983).

Among the advantages of this scheme are that it is simple and efficient. Also, because a uniform integer distribution is

used for the due dates, the inter-duedate times are approximately exponentially distributed. Moreover, this procedure can be modified to ensure that a feasible schedule always exists for deadline problems.

One difficulty with using the actual values is that every finish date is dependent on *all* of the processing times. This implies that, if the jobs are indexed in earliest finish date order, the finish date of the first job is dependent on the processing time of the nth job. This seems unlikely to occur in practice. Therefore, the generation scheme does not have practical relevance (Property 2). Furthermore, it violates Principle 3 by introducing unintended biases.

A further problem occurs with the objective of maximizing the weighted number of on-time jobs. When data is generated according to the proposed method, the number of due date constraints that are active for the same subset of jobs is usually small (Potts and Van Wassenhove 1988). Most of the subproblems defined by bottleneck due dates are approximately knapsack problems, and are easy to solve computationally. As a result, this scheme does not have variety (Property 1).

Finally, for deadline problems, the scheme may not have regularity (Property 5). An example occurs with the use of the Potts and Van Wassenhove scheme described above. When $h(p_1, \ldots, p_n) = \sum p_i$ and α is close to 1, the slack associated with the early jobs is much greater than that associated with the late jobs.

3.3. Precedence Constraints

A convenient technique for generating precedence constraints, which we call GGEN, is described by Potts (1985) and van de Velde (1995). First, a parameter D, where $0 < D < 1$, is specified. For each pair of nodes i and j in the graph, where $1 \leqslant i < j \leqslant n$, a random number γ_{ij} is generated from the uniform distribution over the interval [0, 1]. If $\gamma_{ij} < D$, then arc (i, j) is included in the graph. The requirement that $i < j$ ensures that the graph is acyclic. If needed, any transitive arcs can then be added.

A difficulty with GGEN is that it is hard to control the number of precedence constraints that are implied by transitivity. Suppose, for example, that n_p jobs precede job i and n_s jobs are successors of job j. Then, adding arc (i, j) to the graph may include another $n_p n_s$ arcs that are implied by transitivity. Thus, the graph may become dense in erratic and unpredictable ways as more arcs are generated. This may make it difficult to achieve a given target density. For example, suppose we have two chains $\{1, \ldots, \lfloor n/2 \rfloor\}$ and $\{\lfloor n/2 \rfloor + 1, \ldots, n\}$, giving a density for the implied precedence constraint graph of $[1 + \cdots + (\lfloor n/2 \rfloor - 1) + 1 + \cdots + (n - \lfloor n/2 \rfloor - 1)]/(n(n-1)/2) \approx 50\%$. If we now add the single arc $(\lfloor n/2 \rfloor, \lfloor n/2 \rfloor + 1)$, then all jobs form a chain, and the density becomes 100%.

A second difficulty with GGEN is that the probability of existence is not the same for all arcs in the precedence graph. Moreover, the precedence constraint graph has a density that is not uniform across the nodes. If $j > i$, $l > k$, and $j - i > l - k$, then arc (i, j) has a greater probability of existing than arc (k, l). As an example, consider a problem with three jobs (and thus three nodes). If arcs (1, 2) and (2, 3) each exist with probability $D = 0.5$, then the probability that there exists a path from node 1 to node 3 in the precedence constraint graph is $0.5 + (1 - 0.5)0.5^2 = 0.625$. The expected degree of nodes 1 and 3 is $0.5 + 0.625 = 1.125$, and of node 2 is $0.5 + 0.5 = 1.0$. The expected density of the graph is $0.5417 > D$.

A third difficulty with GGEN is that for a given value of D, as the number of nodes increases, the density of the precedence constraint graph increases. For example, a graph with five nodes and $D = 0.5$ has an expected density of more than 0.61. Thus, the value of D used in the immediate precedence graph is not a good predictor of the density of the precedence constraint graph, nor of the number of feasible solutions. Further, as the number of nodes increases, the difference in the probabilities that arc $(i, i+1)$ and arc $(1, n)$ exist increases, and the maximum difference in the expected degree of the nodes also increases. As a result, GGEN does not have size invariance and regularity (Properties 4 and 5).

Demeulemeester et al. (1993) suggest methods for generating random activity networks. However, because they construct activity networks, their methods ensure that there is a path from source to sink and explicitly consider the density of the immediate precedence graph. For most machine scheduling problems with precedence constraints, a path from the first job to the last is not a requirement. Also, as noted above, the density of the precedence constraint graph is usually much more relevant than that of the immediate precedence graph.

3.4. Machine Routings and Speeds

For job shop problems, there are important issues associated with the routing of a job through the machines. In most existing schemes, each machine that is not yet a part of the routing has an equal probability of being selected as the next machine. Storer et al. (1992) consider a more general generation procedure. They decompose the set of machines into two groups, and then generate random routings for each group. This generation method increases the likelihood of bottlenecks throughout the shop. They observe that problem difficulty increases when using two group routings instead of the standard one group routings. Both one-and two-group routing instances are generated in the problem test bank of Demirkol et al. (1998). Anecdotal evidence suggests that the computationally hardest job shop problems contain embedded flow shops, which create scheduling bottlenecks. While the work of Storer et al. (1992) is a step towards exploring these beliefs, more robust generation procedures are needed.

With respect to generation of machine speeds for uniform parallel processors, there are not many studies. The standard approach is to generate speeds from a uniform distribution (see Ow 1985, for example). While this scheme

may be applicable in some situations, this generates problems where most of the machines have approximately the same speeds. For example, suppose that we generate nine machine speeds with values $17, 15, \ldots, 1$. Then, machine 1 is seventeen times faster than machine 9. However, machine 1 is less than twice as fast as machines 2, 3, 4 and 5. Consequently, in this scheme, most machines have approximately the same speed. Depending on the features of the experiment, this may introduce an unintended bias into the results.

4. DATA GENERATION FRAMEWORKS

In this section, we make specific proposals for the design of data generation schemes for various scheduling problems. As part of our procedures, we incorporate the ability to generate information for individual jobs. There are numerous instances in manufacturing processes where prior job information influences the release dates of subsequent jobs. JIT systems provide an example (Weiss and Gershon 1993). The ability to generate job information on an individual basis results in a more robust generation scheme. Unfortunately, as discussed in §3, most existing schemes do not permit this possibility.

When generating a set of data there are two types of possible dependencies. Relationships within a given type of data are called *intradependencies*. An example is when the deadline of the next job depends on the deadline of the previous job. *Interdependencies* occur when there are relationships between different types of data. An example is generating data in which jobs with large processing times have large weights. We discuss these relationships in §5.

Suppose there are no intradependencies or interdependencies. Then, for a particular type of data, the data generation process is comparatively simple. First, select an appropriate distribution. Then, sample from this distribution to determine the values of the data. Depending on the experiment, such types of data can include job weights and processing times. Also, setup times usually have no intradependencies or interdependencies and can be generated by this procedure.

The use of the uniform distribution is standard for these types of data. While a researcher may want to consider a distribution with a long tail such as the exponential distribution, we recommend the use of the uniform distribution for most situations. For particular applications, some authors use a truncated normal distribution instead of a uniform distribution (see, for instance, Srinivasan 1971, Baker and Martin 1974, and Ow and Morton 1989). Other studies show that a beta distribution might sometimes be appropriate. For example, Clark (1962) suggests that the beta distribution is a natural choice for generating data for large problems and is also easy to use. Farnum and Stanton (1987) state that, given a finite interval, the beta distribution provides a rich family of distribution functions and is useful when little is known about the actual distribution. Recent experimental evidence suggests that a lognormal distribution might be appropriate for some types of problems (Pinedo 1995).

When there are intradependencies, the generation process can be much more complex. When generating such data, we recommend the use of Principles 2 and 3 as guides. We now discuss several types of data that often have intradependencies. In §4.l, we discuss the generation of release dates. We consider problems with finish dates in §4.2 and with precedence constraints in §4.3. §4.4 discusses machine routings and machine speeds in multiple machine problems.

4.1. Release Dates

We propose a generation method for release dates that provides flexibility in selecting the interrelease date times. Our proposal is to let

$$r_1 = 0 \text{ and } r_i = g_i(p_i, r_{i-i}) + X_i, \quad i = 2, \ldots, n,$$

where X_i is a random variable generated from some probability distribution and $g_i(p_i, r_{i-1})$, $i = 1, \ldots, n$, is a bivariate function of the processing time of the job and the prior release date. Observe that if $g_i(p_i, r_{i-1}) = r_{i-1}$, and X_i comes from an exponential distribution, then we can approximate the generation scheme used by Hariri and Potts (1983). In order to create burstiness of the job arrivals, we can let $g_i(p_i, r_{i-1})$ be a multimodal function. The possibility of using different distributions adds valuable flexibility to the experimental design. The proposed method has Properties 1 and 3 through 8. The requirements of practical relevance (Property 2) depend upon the features of the problem and the features of the experiment. Nevertheless, our method has enough flexibility to meet many typical requirements.

4.2. Finish Dates

We propose a generation method for finish dates which incorporates many of the ideas that are presented in §1. This method can be used to generate data similar to that of Potts and Van Wassenhove (1983), but also provides a much richer set of test problems. Let

$$f_0 = K \text{ and } f_i = h_i(p_1, \ldots, p_n, f_{i-1}) + X_i, \quad i = 1, \ldots, n,$$

where $K \geqslant 0$ and X_i is a random variable generated from some probability distribution. When finish dates do not depend upon processing times, a reasonable choice is to let $h_i(p_1, \ldots, p_n, f_{i-1}) = f_{i-1}$. An example of this type of situation is in a make-to-stock environment. In this environment, the finish date depends only on the prior finish date and the expected processing time, where the expected processing time can be incorporated in the distribution of the X_is. An alternative choice is to let $h_i(p_1, \ldots, p_n, f_{i-1}) = i\mathrm{E}[p_i]$. This is appropriate when finish dates depend upon processing times of previous and current jobs, but the exact values of those processing times are not known. Therefore, their expected value is used as a proxy measure. Setting $h_i(p_1, \ldots, p_n, f_{i-1}) = \sum_{j=1}^{i} p_j$ is useful when there is dependence on the known processing

times of previous and current jobs, as in a typical make-to-order environment.

Each of these choices for h satisfies the properties in §1. In particular, we can increase the number of jobs without changing the characteristics of the data (Property 4). We also note that the data interdependencies in our scheme are more reasonable than in the schemes used by most researchers. Specifically, if the jobs are indexed in earliest finish date order, then f_i does not depend on p_j for any $j > i$. Also, our proposed method provides the ability to change the slack of job j in a way that is uniform across all jobs.

4.3. Precedence Constraints

We present an approach to solve the problems of excessive and nonuniform graph density discussed in §3.3. We recommend adjusting the probability of generating a given arc so that in the precedence constraint graph, each arc has the same probability of existing, and each node has the same expected degree. To do this, let $P_{ij} = \Pr\{\text{arc}(i, j) \text{ exists in the immediate precedence graph}\}$, and let $D =$ target density of the precedence constraint graph $= \Pr\{\text{arc}(i, j) \text{ exists in the precedence constraint graph}\}$, for $1 \leqslant i < j \leqslant n$. Then,

$$
\begin{aligned}
D &= \Pr\{\text{arc}(i, j) \text{ exists in the immediate precedence graph}\} \\
&\quad + \Pr\{\text{arc}(i, j) \text{ doesn't exist, but there is some other path between } i \text{ and } j\} \\
&= P_{ij} + (1 - P_{ij})\big[\Pr\{\text{arc}(i, i+1)\} \cdot \Pr\{\text{arc}(i+1, j)\} \\
&\quad + \Pr\{\neg\text{arc}(i, i+1)\} \cdot \Pr\{\text{arc}(i, i+2)\} \cdot \Pr\{\text{arc}(i+2, j)\} \\
&\quad + \Pr\{\neg\text{arc}(i, i+1)\} \cdot \Pr\{\neg\text{arc}(i, i+2)\} \\
&\quad \cdot \Pr\{\text{arc}(i, i+3)\} \cdot \Pr\{\text{arc}(i+3, j)\} \\
&\quad + \cdots\cdots + \Pr\{\neg\text{arc}(i, i+1)\} \cdots \Pr\{\neg\text{arc}(i, j-2)\} \\
&\quad \cdot \Pr\{\text{arc}(i, j-1)\} \cdot \Pr\{\text{arc}(j-1, j)\}\big] \\
&= P_{ij} + \big(1 - P_{ij}\big)[DD + (1-D)DD + (1-D)^2 DD \\
&\qquad + \cdots + (1-D)^{j-i-2} DD] \\
&= P_{ij} + (1 - P_{ij})D\big(1 - (1-D)^{j-i-1}\big).
\end{aligned}
$$

Therefore,

$$
P_{ij} = \frac{D(1-D)^{j-i-1}}{1 - D\Big(1 - (1-D)^{j-i-1}\Big)}. \tag{1}
$$

Equation (1) can be used to determine the appropriate probability of generating an arc (i, j) to achieve the desired density, D. These values can be calculated once, independently of n, and then used when needed. Table 2 shows the values of P_{ij}, computed to four decimal places for $D \in \{0.1, 0.2, \ldots, 0.9\}$ and $j - i = 1, \ldots, 40$. It can be seen that as $j - i$ increases, the value of P_{ij} decreases more rapidly for higher values of D. This illustrates the intuition that the existence of an arc $(i, i+k)$ is more likely to be implied by transitivity in higher density graphs. Note that if each arc (i, j) is generated with probability P_{ij} according to (1), then the expected degree of each node in the precedence constraint graph is $(n-1)D$. Thus, the expected density is uniform across all the nodes.

To reduce the computational effort required to construct the graph, we recommend that all arcs which leave node i be generated before any of those which leave node $i-1$, for $i = n-1, \ldots, 1$. Also, for the arcs which leave node i, the entering nodes should be considered in index order. Arcs that are part of the transitive closure can be determined by identifying the arcs that originate from the termination of a new arc. For example, suppose arcs $(n-3, n-2)$ and $(n-3, n)$ exist. If we next generate arc $(n-4, n-3)$, then this automatically implies the existence of arcs $(n-4, n-2)$ and $(n-4, n)$. The end points of these implied arcs are exactly the nodes that are successors of node $n-3$.

In some industrial applications, jobs belong to product families, and there may be precedence constraints between the families. For example, in the chemical, paint, and fertilizer industries, restrictions on processing sequences may be used to minimize chemical contamination between job families. Thus, a set of jobs that are all from the same family may have few precedence constraints, while a set of jobs taken from different families may have many. In order to model this situation, we need to generate a precedence constraint graph which contains areas with different densities. To vary the density, we can change the value of P_{ij} as needed. An example is $P_{ij} = (i+j)D/2n$.

4.4. Machine Routings and Speeds

We present an approach for generating a routing of a job through the machines in a job shop. In the simplest case, the routings of the jobs are independent of each other. However, to model embedded flow shops, we need positive correlation between the routings of the jobs.

We recommend an approach based on the use of a set of transition probability matrices. In each matrix, the entry in row i and column j represents the probability that an operation on machine j follows one on machine i. The probabilities in each row to one. After each one-step (i.e., one-operation) routing is generated, we can adjust the matrix. Suppose, for example, that jobs are not permitted to revisit machines. Then, after a job visits a machine, the appropriate entry in the matrix can be assigned a value of zero and the remaining entries in the row can be normalized so that they again sum to one. If the job is allowed to reenter the machine but with a lower probability, similar changes can be made. Also, we can adjust the matrix after a specified number of one-stage routings so that several jobs converge on the same machine, thereby creating a bottleneck.

The probabilities in the matrix can be based on actual transition frequencies within the job shop. Alternatively, randomly generated transition probabilities can be used.

Table 2. Values of P_{ij} which ensure a given expected density

	Expected Density								
$j-1$	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90
1	.1000	.2000	.3000	.4000	.5000	.6000	.7000	.8000	.9000
2	.0909	.1667	.2308	.2857	.3333	.3750	.4118	.4444	.4737
3	.0826	.1379	.1736	.1935	.2000	.1935	.1736	.1379	.0826
4	.0749	.1135	.1282	.1259	.1111	.0876	.0593	.0310	.0089
5	.0679	.0929	.0933	.0795	.0588	.0370	.0185	.0064	.0009
6	.0616	.0757	.0672	.0493	.0303	.0151	.0056	.0013	.0001
7	.0558	.0615	.0480	.0302	.0154	.0061	.0017	.0003	.0000
8	.0505	.0498	.0341	.0183	.0078	.0025	.0005	.0001	.0000
9	.0456	.0403	.0241	.0111	.0039	.0010	.0002	.0000	.0000
10	.0413	.0325	.0170	.0067	.0019	.0004	.0000	.0000	.0000
11	.0373	.0261	.0120	.0040	.0010	.0002	.0000	.0000	.0000
12	.0337	.0210	.0084	.0024	.0005	.0001	.0000	.0000	.0000
13	.0304	.0169	.0059	.0014	.0002	.0000	.0000	.0000	.0000
14	.0275	.0136	.0041	.0009	.0001	.0000	.0000	.0000	.0000
15	.0248	.0109	.0029	.0005	.0001	.0000	.0000	.0000	.0000
16	.0224	.0087	.0020	.0003	.0000	.0000	.0000	.0000	.0000
17	.0202	.0070	.0014	.0002	.0000	.0000	.0000	.0000	.0000
18	.0182	.0056	.0010	.0001	.0000	.0000	.0000	.0000	.0000
19	.0164	.0045	.0007	.0001	.0000	.0000	.0000	.0000	.0000
20	.0148	.0036	.0005	.0000	.0000	.0000	.0000	.0000	.0000
21	.0133	.0029	.0003	.0000	.0000	.0000	.0000	.0000	.0000
22	.0120	.0023	.0002	.0000	.0000	.0000	.0000	.0000	.0000
23	.0108	.0018	.0002	.0000	.0000	.0000	.0000	.0000	.0000
24	.0098	.0015	.0001	.0000	.0000	.0000	.0000	.0000	.0000
25	.0088	.0012	.0001	.0000	.0000	.0000	.0000	.0000	.0000
26	.0079	.0009	.0001	.0000	.0000	.0000	.0000	.0000	.0000
27	.0071	.0008	.0000	.0000	.0000	.0000	.0000	.0000	.0000
28	.0064	.0006	.0000	.0000	.0000	.0000	.0000	.0000	.0000
29	.0058	.0005	.0000	.0000	.0000	.0000	.0000	.0000	.0000
30	.0052	.0004	.0000	.0000	.0000	.0000	.0000	.0000	.0000
31	.0047	.0003	.0000	.0000	.0000	.0000	.0000	.0000	.0000
32	.0042	.0002	.0000	.0000	.0000	.0000	.0000	.0000	.0000
33	.0038	.0002	.0000	.0000	.0000	.0000	.0000	.0000	.0000
34	.0034	.0002	.0000	.0000	.0000	.0000	.0000	.0000	.0000
35	.0031	.0001	.0000	.0000	.0000	.0000	.0000	.0000	.0000
36	.0028	.0001	.0000	.0000	.0000	.0000	.0000	.0000	.0000
37	.0025	.0001	.0000	.0000	.0000	.0000	.0000	.0000	.0000
38	.0022	.0001	.0000	.0000	.0000	.0000	.0000	.0000	.0000
39	.0020	.0001	.0000	.0000	.0000	.0000	.0000	.0000	.0000
40	.0018	.0000	.0000	.0000	.0000	.0000	.0000	.0000	.0000

Groups of jobs can have their own sets of matrices. This creates job groups with similar characteristics. All existing methods for generating machine routings are included in our method.

For generating machine speeds, we recommend a generation process that creates a uniform selection of proportional speeds. That is, speeds should be generated from a logarithmic distribution. In this case, the expected ratio of the speeds of a pair of adjacent machines ordered by speed is constant across all such pairs. This creates instances where the machine speeds are more evenly varied than in the method of Ow (1985).

5. OTHER ISSUES

In some applications, a problem arises with respect to feasibility of the data. Usually, a set of feasible problem instances is needed for testing. There are two basic approaches to generating this set. One approach is to enforce feasibility during the data generation process. For example, one common scheme for generating deadline problems is first to generate all the processing times. Then the deadlines are generated so that a feasible schedule exists. That is, a given deadline d_i is generated from $U[\sum_{j=1}^{i} p_j, \sum_{j=1}^{n} p_j]$. In this example, the last deadline has no degrees of freedom, which is not in accordance with regularity (Property 5).

A second approach is to generate data without regard for feasibility and then discard instances that have no feasible schedule. For example, in a single machine scheduling problem without release dates, sequencing the jobs in deadline order produces a feasible schedule whenever one exists (Jackson 1955). Consequently, feasibility can be checked in polynomial time. Unfortunately, if release dates are included, then checking for feasibility is usually an *NP*-

hard problem (Lenstra et al. 1977). In this case, checking for feasibility requires almost as much effort as finding an optimal schedule, a distinct drawback to the approach. However, it is frequently of interest to determine how long it takes for an optimization procedure to discover that there is no solution. Also, having information about the proportion of solutions that are infeasible provides insight into the data sets used for testing. When possible, the data should not be distorted to ensure feasibility. It is important to keep the test data as free as possible from spurious dependencies and aberrations (Principle 3). Thus, we support the methodology of Potts and Van Wassenhove (1983), in which infeasible solutions are recorded and then discarded.

When there are interdependencies between types of data, steps must be taken to represent these relationships properly. The two major issues that should be addressed are maintaining comparability (Principle 2) and unbiasedness (Principle 3). As opposed to creating relationships and then observing the actual correlation, we recommend that specific expected correlations be created (Moore and Reilly 1993). The former approach can lead to unanticipated high degrees of correlation in a generation scheme (see Potts and Van Wassenhove 1988, John 1989, and Hariri and Potts 1991). However, inducing a specific level of expected correlation permits more accurate study of the effects of correlation on the performance of solution procedures, and also helps to prevent biased data sets.

6. CASE STUDIES

We now illustrate our recommendations for the design of a data generation scheme for scheduling problems. In §6.1 and 6.2, we consider two problems with a variety of features, discuss some relevant factors, suggest specific generation schemes, and offer some supporting comments. Section 6.3 summarizes our conclusions from these case studies.

6.1. Minimizing Maximum Tardiness with Release Dates

Features of the Problem.

Application: Online assignment of aircraft to gates after a service disruption.

Objective: Minimize maximum tardiness, motivated by customer service requirements.

Constraints: Release dates are determined by airplane arrivals.

Complexity: Unary *NP*-hard.

Procedure being tested: Branch and bound, using job dominance rules to prune the tree.

Job arrival process: Random arrivals from an external source, release dates are known before due dates are determined.

Processing times: Normally distributed (truncated below a known lower bound on aircraft service time), values are not known before due dates are decided.

Feasibility: Not an issue, because there are no deadline constraints.

Features of the Experiment.

Evaluate performance of the procedure over a large variety of problem instances that simulate the real problem.

Implement the procedure if successful, and publish the results.

Proposed Generation Scheme.

$r_0 = 0$ and $r_i = r_{i-1} + X_i$, where $X_i \sim \exp(\lambda)$ for $i = 1, \ldots, n$.

$p_i \sim N(\mu, \sigma)$ for $i = 1, \ldots, n$, truncated below a known lower bound.

$d_i = r_i + kE[p_i]$ for $i = 1, \ldots, n$, where $k \geqslant 1$.

Comments. Given the need to evaluate performance, Principle 1 suggests the need for variety and practical relevance. Principle 3 implies the need for scale and size invariance, and regularity. Since one purpose is to publish and implement the results, Principle 4 implies the need for describability and efficiency. Allowing λ, μ, σ, and k to vary creates a wide variety of problem instances while maintaining parsimony. Because all inputs of the same type are treated similarly, the proposed scheme has regularity. The above generation scheme satisfies all of the properties presented in §1.

Both the release dates and due dates are generated according to the methods described in §4. For this application, the release date of a job is independent of the processing time. Because the actual processing time is not known, each due date is calculated from the expected processing time. To determine algorithmic performance for different size problems, n should be varied.

6.2. Minimizing Weighted Completion Time with Deadlines

Features of the Problem.

Application: Make-to-order supplier of parts to an auto manufacturer.

Objective: Total weighted completion time, motivated by the desire to reduce inventory costs.

Constraints: Deadlines, motivated by customer requirements.

Complexity: Unary *NP*-hard.

Procedures being tested: Several heuristics. A lower bound is used to determine the accuracy of each heuristic for large-size problems.

Job arrival process: Batch arrivals from an upstream internal process.

Processing times: Beta distributed, known before deadlines are decided.

Weights: The labor component is highly correlated to processing time. The raw material component is not correlated.

Precedence constraints: The assembly process has general job precedence constraints.

Feasibility: Need to identify feasible schedules. Include results when the heuristic cannot find a feasible schedule although one exists, and when no feasible schedule exists.

Features of the Experiment.

Evaluate performance of several heuristics over a set of problem instances that simulate the real-world problem.

Identify and implement the best heuristic. Publishability is not a requirement.

Proposed Generation Scheme.

$p_i \sim \beta[a_1, b_1]$ for $i = 1, \ldots, n$.

$d_0 = K$ and $d_i = d_{i-1} + X_i$, where $X_i = k \cdot \exp(E[p_i])$ for $i = 1, \ldots, n$.

$w_i = p_i + Y_i$, where $Y_i \sim U[a_2, b_2]$ for $i = 1, \ldots, n$ is the raw material component.

Precedence constraints: the values of P_{ij} are generated as discussed in §4.3.

Comments. Given the need to evaluate real-world performance, Principle 1 suggests the need for practical relevance. Since several heuristics are being compared, Principle 2 implies the need for size invariance and scale invariance. Principle 3 suggests the need for regularity. Because publishing the results is not planned, describability is not required. Further, since only a small variety of tests will be performed, Principle 4 does not require efficiency. However, it is still sensible to satisfy parsimony. The data has scale invariance and size invariance. It also has regularity, which is important because of the comparisons being made between different procedures.

Because inventory cost is related to the value of an item, which in turn is a function of processing time, the processing time information is used to generate weights. The proposed generation scheme does not guarantee the existence of a feasible schedule. For large size problems, it is not possible to determine in a reasonable time whether a feasible schedule exists. Consequently, the information about the heuristic's ability to find feasible solutions must be inferred from smaller size problems. Variations in a_1, b_1, a_2, b_2, K, and k may have a significant effect on the heuristic's performance. Similarly, the target density of the precedence graph, D, affects the number of feasible solutions, which in turn may affect heuristic performance.

6.3. Summary

The two case studies above illustrate the complexity of designing a data generation scheme. As mentioned in the introductory section, a single data generation scheme cannot be used in all circumstances. Many variations are necessary, depending upon the problem, the solution procedures being tested, and the purposes of the test. However, the principles, features, and properties that we consider in each experiment are similar.

7. CONCLUDING REMARKS

We demonstrate that some frequently used schemes for the generation of data for machine scheduling problems are biased, inflexible, unrealistic, and hard to justify. By considering a set of principles and properties, we develop general frameworks for data generation. These frameworks offer greater flexibility than previous schemes, and also provide the ability to avoid previous biases. For example, we present a scheme for generating precedence constraint graphs with a target density that is uniform. Our frameworks use various components depending upon the specifics of each situation. We indicate some of the issues that should be considered in designing those components. Our work should help future researchers to develop generation schemes that can be better justified from basic principles and properties, that contain fewer inherent biases, and that are more consistent and comparable with schemes developed by other researchers.

The format we present in §6 for generating data sets can be used in a variety of situations. As suggested by Hammer and Shanno (1999), it may be possible to elicit information on confidential data sets by using this format to obtain the specifications of the generation process.

Problems for future study include the development of more detailed guidelines for many specific situations that arise in generating data. An example for scheduling problems might be the generation of deadlines that are agreeable with other parameters. Also, given a choice of procedures, it is useful to determine which is most effective for a given data set. The development of indicators that relate to this choice is a new area of research (see Yang 1998, for example). However, this research is essential for the development of more effective computational testing. We hope that our work stimulates further research in these directions.

ACKNOWLEDGMENTS

This research is supported in part by the National Science Foundation under grant numbers DMI-9821033 and DMI-9908437, and in part by the Summer Fellowship Program, Fisher College of Business, The Ohio State University.

REFERENCES

Ahuja, R. K., T. L. Magnanti, J. B. Orlin. 1993. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, Englewood Cliffs, NJ.

Baker, K. R., J. B. Martin. 1974. An experimental comparision of solution algorithms for the single-machine tardiness problem. *Naval Logis. Res. Quart.* **21** 187–199.

Barr, R. S., B. L. Golden, J. P. Kelly, M. G. C. Resende, W. R. Stewart, Jr. 1995. Designing and reporting on computational experiments with heuristic methods. *J. Heuristics* **1** 9–32.

Chandra, R. 1979. On $n/1/\overline{F}$ dynamic deterministic problems. *Naval Res. Logist. Quart.* **26** 537–544.

Clark, C. E. 1962. The PERT model for the distribution of an activity time. *Oper. Res.* **10** 405–406.

Crowder, H. P., R. S. Dembo, J. M. Mulvey. 1978. Reporting computational experiments in mathematical programming. *Math. Programming* **15** 316–329.

——, P. B. Saunders. 1980. Results of a survey on MP performance indicators. *COAL Newsletter*(Jan) 2–6.

Demeulemeester, E., B. Dodin, W. Herroelen. 1993. A random activity network generator. *Oper. Res.* **41** 972–980.

Demirkol, E., S. Mehta, R. Uzsoy. 1998. Benchmarks for shop scheduling problems. *Eur. J. Oper. Res.* **109** 137–141.

Dessouky, M. I., J. S. Deogun. 1981. Sequencing jobs with unequal ready times to minimize mean flow time. *SIAM J. Comput.* **10** 192–202.

Farnum, N. R., L. W. Stanton. 1987. Some results concerning the estimation of beta distribution parameters in PERT. *J. Oper. Res. Soc.* **38** 287–290.

Fisher, M. L. 1980. Worst case analysis of heuristic algorithms. *Management Sci.* **26** 1–17.

Fleischer, M., S. H. Jacobson. 1999. Information theory and the finite-time behavior of the simulated annealing algorithm: Experimental results. *INFORMS J. Comput.* **11** 35–43.

Florian, M., B. Fox, H. Crowder, R. Dembo, J. Mulvey. 1979. Reporting computational experience in *Operations Research. Oper. Res.* **27** vii–x.

Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of* NP-*Completeness.* Freeman, San Francisco, CA.

Greenberg, H. J. 1990. Computational testing: Why, how and how much. *ORSA J. Comput.* **2** 94–97.

Hammer, P. L., D. F. Shanno. 1999. Private communication.

Hariri, A. M., C. N. Potts. 1983. An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Appl. Math.* **5** 99–109.

——, ——. 1991. Heuristics for scheduling unrelated parallel machines. *Comput. and Oper. Res.* **18** 323–331.

Hoffman, K. L., R. H. F. Jackson. 1982. In pursuit of a methodology for testing mathematical programming software. *Proc. Conf. Evaluating Math. Programming Techniques* Boulder, CO. 177–199.

Hooker, J. N. 1994. Needed: An empirical science of algorithms. *Oper. Res.* **42** 201–212.

——. 1995. Testing heuristics: We have it all wrong. *J. Heuristics* **1** 33–42.

Jackson, J. R. 1955. Scheduling a production line to minimize maximum tardiness. Research Report **43** Management Science Research Project, University of California, Los Angeles, CA.

Jackson, R. H. F., P. T. Boggs, S. G. Nash, S. Powell. 1991. Guidelines for reporting results of computational experiments: Report of the *ad hoc* committee. *Math. Programming* **49** 413–425.

John, T. C. 1989. Tradeoff solutions in single machine production scheduling for minimizing flow time and maximum penalty. *Comput. and Oper. Res.* **16** 471–479.

Klee, V., G. J. Minty. 1972. How good is the simplex algorithm? O. Shisha, ed., *Inequalities III*, Academic Press, New York, 159–175.

Lawler, E. L., J. K. Lenstra, A. H. G. Rinnoy Kan, D. B. Shmoys, eds. 1985. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* Wiley, New York.

Lee, C-Y., J. Bard, M. Pinedo, W. E. Wilhelm. 1993. Guidelines for reporting computational results in *IIE Transactions. IIE Trans.* **25** 121–123.

Lenstra, J. K., A. H. G. Rinnooy Kan, P. Brucker. 1977. Complexity of machine scheduling problems. *Ann. Discrete Math.* **1** 343–362.

Lu, L., M. E. Posner. 1993. An NP-hard open shop scheduling problem with polynomial average time complexity. *Math. Oper. Res.* **18** 12–38.

McGeoch, C. C. 1996. Toward an experimental method for algorithm simulation. *INFORMS J. Comput.* **8** 1–15.

Moore, B. A., C. H. Reilly. 1993. Randomly generating synthetic optimization problems with explicitly induced correlation. OSU/ISE Working paper 1993-002, The Ohio State University, Columbus, OH.

Ow, P. S. 1985. Focused scheduling in proportionate flowshops. *Management Sci.* **31** 852–869.

——, T. E. Morton. 1989. The single machine early/tardy problem. *Management Sci.* **35** 177–191.

Pinedo, M. 1995. *Scheduling: Theory, Algorithms, and Systems.* Prentice-Hall, Englewood Cliffs, NJ.

Posner, M. E. 1986. A sequencing problem with release dates and clustered jobs. *Management Sci.* **32** 731–738.

Potts, C. N. 1985. A Lagrangean based branch and bound algorithm for single machine sequencing with precedence constraints to minimize total weighted completion time. *Management Sci.* **31** 1300–1311.

——, L. N. Van Wassenhove. 1983. An algorithm for single machine sequencing with deadlines to minimize total weighted completion time. *Eur. J. Oper. Res.* **12** 379–387.

——, ——. 1988. Algorithms for scheduling a single machine to minimize the weighted number of late jobs. *Management Sci.* **34** 843–858.

Srinivasan, V. 1971. A hybrid algorithm for the one machine sequencing problem to minimize total tardiness. *Naval Res. Logist. Quart.* **18** 301–313.

Storer, R. H., S. D. Wu, R. Vaccari. 1992. New search spaces for sequencing problems with application to job shop scheduling. *Management Sci.* **38** 1495–1509.

van de Velde, S. L. 1995. Dual decomposition of a single-machine scheduling problem. *Math. Programming* **69** 413–428.

Weiss, H. J., M. E. Gershon. 1993. *Production and Operations Management,* 2nd ed. Allyn and Bacon, Needham Heights, MA.

Yang, J. 1998. Scheduling with batch objectives. Doctoral dissertation, Industrial and Systems Engineering, The Ohio State University, Columbus, OH.