Hindawi Publishing Corporation
The Scientific World Journal
Volume 2014, Article ID 370656, 11 pages
http://dx.doi.org/10.1155/2014/370656

Research Article

Constraint Violations in Stochastically Generated Data: Detection and Correction Strategies

Adam Fadlalla[1] and Toshinori Munakata[2]

[1] *Department of Accounting and Information Systems, Qatar University, P.O. Box 2713, Doha, Qatar*
[2] *Department of Computer and Information Science, Cleveland State University, Cleveland, OH 44114, USA*

Correspondence should be addressed to Toshinori Munakata; t.munakata@csuohio.edu

Received 9 August 2013; Accepted 19 October 2013; Published 4 February 2014

Academic Editors: P. Bala and Y.-P. Huang

We consider the generation of stochastic data under constraints where the constraints can be expressed in terms of different parameter sets. Obviously, the constraints and the generated data must remain the same over each parameter set. Otherwise, the parameters and/or the generated data would be inconsistent. We consider how to avoid or detect and then correct such inconsistencies under three proposed classifications: (1) data versus characteristic parameters, (2) macro- versus microconstraint scopes, and (3) intra- versus intervariable relationships. We propose several strategies and a heuristic for generating consistent stochastic data. Experimental results show that these strategies and heuristic generate more consistent data than the traditional discard-and-replace methods. Since generating stochastic data under constraints is a very common practice in many areas, the proposed strategies may have wide-ranging applicability.

1. Introduction

The use of stochastically (randomly) generated data is very common in various domains and for various reasons including:

(i) the Monte Carlo method where events (samples and data) are simulated by random numbers,

(ii) verification of complex analytical solutions,

(iii) assessment of heuristic methods through randomly generated test data,

(iv) the so-called guided random search techniques, such as genetic algorithms and neural networks [1], which are particularly suited for search and optimization problems.

In these applications, certain probability distributions are assumed when generating random data. The distribution can be static or dynamic, depending on whether its associated parameters are fixed or are changing over time. Usually the generated data has to satisfy constraints in addition to the probability distribution constraints. All these constraints can be formulated in terms of different parameter sets. Each parameter set formulation must equivalently represent the same constraints.

We consider a problem where n events are randomly generated and each event is represented by k variables, $x_1, \ldots, x_k$. For a simple queuing problem, n can represent the number of units to arrive at a service station, and x_1 and x_2 are service time and arrival time for each unit, respectively. Generally, the n samples can be represented by a $k \times n$ matrix as follows: $X = [x_{ij}]$, $i = 1, k$, $j = 1, n$, where x_{ij} represents the value of variable x_i for the jth sample. We note that the ith row of matrix X represents n samples of variable x_i and the jth column represents the values of k variables, $x_1, \ldots, x_k$ for the jth sample. In this paper each x_{ij} is assumed as a scalar, for example, a real number between 0 and 1, or an integer in a certain range, say, between 0 and 10, depending on problem specifics. We generate x_{ij} randomly under a certain probability distribution and certain constraints. When we extend the assumption of each x_{ij} being a scalar, for example, to a vector, we would consider a scalar element of the vector.

Let the set of constraints to be imposed on X be $C(X)$. A constraint can be *intravariable*, meaning that it is imposed on a single variable. For example, there may be lower and upper bounds for each variable x_i as $x_{iL} \leq x_i \leq x_{iU}$, $i = 1, k$. The set of such constraints for all x_i, $i = 1, k$, would then represent a "hyperbox" in a k-dimensional space for x_i, $i = 1, k$. A constraint can be *intervariable*, meaning it is imposed between multiple variables; for example, for specific values of i and i', $x_{ij} \geq x_{i'j}$, $j = 1, n$. In a k-dimensional space for $x_i, i = 1, k$, this constraint indicates that all sample points must be in one side of the hyperplane of $x_{ij} = x_{i'j}$, $j = 1, n$. All the above-mentioned constraints can be elements of $C(X)$, and they represent an ultimate set of constraints, such that every constraint in $C(X)$ must be satisfied, and conversely, if every constraint in $C(X)$ is satisfied, the generated random data is $C(X)$ valid.

Let $S_1 = (\pi_{11}, \ldots, \pi_{1\mu})$ and $S_2 = (\pi_{21}, \ldots, \pi_{2\nu})$ be two sets of parameters associated with a problem. Let the sets of constraints to be imposed on S_1 and S_2 be $C(S_1)$ and $C(S_2)$, respectively. Obviously the constraints $C(X)$, $C(S_1)$, and $C(S_2)$, or n data samples generated under these constraints, must be consistent with each other. For example, if $C(X)$ says all variables must be nonnegative and some of randomly generated variables under $C(S_2)$ are negative, $C(X)$ and $C(S_2)$ are not consistent and the resulting data is not $C(X)$ valid.

We can consider multidimensional spaces defined on X, S_1, and S_2. For X, it is a k-dimensional space corresponding to k variables, $x_1, \ldots, x_k$. For S_1 and S_2, they are μ- and ν-dimensional spaces, respectively, corresponding to the number of parameters. The set of constraints for each space will specify a certain region (domain) within the space. For example, for X, $C(X)$ will specify a certain domain in the k-dimensional space. Randomly generated n events for a specific run will be represented as scattered points within the domain. For S_1 and S_2, $C(S_1)$ and $C(S_2)$, respectively, will specify their domains. For example, for S_1, parameters $\pi_{11}, \ldots, \pi_{1\mu}$ must be confined within the domain that satisfies $C(S_1)$.

The major contributions of this article are to

(1) point out some potential inconsistencies of random data generation,

(2) discuss methods of detecting such inconsistencies,

(3) propose techniques of avoiding or correcting such inconsistencies.

We illustrate these concepts on three widely researched problems: a queuing problem, fluid dynamics, and the total tardiness problem. The total tardiness problem is an NP-hard job scheduling problem [2] that "continues to attract significant research interest from both a theoretical and a practical perspective" [3].

We characterize two types of parameters called *data* and *characteristic* parameters for easy reference. The former are often naturally derived parameters directly associated with the random data. The latter are additional parameters introduced to better represent the characteristics of the problem overall, such as the difficulty of the problem. These parameter sets can be corresponded to S_1 and S_2 described earlier.

Section 2 considers simple examples to show how constraint violations may occur among different parameter sets. In Sections 3 and 4, details of the total tardiness problem are discussed in the form of a case study. In particular, Section 3 reviews the total tardiness problem and Section 4 discusses various types of constraints and how their violations can occur. Sections 5–8 discuss various correction algorithms for different types of constraint violations. Section 9 presents results of a numerical experiment. Section 10 provides general guidelines for generating stochastic data under constraints and recommends possible further studies. The basic concepts discussed in the problems are applicable to any other problems that employ random data involving parameter sets and constraints.

2. Simple Illustrations of Multiple Parameter Sets

We consider two relevant examples in this section—a simple queuing problem [4] and fluid dynamics [5].

2.1. Queuing Problem. Customers arrive stochastically at a service station of m servers at rate λ and are served at rate μ for each server ($m\mu$ represents the service rate at the entire station). These parameters m, λ, and μ can be considered as data parameters since they are directly associated with the randomly generated data. In addition, the traffic intensity $\rho = \lambda/(m\mu)$ is the steady-state fraction of the time in which the server is busy, and characterizes the difficulty of the problem—the higher the ρ value, the harder the problem. Thus, the parameter ρ can be considered as a characteristic parameter, since it is derived indirectly from data parameters and it is for the purpose of characterizing the problem as a whole.

When constraints are imposed on different parameter sets, constraints on one parameter set must be consistent with constraints on the other parameter set so that randomly generated data are consistent under both parameter sets. Common constraints on ρ are $0 \leq \rho \leq 1$; the first nonnegative condition must hold since all the parameters involved are nonnegative. The second condition is assumed since the queue grows infinitely otherwise. One can perform simulation of the queuing problem selecting various values of m, λ, and μ, under certain types of probability distributions (e.g., Poisson and exponential) for arrivals and services. Suppose that we arbitrarily set the ranges of m, λ, and μ as, for example, $m = [1, m_{\max}]$, $\lambda = [0, \lambda_{\max}]$, and $\mu = [0, \mu_{\max}]$, and consider all discrete combinations of (m, λ, μ) in these ranges. Obviously some (m, λ, μ) triplets can violate the $0 \leq \rho \leq 1$ constraints (e.g., triplets with $m = 1$, $\lambda > \mu$ are violations).

Such constraint violations may be trivial in the above queuing example since the number of parameters is small, and the associated constraints are straightforward. When the number of parameters becomes larger and the associated constraints are more complex or dynamic, however, violations may not be so obvious.

2.2. Fluid Dynamics. Many fluid dynamics phenomena are highly nonlinear and present challenging problems both theoretically and experimentally. For example, in the year 2000 a US based mathematical society announced the seven "Millennium Prize Problems." These are considered some of the world's hardest unsolved problems and a $1 million prize is posted for each question. One problem is in fluid dynamics and is concerned with the Navier-Stokes equation.

Since solving the whole Navier-Stokes equation is very difficult, usually researchers consider special cases under certain assumptions or simplifications. For this purpose, a characteristic parameter called the Reynolds number, R, is commonly employed:

$$R = \frac{dvL}{\eta}, \tag{1}$$

where d is the density, v the velocity, L a characteristic length scale, and η the viscosity. R characterizes the flow's likelihood of being turbulent or laminar; the higher the R is, the more likely there is to be turbulence. Suppose one wants to study a hard fluid dynamics problem that involves random noise. One would numerically experiment with the flow by generating random data for the noise and by considering various values of R and the data parameters of the right-hand side.

This is similar to the scenario discussed in the queuing problem. For example, suppose we consider only discrete values of the parameters. Similar to the queuing problem, upper and lower bounds can be set for each parameter. Constraints on R, $R_{\min} \le R \le R_{\max}$ may indicate the study is to be performed only for nonturbulent, laminar flows or certain types of turbulent flows. If we consider all combinations of d, v, L, and η, some parameter value combinations may not be consistent with the constraints for R and the resulting data may be flawed. The types of characteristic and data parameters discussed in this section are common in many disciplines; hence we should be cautious when random data generation is considered.

3. The Total Tardiness Problem: A Case Study

We briefly describe the static total tardiness problem; that is, parameter values do not change dynamically. At time $t = 0$, n jobs wait for processing. For the simplest, single machine model, each job is processed by the single machine one at a time. For each problem instance, the n jobs have *processing times* $p_1, p_2, \ldots, p_n$ and *due dates* $d_1, d_2, \ldots, d_n$, respectively. The objective of the total tardiness problem is to determine the order of jobs to be processed to minimize the *total tardiness*, that is, the total number of days past due dates of tardy jobs (jobs whose completion times exceed their due dates). (Note that although "days" are used here, any other time units such as hours and minutes can be employed depending on a specific application.)

We use the following notations: subscripts: s: "preset" or assigned, e: "expected," a: "actual," L: lower bound, U: upper bound; parameters: n: number of jobs in each problem instance; (note: in job scheduling research including tardiness it is customary to call "a problem instance" simply "a problem" and we follow this practice hereafter. A problem is specific values of n jobs with associated values of p_is and d_is)

p: processing time: processing time for each job is denoted by p_i, $1 \le i \le n$;

p_L, p_U: pre-set lower and upper bounds of processing time;

$\overline{p}$: average processing time: in most research works, a uniform distribution between p_L and p_U is assumed for processing time p_i; in this case, the pre-set average processing time $\overline{p}_s$ is $(p_L + p_U)/2$; this value may or may not be the same as the expected average processing time $\overline{p}_e$ for a specific random data generation method and is typically different from the actual average processing time $\overline{p}_a$ for a specific run for a specific random data generation method; the "expected average" of any variable x_i, $1 \le i \le n$, is computed by $\sum_{i=1}^{n} P_i x_i$, where P_i is the probability of x_i.

d: due date: due date for each job is denoted by d_i, $1 \le i \le n$; other notations for processing time, such as lower and upper bounds, and average for pre-set, expected and actual, also apply for *due date*; in particular, a uniform distribution between d_L and d_U is commonly assumed for due date d_i; in this case, the pre-set average due date $\overline{d}_s$ is $(d_L + d_U)/2$.

Two common parameters, τ and δ, called the *tardiness factor* and *relative range* of *due dates*, respectively, are employed to characterize a specific tardiness problem. Their definitions and meanings are as follows:

$$\tau = 1 - \frac{\overline{d}_s}{(n \cdot \overline{p}_s)} = 1 - \frac{(d_L + d_U)}{(2n \cdot \overline{p}_s)}, \quad 0 \le \tau \le 1, \tag{2}$$

$$\delta = \frac{(d_U - d_L)}{(n \cdot \overline{p}_s)}, \quad 0 \le \delta \le 1. \tag{3}$$

The tardiness factor τ represents the average ratio of the number of jobs that do not finish on time over the total number of jobs. The relative range of due dates, δ, is a measure for the due date span $(d_U - d_L)$ over the total processing time $(n \cdot \overline{p}_s)$. In practice, τ and δ may be pre-set to specific values (e.g., $\tau = 0.8$ and $\delta = 0.2$) first, and then d_U and d_L may be subsequently determined. This can be achieved by solving the above two equations for d_L and d_U in terms of τ and δ as follows:

$$d_L = \frac{(n \cdot \overline{p}_s)\{2(1-\tau) - \delta\}}{2}, \tag{4}$$

$$d_U = \frac{(n \cdot \overline{p}_s)\{2(1-\tau) - \delta\}}{2}. \tag{5}$$

Note that d_L and d_U differ only for the sign of δ.

In the total tardiness problem, d_L and d_U are data parameters, while τ and δ are characteristic parameters [6]. It is common that stochastic data is generated under constraints that are expressed in terms of different parameter sets. In the illustration here, $\{d_L, d_U\}$ is the set of data parameters, and $\{\tau, \delta\}$ is the set of characteristic parameters. Each parameter set (e.g., $\{\tau, \delta\}$) should satisfy the same constraints imposed by the other parameter set (e.g., $\{d_L, d_U\}$); otherwise, the constraints are violated, resulting in inappropriate data.

Extensive research on solution methods for the tardiness problem has been undertaken. One category of these methods is heuristic approaches which include [7–14].

4. Common Data Generation Method for the Total Tardiness Problem

Previous works [12–18] studied the performance of their heuristic algorithms by randomly generating test data. Typically, the test data were generated as follows.

For each job i, $1 \le i \le n$, an integer processing time p_i is generated from the uniform distribution (p_L, p_U); for example, $p_L = 1$, $p_U = 100$. Then $n \cdot \overline{p}_a = \sum_{i=1}^{n} p_i$ is computed, and the τ and δ values are selected from the $\{0.2, 0.4, 0.6, 0.8, 1.0\}$ set. That is, there will be $5 \times 5 = 25$ pairs of (τ, δ) as $(0.2, 0.2), (0.2, 0.4), \ldots, (1.0, 1.0)$. Then for each job i, an integer due date d_i is generated from the uniform distribution $[d_L, d_U]$, where $d_L = (n \cdot \overline{p}_a)\{2(1-\tau) - \delta\}/2$ and $d_U = (n \cdot \overline{p}_a)\{2(1-\tau) + \delta\}/2$. Five problems are generated for each pair of τ and δ values, yielding a total of $5 \times 25 = 125$ problems.

There are two obvious data constraints in the total tardiness problem.

(1) Nonnegative due date. The due date for every job must be non-negative; that is, $d_i \ge 0$, for $1 \le i \le n$.

(2) Due date $\ge$ processing time for every job. It is reasonable to assume that, no one will accept a job that takes more time to process than its due date. We would be penalized even if we start such a job first. That is, $d_i \ge p_i$ must hold for $1 \le i \le n$.

We note that the first constraint is imposed on *individual* variable d_i; hence we call it *intravariable* constraint. The second constraint is between two variables (d_i, p_i); hence we call it *intervariable* constraint. As discussed later, intervariable constraints are typically harder to deal with than intra-variable constraints. We also note that data parameters $\{d_L, d_U\}$ and characteristic parameters $\{\tau, \delta\}$ provide *macro*- (i.e., *problem-level*) characterization of the problem. On the other hand, the intra- and intervariable constraints ($d_i \ge 0$ and $d_i \ge p$, resp.) provide *micro*- (i.e., *job-level*) characterization of the problem. These basic concepts of intra- versus inter-variable constraints and macro- versus microparameters help to better define the nature of the problem.

The above data generation procedure will yield violations of both intra- and intervariable constraints [19], depending on the values of τ, δ, n, p_L, and p_U. Why this procedure yields violations can be best understood by examining Figures 1 and 2. Figure 1 explains why some of the 25 pairs of (τ, δ) described above lead to intravariable constraint violations (negative due dates). The basic reason is that these (τ, δ) combinations (in the shaded area of the triangular sub-domain BCD in Figure 1(a)) correspond to negative d_L values (in the shaded area of the triangular sub-domain $\mathrm{B'C'D'}$ in Figure 1(b)). The negative d_L values lead to some of the randomly generated d_i values becoming negative. Figure 2 illustrates a situation where inter-variable constraint violations occur.

When we compare the shaded/dotted areas of Figures 1(a) and 1(b), we see that there is one-to-one correspondence between every point in Figure 1(a) and every point in Figure 1(b). For example, this is true for the trapezoid ABDE in Figure 1(a) and the trapezoid $\mathrm{A'B'D'E'}$ in Figure 1(b). When two domains (e.g., ABDE and $\mathrm{A'B'D'E'}$) in two different parameter spaces have this one-to-one correspondence property, we say the two domains are *constraint isomorphic* (or simply *isomorphic*). When two domains are isomorphic and constraint violations are identified in one domain, this will help to easily determine constraint violations in the other domain [6]. Similar constraint violation problems are also considered in [20, 21]. We consider correction algorithms for intravariable constraint violations in Section 5 and correction algorithms for intervariable constraint violations in the succeeding sections.

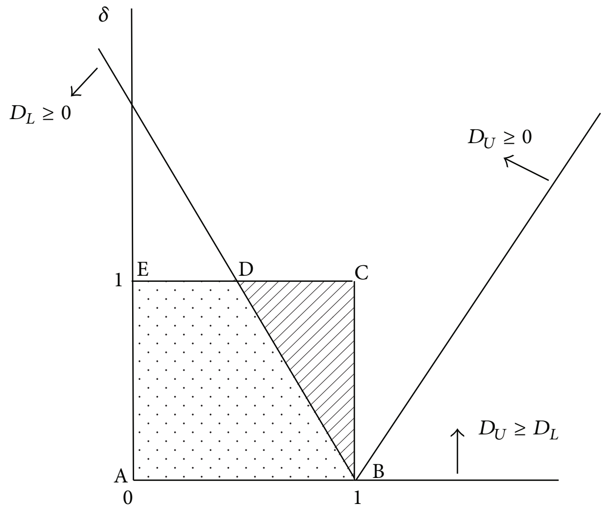
5. Correction Algorithms for Intravariable Constraint Violations

We consider various algorithms to avoid or correct intravariable constraint violations.

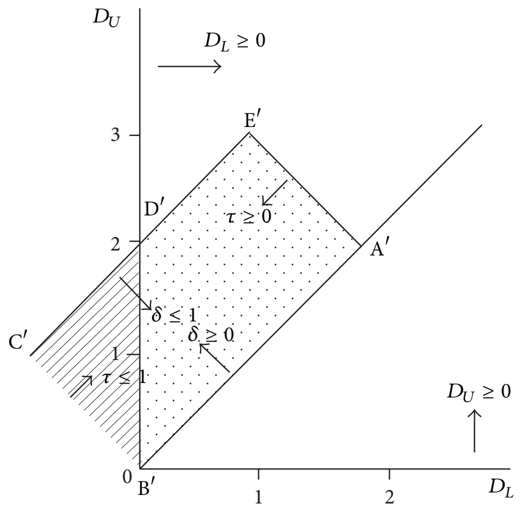
Safe-Zone Algorithm. Use only (τ, δ) combinations that yield no violations, that is, those in the trapezoid ABDE in Figure 1(a). This is the simplest and easiest approach. When we adopt this algorithm, out of the 25 pairs of (τ, δ) that were used in the previous work cited earlier, nine combinations would not be considered as they will lead to intravariable constraint violations. These are $(\tau, \delta) = (0.6, 1.0)$, $(0.8, 0.6)$, $(0.8, 0.8)$, $(0.8, 1.0)$, $(1.0, 0.2)$, $(1.0, 0.4)$, $(1.0, 0.6)$, $(1.0, 0.8)$, and $(1.0, 1.0)$.

We note that although this approach is the best for its simplicity and was used by many researchers (e.g., [22, 23]), whether it can be legitimately employed is another issue. The problem characteristics, rather than the simplicity of constraint satisfaction, should dictate the selection of parameters. If the problem requires combinations outside the safe zone, we may have to give up this algorithm and rely on other approaches.

Discard-and-Replace Algorithm. Whenever negative d_i is generated, either replace it with a constant [24] or simply discard it and continue data generation until the next non-negative d_i is generated [25]. This is a widely used approach for stochastic data generation in general. We must be cautious though, since this discard-and-replace process in general may alter our original data intentions. Here we address three problematic issues relating to (1) the data constraints, (2) the data distribution (uniform, normal, etc.) characteristics, and



(a) The square ABCDE represents $0 \leq \tau \leq 1$ and $0 \leq \delta \leq 1$, where $5 \times 5 = 25$ pairs of (τ, δ) as $(0.2, 0.2)$, $(0.2, 0.4), \ldots, (1.0, 1.0)$ being previously taken for simulation. The triangular subdomain BCD shows the invalid area where constraint $D_L \geq 0$ is violated as shown in (b)



(b) The rectangle A′B′C′D′E′ corresponds to the square ABCDE in (a). The shaded triangle B′C′D′ is an invalid subdomain where $D_L < 0$

FIGURE 1: Two-dimensional spaces illustrating invalid subdomains where $D_L < 0$.

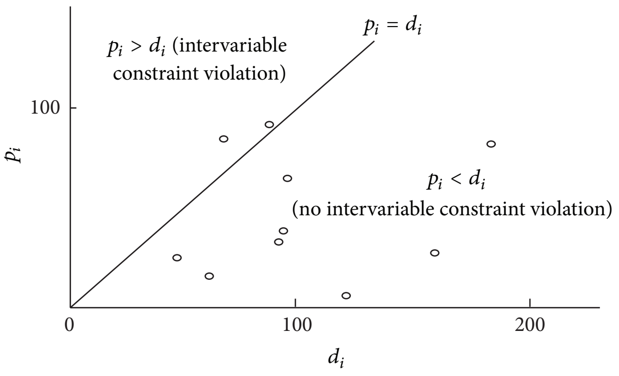


FIGURE 2: Sample space (d_i, p_i) for Example 1. Two dots above the $p_i = d_i$ line are inter-variable constraint violations.

(3) the characteristic constraints (e.g., (τ, δ) values). When we employ the discard-and-replace method, obviously (1) the data constraints are satisfied, but (2) the distribution properties and (3) the characteristic values may or may not remain the same. As to the data distribution, if the original distribution is uniform, it is likely that this is preserved (because every d_i is equally likely to be picked over the range $[0, d_U]$). But if the original distribution is nonuniform, by chopping off portion of the range of the random variable (as, for, e.g., $d_i < 0$) it is likely to change the distribution. In case of the normal distribution, the leftmost portion may be cut off, resulting in a nonsymmetric distribution; that is, it is no longer a normal distribution in the new range.

When we apply the discard-and-replace method to our intravariable constraint violations, the above discussion of data distribution holds. If we assume a uniform distribution as most literature in the total tardiness problem has, the new distribution will remain uniform (again because every d_i is equally likely to be picked over the range $[0, d_U]$). The (τ, δ) values, however, change because the new $d_L = 0$. This is a change to a macrocharacterization of the problem. The new (τ, δ) values can be determined by substituting $d_L = 0$ into (2) and (3), yielding

$$\tau' = 1 - \frac{d_U}{(2n \cdot \overline{p}_s)}, \quad 0 \leq \tau \leq 1,$$
$$\delta' = \frac{d_U}{(n \cdot \overline{p}_s)}, \quad 0 \leq \delta \leq 1. \tag{6}$$

Incidentally, substituting these τ' and δ' into (4) gives $d_L \equiv 0$; that is, (6) and (4) are consistent.

One may wonder about the significance of the discard-and-replace approach in this context. We start with (τ, δ) value combinations that can cause intra-variable constraint violations and go through the computational process, ending up with the random data with the new (τ', δ') values given in (6). Why not start with the (τ', δ') values from the beginning without discard-and-replace? Yes, this approach should give the same result with much more efficient computing time! In our case, this means using the safe-zone algorithm rather than the discard-and-replace algorithm. We can extend this idea to some other applications. Summarizing, we propose the following.

Tip. Avoid the discard-and-replace method in general. More specifically, consider a two-step process.

Step 1. Whenever the discard-and-replace method is employed for random data generation, consider if there are any side effects on the problem characterizations such as (1) the data constraints, (2) the data distribution characteristics, and (3) the associated characteristic constraints. If any side effects exist, then move to Step 2.

Step 2. Determine which characterizations need to be preserved or changed. Consider whether the same data can be generated by adjusting some of the characterizations without employing the discard-and-replace method. This approach is likely much more efficient computationally than the discard-and-replace one.

6. Analysis of Intervariable Constraint Violations

For any i, $1 \le i \le n$, $d_i < p_i$ is an intervariable constraint violation. This is a microaspect concerned with individual jobs. We first address a macroaspect by considering feasible orderings of the four data parameters, p_L, p_U, d_L, and d_U [6]. The number of all permutations of the four parameters is $4! = 24$, but since $p_L \le p_U$ and $d_L \le d_U$, we are left with six possible permutations. Furthermore, we can assume $p_L \le d_L$ and $p_U \le d_U$. If $p_L > d_L$, then the shortest due date will be smaller than the shortest processing time, which violates the assumption and constraints. Similarly, $p_U > d_U$ violates the assumption and constraints. These two additional conditions reduce the feasible orderings to the following two:

$$p_L \le p_U \le d_L \le d_U, \tag{7}$$

$$p_L \le d_L \le p_U \le d_U. \tag{8}$$

Case 1 ($p_L \le p_U \le d_L \le d_U$). In this case, $d_i \ge p_i$ for every i; hence, inter-variable constraint violations never occur. The condition for which relation (7) holds in terms of τ and δ can be determined as follows:

$$p_U \le d_L = n\left(p_L + p_U\right)\frac{2\left(1-\tau\right)-\delta}{4}. \tag{9}$$

Hence, we have a condition for which an inter-variable constraint violation never occurs as

$$2\left(1-\tau\right)-\delta \ge \frac{4p_U}{n\left(p_L+p_U\right)}. \tag{10}$$

Case 2 ($p_L \le d_L \le p_U \le d_U$). We see that inter-variable constraint violations can occur in this case because the relation $d_L \le p_U$ can cause $d_i < p_i$ for some jobs. The following is a simple example, randomly generated following the typical procedure described in Section 4 (Figure 2).

Example 1. $n = 10$; $p_L = 1$, $p_U = 100$; hence $\overline{p}_s = 50.5$; $\tau = 0.8$, $\delta = 0.3$; hence $d_L = 25.2$, $d_U = 176.8$, and $\overline{d}_s = 101$ (see Table 1).

In this specific example, two jobs violate the inter-variable constraint. As we see below (13), for this particular parameter value combination, the probability of the violation is 0.187; that is, on average 1.87 jobs will have $d_i < p_i$.

When there are values with fractions (e.g., $d_L = -25.2$), for practical purposes they can be rounded to the nearest integer (e.g., $d_L = -25$). In the following, we first discuss theoretical analysis of violations and heuristic procedures for how to avoid them [6]. We will primarily use uniform distributions which are employed by most researchers in the tardiness problem.

We must satisfy $p_i \le d_i$ for every i, but $p_i > d_j$ is perfectly fine for different i and j. We can show that τ and δ must satisfy the following three conditions (two conditions in (8)) in terms of n, p_L, and p_U, to have the relation $p_L \le d_L \le p_U \le d_U$:

$$\frac{4p_U}{\{n\cdot\left(p_L+p_U\right)\}} \ge 2\left(1-\tau\right)-\delta \ge \frac{4p_L}{\{n\cdot\left(p_L+p_U\right)\}},$$
$$2\left(1-\tau\right)+\delta \ge \frac{4p_U}{\{n\cdot\left(p_L+p_U\right)\}}. \tag{11}$$

By adding the last two relations, we also have

$$\tau \le 1-\frac{1}{n}. \tag{12}$$

For example, if $n = 10$, then τ needs to be ≤ 0.9 to satisfy the last two relations.

Let $P(d_i < p_i)$ be the probability of $d_i < p_i$ for a specific job i. This probability is given by the following formula:

$$P\left(d_i < p_i\right) = \frac{\left(p_U-d_L\right)^3}{2\left(p_U-p_L\right)\left(d_U-d_L\right)\left(p_U-d_L+1\right)} \tag{13}$$

$$\approx \frac{\left(p_U-d_L\right)^2}{2\left(p_U-p_L\right)\left(d_U-d_L\right)} \quad \left(\text{when } p_U-d_L \gg 1\right). \tag{14}$$

The expected number of jobs for which $d_i < p_i$ for a problem of n jobs can be determined by $n \times P(d_i < p_i)$.

We can use $P(d_i < p_i)$ to compute other related probabilities. Let $q = P(d_i < p_i)$, for simplicity. The probability of not $d_i < p_i$, that is, $P(d_i \ge p_i)$, is $1-q$. Out of the total n jobs, the probability that exactly m jobs are $d_i < p_i$ is

$$P\ \left(m \text{ out of } n \text{ jobs are } d_i < p_i\right) = {}_nC_m\, q^m\left(1-q\right)^{n-m}, \tag{15}$$

where $q = P(d_i < p_i)$ and ${}_nC_m$ is the number of combinations for selecting m objects out of n objects at a time.

Table 1

Job no., i	1	2	3	4	5	6	7	8	9	10
Proc. time, p_i	92	41	10	21	37	86	85	66	25	37
Due date, d_i	90	95	116	64	151	171	66	97	49	93
Is $d_i < p_i$?	y						y			

Table 2

Job no., i	1	2	3	4	5	6	7	8	9	10
Proc. time, p_i	92	41	10	21	37	86	85	66	25	37
Due date, d_i	95	116	64	151	171	97	93	69	131	58

This probability distribution for $m = 0$ to n is the binomial distribution. In particular, the probability that at least one job has a violation is $1 - P$ ($m = 0$ jobs are $d_i < p_i$) $= 1-(1-q)^n$.

7. Removing Intervariable Constraint Violations by Simple Approaches

Without loss of generality, let us assume that we generate p_i's first, followed by d_i's, as suggested by previous research. Suppose that $p_1 = 15$ and $p_2 = 95$. If $d_1 = 45$ and $d_2 = 125$, there are no violations, but if these d_i's are swapped, a violation occurs. That is, we must satisfy $p_i \le d_i$ for every i, but we do not know the specific values of p_i's and d_i's until they are randomly generated. How to efficiently generate stochastic data that does not violate intervariable constraints is a challenging problem.

7.1. Discard-and-Replace Methods. There can be different versions of this classic approach depending on how one picks a new data element.

Discard-and-Replace with Next Random Valid Values. This is the most common version of discard-and-replace methods in general. For the tardiness problem, "if a negative processing time value is generated during the simulations, it is simply *ignored* and generated again" [25]. When $d_i < p_i$ we would continue to generate the next random d_i, until it satisfies $d_i \ge p_i$. A side effect of this approach is that it will skew the due date distribution to a higher range. The resulting (p_i, d_i) distribution will remain stochastic (even though d_i's skewed upward). The following is an illustrative example.

Example 2 (same as Example 1, except here we discard and replace with random valid values). $n = 10$; $p_L = 1$, $p_U = 100$; hence $\overline{p}_s = 50.5$; $\tau = 0.8$, $\delta = 0.3$; hence $d_L = 25.2$, $d_U = 176.8$, and $\overline{d}_s = 101$.

Actual values are $d_{L,a} = 58$, $\overline{p}_a = 104.5$, $\tau_a = 0.79$, and $\delta_a = 0.23$ (see Table 2).

This example is exactly the same as Example 1 except that d_i is replaced with the next d_i whenever a violation $d_i < p_i$ occurs. The pre-set due date average $\overline{d}_s$ is 101. The actual due date average $\overline{d}_a$ for Example 1 is 99.2, while that of Example 2 is 104.5, showing, expectedly, an overall increase of d_i's.

We can show that the overall expected $d_{L,e}$ is as follows:

$$d_{L,e} = \frac{\{p_U(p_U+1) + (d_L - 2p_L + 1)\}}{2(p_U - p_L + 1)}. \quad (16)$$

We note that $d_{L,e}$ is the expected value, not the actual value; the actual value is bounded from below by d_L. Similarly, the overall expected average due date, $\overline{d}_e$, is given by

$$\begin{aligned}\overline{d}_e = \{&p_U(p_U+1) + d_L(d_L - 2p_L + 1) \\ &+ 2d_U(p_U - p_L + 1)\} \\ &\times (4(p_U - p_L + 1))^{-1}.\end{aligned} \quad (17)$$

Since the new distributions are skewed, we are not precisely dealing with τ and δ as they were specified originally. But, it is most reasonable to define the effective τ_e and δ_e by substituting $\overline{d}_s$ and d_L in the original definitions of τ and δ in (2) and (3) with their effective counterparts τ_e and δ_e. That is,

$$\begin{aligned}\tau_e &= 1 - \frac{\overline{d}_e}{(n\cdot\overline{p}_s)} = 1 - \frac{(d_{L,e} + d_U)}{(2n\cdot\overline{p}_s)}, \\ \delta_e &= \frac{(d_U - d_{L,e})}{(n\cdot\overline{p}_s)}.\end{aligned} \quad (18)$$

In Example 2, these effective values are $d_{L,e} = 53.6$, $\overline{d}_e = 115.2$, $\tau_e = 0.772$, and $\delta_e = 0.244$. We note that (16)–(18) can also be expressed in terms of n, p_L, p_U, τ, and δ, by using (4) and (5).

As an alternative version of discard-and-replace, one can set $d_i = p_i$ when the generated values are such that $d_i < p_i$. This version of replacing $d_i < p_i$ with $d_i = p_i$ will have two shortcomings. First, the resulting due date distribution will be skewed toward a higher range, as in the previous version, since due dates with $d_i < p_i$ are replaced with higher values of p_i. Second, the resulting (p_i, d_i) distribution will be less stochastic than the previous version since all the jobs with replaced d_i will have exactly the same due dates as their processing times.

7.2. Augmented Probability Distributions. In the discard-and-replace method discussed in the previous subsection, we encountered violations of $d_i < p_i$, and replacing d_i with a larger d_i caused distortion of the underlying distribution. Here, we ask whether there are any guaranteed methods in which violations never occur. We can, for example, employ a d_i-generation function as follows:

$$d_i = p_i + h_i, \tag{19}$$

where h_i is some random function which is $h_i \geq 0$. In this way, not only d_i is guaranteed to be $\geq p_i$, but also lower d_i tends to be assigned to lower p_i and higher d_i to higher p_i [26]. Variations of (19) include $d_i = \alpha_i p_i$, where $\alpha_i \geq 1$, and a combination of (19) and $d_i = \alpha_i p_i$ as $d_i = \alpha_i p_i + h_i$.

We must, however, be cautious in employing such methods. For example, if we select uniform distributions for p_i and h_i in (19), d_i will not be uniform any more (its probability density function will be trapezoidal). How to reasonably define τ and δ in such a situation is another question. In short, we need careful consideration before employing these methods.

8. A Neighborhood Expanding Data-Interchanging Heuristic

8.1. General Description. The method discussed in this section is a heuristic for reducing the impact of constraint violations on the generated data. The basic idea of this method should be applicable to many types of problems.

General Idea. We study randomly generating values of variable x_i. A set of these values may contain n values for x_i, $i = 1, n$. Further, we can extend the size of the data as a group of multiple sets and a group of groups of sets and so forth. We consider the neighborhood of these data. The most local neighborhood of x_i can be the neighboring data elements of x_i as, for example, x_{i-1}, x_i, x_{i+1}. When the neighborhood coverage of data elements is extended to the entire set or a group of sets and so forth, the scope of the neighborhood will be more "global." We perform data interchanging starting from the most local neighborhood level to resolve violations. If they are not resolved, we extend the neighborhood toward a more global level, until all violations are resolved (Figure 3). Hence, we use the following steps.

(i) Item-by-item swapping at the most local level: when a violation is found for a specific data item, find another data item such that when these two data items are swapped the violation is resolved.

(ii) Intraset swapping: when the above item-by-item swapping does not work, consider the entire data set in which the data item is an element. Swap any data items (elements) within the set so that violations can be removed.

(iii) Interset swapping: when the above intra-set swapping does not work, include neighboring data sets to the above data set, and try to resolve violations by taking into account all of the data items in all the data sets under consideration. Start with an adjacent data set, expanding toward the entire collection of data sets until violations are resolved.

(iv) If either the inter-set swapping does not work or there are no other data sets to include, discard and replace some data item(s) or data set(s). Hopefully, the chances of performing this last step are very small.

8.2. An Illustration Using the Total Tardiness Problem

8.2.1. Preliminaries. The heuristic here is a special case of the above basic idea of the neighborhood expanding data-interchanging method, where "data item" and "data set" are replaced by "job" and "problem," respectively. In the implementation of the heuristic, we skip the most local, item-by-item swapping, described in the above general outline of the method, since it does not appear particularly effective for the inter-variable violation problem. For some other types of violations, this step may be useful. Before describing the heuristic, we introduce a term and a theorem.

Definition 3. Processing times and due dates of a problem are *pairable* if there is at least one permutation of p_i's and at least one permutation of d_i's that satisfy $p_i \leq d_i$ for every $i = 1$ to n; in this case, we say that the problem is pairable. In other words, if a problem is pairable, we can make an invalid problem internally valid by rearranging p_i's and d_i's; otherwise, it is impossible to make the problem internally valid, no matter how we shuffle p_i's and d_i's.

Theorem 4. *Sort all p_i's and d_i's in a problem, so that $p_1 \leq p_2 \leq \cdots$ and $d_1 \leq d_2 \leq \cdots$. A necessary and sufficient condition for a problem to be pairable is $p_i \leq d_i$ for every $i = 1$ to n.*

Proof. If $p_i \leq d_i$ for every $i = 1$ to n for sorted sequences of p_i's and d_i's, the set of the sorted sequences is an internally valid problem. Therefore, we can make at least one (and possibly many more) internally valid problem(s). Hence, the condition is sufficient. Conversely, suppose that $p_i > d_i$ for some i. Then this p_i must be paired with another d_j, $j > i$. This leaves fewer d's than p's for pairing (the pigeon-hole principle), which means that pairing all the remaining p's and d's is impossible. Thus, the condition is necessary. □

Data-Interchanging Heuristic

```
Step 1 (intraproblem swapping)
Sort all $p_i$'s and $d_i$'s (i.e., $p_1 \leq p_2 \leq \cdots$ and $d_1 \leq d_2 \leq \cdots$).
For $i = n$ step $-1$ down to 1 do
    For $p_i$, find the smallest $d_{\min}$ such that $d_{\min} \geq p_i$.
    Randomly select $d_j$ in $d_{\min}$ to $d_i$.
    Pair $(p_i, d_j)$ and output it as a valid pair.
    Rearrange $d_k$ by $d_k \leftarrow d_{k+1}$ for $k = j$ to $i - 1$.
    Enddo.
```

Restore the original (presorting) order of p_i's for the generated n pairs of (p_i, d_i); that is, p_i's in new pairs of (p_i, d_i) appear in the same order as originally

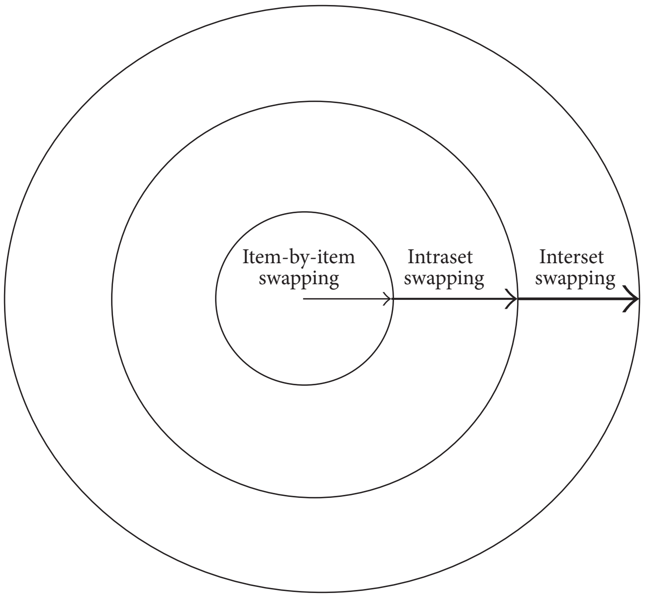


Figure 3: A schematic representation of the neighborhood expanding data-interchanging heuristic. As the neighborhood expands, the probability of finding pairable swaps increases (indicated by the thickness of the arrow).

generated at random. (so that p_i's are not in any particular sequence such as being sorted).

Step 2 (interproblem swapping). When Step 1 does not work, include gradually increasing number of neighboring problems to the above problem. We may start with combining two problems, the above problem and the succeeding problem, having a total of $2n$ jobs, and apply Step 1 to this $2n$-jobs problem. When $2n$ jobs are successfully paired, restore the original (pre-sorting) orders of p_i's in each problem. If this does not work, include three problems and so on, until the entire problem set is used. Apply Step 1 to each kn-jobs problem, where $k = 2$ to number of problems.

Step 3. If Step 2 does not work, or there is no other problem in Step 2, discard and replace some d_i's, jobs, or problems. Of course, such discard-and-replace process will distort the original data characteristics, like other methods, as discussed previously. Our experiments, as discussed in Section 10, show that the chances of performing Step 3 are extremely small.

8.2.2. Additional Notes on the Data-Interchanging Heuristic

Item-by-Item Interchanging. In the above algorithm, although we skipped the most local data interchanging described in the general method, we briefly discuss it here to illustrate how the concept can be applied.

Job-by-Job Swapping for the Total Tardiness Problem. When a violation is found for a specific job i, find another job j such that $d_i \geq p_j$ and $d_j \geq p_i$, and swap d_i and d_j (or p_i and p_j). A choice of swapping (d_i and d_j) or (p_i and p_j) can also be made

Table 3

Job no., i	1	2	3
Processing time, p_i	10	8	5
Due date, d_i	7	11	9

randomly to avoid, for example, larger d_i tending to appear earlier.

We note that this procedure does not accomplish the same result as Step 1 in the heuristic. Consider the following example.

Example 5. (see Table 3) Since job 1 is a violation, we search for $d_i \geq p_j$ and $d_j \geq p_i$, but this search fails even though the problem is pairable. We may call such a situation a "*three-way deadlock*." There can be extensions of this, as *four-way*, . . ., *n-way deadlocks*.

Effect of Step 2 on Data Characteristics. In Step 2 of the heuristic, we combine two, three, and more problems as needed to come up with valid sequences of p_i's and d_i's. We might wonder whether in effect this process changes the problem size from n to $2n$, $3n$, and so on. If so, the process would affect the values of τ and δ, since they depend on the problem size. However, this is not the case. For pairing purposes, we scramble p_i's and d_i's of multiple problems. But after pairing is complete, the original order of p_i's in each problem is restored and the problem size remains the same as n.

9. Experimental Results

The data-interchanging heuristic was implemented and tested with $n = 8, 16, 32, 64$, and 128; $p_L = 1$, $p_U = 100$;

Table 4: Number of violations in generated data and after each of the steps of the proposed heuristic.

n	In generated data	After Step 1	After Step 2	After Step 3
8	365	91	25	0
16	131	86	0	0
32	14	14	0	0
64	0	0	0	0
128	0	0	0	0

hence $\overline{p}_s = 50.5$; $\tau = 0.8$ and $\delta = 0.3$. d_L and d_U can be determined by using (4) and (5), respectively, as $d_L = 2.5n$ and $d_U = 17.7n$. For each problem size n, data for $w = 100$ problems of the given size were generated and the violations in the generated data as well as after each of the three steps of the proposed algorithm were recorded (see Table 4). For example, generating 100 problems of size 8 each, the generated data had a total of 365 violations. Using Step 1 of the proposed heuristic, 91 violations remained—a 75% reduction in the number of violations in the generated data. When Step 2 of the heuristic is used, only 25 violations remained—an impressive 93% reduction in the number of violations in the generated data. When Step 3 of the algorithm is used, there was a 100% reduction in the violations in the generated data. Similar performance is observed for other values of n. For example, for $n = 16$ and 32, a 100% reduction in the violations was achieved after Step 2 and no further steps were required. While there is no guarantee that such a reduction is expected for every data in general, the result is indicative for effectiveness of the algorithm. Notice that as n increases, the probability of having a violation decreases, because p_L and p_U remain constant while d_L and d_U increase with n. This is why no violations were encountered for $n = 64$ and 132.

10. Conclusions

In this paper, we discussed how implicit constraints were overlooked in some previous practices for generating data to simulate the total tardiness problem. This may not be an isolated case and may extend to other practical approaches involving generation of random data under constraints. When there are possible data violations, analytical approaches such as the one demonstrated in this paper (e.g., Section 4) should be helpful. Heuristics, such as the local-and-global data-interchanging heuristic discussed in this paper, may be used, depending on the nature of the application problems and the types of data violations.

The following are some general guidelines for generating stochastic data under constraints.

(1) Carefully examine the problem to see whether there are certain constraints that must be satisfied (e.g., due dates must be non-negative and each due date must be not less than the processing time in the total tardiness problem).

(2) Check the procedure of generating stochastic data to determine whether it possibly yields invalid data which is in violation of a constraint. (In the total tardiness problem, by glancing at (3), we see that d_L can be negative for certain values of τ and δ, thus possibly yielding negative due dates. Also, by looking at (4), we see that d_U can be less than p_L, thus possibly generating a job whose due date is less than its processing time.)

We need to pay special attention when "characteristic parameters" (e.g., τ and δ) are introduced. These characteristic parameters are important metrics for the problem to be solved, but they are often abstract and only indirectly represent the original characteristics of the data. This may lead to a common error of focusing primarily on the characteristic parameters and forgetting the nature of the original data.

(3) If there may be possible violations, we can theoretically analyze the conditions for which the violations occur. For certain cases, this analysis may lead to a simple revised procedure that guarantees no violations, or a set of parameter values that avoid violations.

(4) Whenever the discard-and-replace method is employed, we must consider the resulting effect in terms of the problem characterizations such as (1) the constraints, (2) the data distribution, and (3) the associated characteristics. Determine which characterizations need to be preserved or changed. Consider whether the same data can be generated by adjusting some of the characterizations without employing the time-consuming discard-and-replace method. This approach is likely much more efficient computationally than discard-and-replace.

(5) For certain problems, Steps (3) and (4) above may not result in a sufficient method. That is, there is no simple procedure that guarantees no violations, or a set of parameter values that completely avoids violations. In such cases, one may attempt to develop a new data generation procedure that satisfies validity criteria such as the following.

  (a) No invalid data has been generated.

  (b) The associated characteristics of data generated are as close as possible to the intended original data (e.g., average and expected values of certain entities), unless the original characteristics themselves are violations.

  (c) The procedure is computationally simple and efficient.

Developing such a procedure satisfying all the above criteria, however, may not be trivial. Often we may find conflicting trade-offs among the various criteria. Usually criterion (a) is the highest priority. Unless we produce massive data, criteria (c) may not be a high priority in comparison with the other criteria, due to the high speed of today's computers. In certain cases, heuristics that are not perfect but practically good enough methods may be used.

Further studies can include the following.

(1) Other problems: we employed the total tardiness problem and the two simple examples of Section 2 to illustrate the core of this article, that is, constraint consistency among different parameter sets. Other problems in different domains for various application types can be considered.

(2) Nonuniform distributions of random variables: in this article, we primarily focused on uniform distributions since they have been most commonly employed in the total tardiness problem. However, other distributions can also be considered, especially for other problems.

(3) Higher number of data and characteristic parameters: for the total tardiness problem, the number of data parameters is two and the number of characteristic parameters is also two. For the two simple examples discussed in Section 2, the number of characteristic parameter is one, and there are several data parameters. We can consider higher number (e.g., three and three, or generally M and N) for these parameters.

(4) More general mapping between data and characteristic parameters: for the total tardiness problem, the mapping is one to one. Other cases, such as many to one, may be considered.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

The authors would like to thank their graduate student Todd Posius for his help on the numerical experiment.

References

[1] T. Munakata, *Fundamentals of the New Artificial Intelligence: Neural, Evolutionary, Fuzzy and More*, Springer, London, UK, 2nd edition, 2008.

[2] J. Du and J. Leung, "Minimizing total tardiness on one process is NP-hard," *Mathematics of Operations Research*, vol. 3, pp. 483–495, 1990.

[3] C. Koulamas, "The single-machine total tardiness scheduling problem: review and extensions," *European Journal of Operational Research*, vol. 202, no. 1, pp. 1–7, 2010.

[4] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, McGraw-Hill, New York, NY, USA, 8th edition, 2004.

[5] L. D. Landau and E. M. Lifshitz, *Fluid Mechanics*, vol. 6 of *Course of Theoretical Physics*, Butterworth-Heinemann, Oxford, UK, 2nd edition, 1987.

[6] T. Munakata and A. Fadlalla, "Constraint isomorphism and the generation of stochastic data," *IIE Transactions*, vol. 38, no. 5, pp. 437–444, 2006.

[7] L. Wilkerson and J. Irwin, "An improved method for scheduling independent tasks," *AIIE Transactions*, vol. 3, no. 3, pp. 239–245, 1971.

[8] V. Srinivasan, "A hybrid algorithm for the one machine sequencing problem to minimize total tardiness," *Naval Research Logistics Quarterly*, vol. 18, pp. 317–327, 1971.

[9] A. H. G. Rinnooy Kan, B. J. Lageweg, and J. K. Lenstra, "Minimizing total costs in one machine scheduling," *Operations Research*, vol. 23, no. 5, pp. 908–927, 1975.

[10] M. L. Fisher, "A dual algorithm for the one-machine scheduling problem," *Mathematical Programming*, vol. 11, no. 1, pp. 229–251, 1976.

[11] C. L. Potts and L. N. Van Wassenhove, "Single machine sequencing heuristics," *IIE Transactions*, vol. 23, pp. 346–354, 1991.

[12] H. A. J. Crauwels, C. N. Potts, and L. N. Van Wassenhove, "Local search heuristics for the single machine total weighted tardiness scheduling problem," *INFORMS Journal on Computing*, vol. 10, no. 3, pp. 341–350, 1998.

[13] R. Congram, C. N. Potts, and S. L. van de Velde, "An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem," *INFORMS Journal on Computing*, vol. 14, pp. 52–67, 2002.

[14] L.-P. Bigras, M. Gamache, and G. Savard, "Time-indexed formulations and the total weighted tardiness problem," *INFORMS Journal on Computing*, vol. 20, no. 1, pp. 133–142, 2008.

[15] C. N. Potts and L. N. Van Wassenhove, "A decomposition algorithm for the single machine total tardiness problem," *Operations Research Letters*, vol. 1, no. 5, pp. 177–181, 1982.

[16] C. Koulamas, "The total tardiness problem: review and extensions," *Operations Research*, vol. 42, no. 6, pp. 1025–1041, 1994.

[17] Y.-D. Kim, "Minimizing total tardiness in permutation flowshops," *European Journal of Operational Research*, vol. 85, no. 3, pp. 541–555, 1995.

[18] W. Szwarc and S. K. Mukhopadhyay, "Decomposition of the single machine total tardiness problem," *Operations Research Letters*, vol. 19, no. 5, pp. 243–250, 1996.

[19] K. C. Tan and R. Narasimhan, "Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach," *Omega*, vol. 25, no. 6, pp. 619–634, 1997.

[20] N. G. Hall and M. E. Posner, "Generating experimental data for computational testing with machine scheduling applications," *Operations Research*, vol. 49, no. 6, pp. 854–865, 2001.

[21] T. Munakata and A. Fadlalla, "Constraint isomorphism and correction algorithms for violations," *PAMM*, vol. 7, no. 1, pp. 2010031–2010032, 2007.

[22] D. P. Ronconi and L. R. S. Henriques, "Some heuristic algorithms for total tardiness minimization in a flowshop with blocking," *Omega*, vol. 37, no. 2, pp. 272–281, 2009.

[23] W.-J. Chen, "Minimizing number of tardy jobs on a single machine subject to periodic maintenance," *Omega*, vol. 37, no. 3, pp. 591–599, 2009.

[24] C. Koulamas, "Single-machine scheduling with time windows and earliness/tardiness penalties," *European Journal of Operational Research*, vol. 91, no. 1, pp. 190–202, 1996.

[25] S. Goren, "Sabuncuoglu. Generating robust and stable schedules for a single machine environment under random machine breakdowns and processing time variability," http://www.ie.bilkent.edu.tr/technicalpapers/Goren_Sabuncuoglu.pdf.

[26] A. Agnetis, A. Alfieri, and G. Nicosia, "Single-machine scheduling problems with generalized preemption," *INFORMS Journal on Computing*, vol. 21, no. 1, pp. 1–12, 2009.