July 1991

Fast Algorithms for Generating Discrete Random Variates With Changing Distributions

Sanguthevar Rajasekaran
University of Pennsylvania

Keith W. Ross
University of Pennsylvania

University of Pennsylvania Department of Comptuer and Information Science Tecchnical Report No. MS-CIS-91-52.

Fast Algorithms for Generating Discrete Random Variates With Changing Distributions

Abstract

One of the most fundamental and frequently used operations in the process of simulating a stochastic discrete event system is the generation of a nonuniform discrete random variate. The simplest form of this operation can be stated as follows: Generate a random variable X which is distributed over the integers 1,2,...,n such that P(X = i) = p_i. A more difficult problem is to generate X when the p_i's change with time. For this case, there is a well-known algorithm which takes O(log n) time to generate each variate. Recently Fox [4] presented an algorithm that takes an expected o(log n) time to generate each variate under assumptions restricting the way the p_i's can change.

In this paper we present algorithm for discrete random variate generation that take an expected O(1) time to generate each variate. Furthermore, our assumptions on how the p_i's change are less restrictive than those of Fox. The algorithms are quite simple and can be fine-tuned to suit a wide variety of application. The application to the simulation of queueing networks is discussed in some detail.

Keywords

simulation, queueing networks, randomized algorithms

Comments

University of Pennsylvania Department of Comptuer and Information Science Tecchnical Report No. MS-CIS-91-52.

Fast Algorithms For Generating Discrete Random Variates With Changing Distributions

MS-CIS-91-52
GRASP LAB 271

Sanguthevar Rajasekaran
Keith Ross

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

July 1991

Fast Algorithms for Generating Discrete Random Variates with Changing Distributions

Sanguthevar Rajasekaran
Dept. of Computer and Information Science
Univ. of Pennsylvania, Philadelphia, PA 19104

Keith W. Ross[1]
Dept. of Systems
Univ. of Pennsylvania, Philadelphia, PA 19104

June 25, 1991

ABSTRACT

One of the most fundamental and frequently used operations in the process of simulating a stochastic discrete event system is the generation of a nonuniform discrete random variate. The simplest form of this operation can be stated as follows: Generate a random variable X which is distributed over the integers $1, 2, \ldots, n$ such that $P(X = i) = p_i$. A more difficult problem is to generate X when the p_i's change with time. For this case, there is a well-known algorithm which takes $O(\log n)$ time to generate each variate. Recently Fox [4] presented an algorithm that takes an expected $o(\log n)$ time to generate each variate under assumptions restricting the way the p_i's can change.

In this paper we present algorithms for discrete random variate generation that take an expected $O(1)$ time to generate each variate. Furthermore, our assumptions on how the p_i's change are less restrictive than those of Fox. The algorithms are quite simple and can be fine-tuned to suit a wide variety of applications. The application to the simulation of queueing networks is discussed in some detail.

Key Words: Simulation, Queueing Networks, Randomized Algorithms.

[1]The work of this author has been partially supported by NSF grant DDM 5-22863.

1 Introduction

The problem of generating a nonuniform discrete random variate is fundamental to simulation of any discrete event stochastic system. The simplest version of this problem is to generate a random variable X which takes on values in $\{1, 2, \ldots, n\}$ such that $P(X = i) = p_i$, given that the p_i's do not change with time. The well-known 'alias algorithm' [2] [pp. 147-148] takes $O(n)$ preprocessing time, after which it can generate a variate in $O(1)$ time. In this paper we are interested in developing efficient algorithms for the case of the p_i's changing with time. Such a procedure is desirable in the simulation of multi-dimensional Markov processes, such as those used to model queueing and telephone networks.

A precise definition of the problem is as follows. Suppose we are given n real numbers $a_1, a_2, \ldots, a_n$. The problem is to generate a random variable X distributed over $\{1, \ldots, n\}$ such that $P(X = i) = p_i$, where $p_i := a_i/(a_1 + \cdots + a_n)$. Call a_i the *ith rate of* X. In the static version of this problem, the a_i's do not change with time. In the dynamic version, zero or more of the a_i's change after each call to the generation of X.

In the context of simulation of discrete-event stochastic systems, the future-event schedule can be organized as a heap [2] so that variates with changing distributions can be generated with $O(n)$ preprocessing time and $O(\log n)$ time per variate. Fox [4] recently showed that a random variate can be generated in an expected $o(\log n)$ time per variate, with preprocessing time being $O(n)$ (see also [5] for related work). However, Fox assumes that the a_i's change in a 'restricted' manner. Roughly speaking, Fox requires that the a_i's deviate infrequently from a vector of nominal values. This assumption is applicable to queueing networks operating under globally heavy traffic, in which case the service rates at the various queues rarely differ from their maximum values.

In this paper we present algorithms for dynamic random variate generation that run in an expected $O(1)$ time (after $O(n)$ time of preprocessing). The data structures and algorithms we propose are quite simple, in contrast to those for the future-event schedule or for the method of Fox. Updating the data structure to reflect the change in one a_i takes just one unit of time. Even though we also assume that the a_i's change in a restricted manner, our restrictions are substantially less severe than those of Fox. Specifically, we assume that:

(A1) The a_i's are uniformly bounded above, i.e., there is an $\bar{a}$ such that $a_i \leq \bar{a}$ for all i.

(A2) There exist lower bounds $\underline{a}_i \geq 0$ for each a_i (i.e., $a_i \geq \underline{a}_i$ for all i) and an $\underline{\alpha} > 0$ such that

$$\frac{1}{n} \sum_{i=1}^{n} \underline{a}_i \geq \underline{\alpha}.$$

(In words, the average of the lower bounds is bounded from below by a positive number.)

(A3) $h := \underline{\alpha}/\bar{a}$ is a constant (independent of n).

(A4) The number of a_i's that change after each call to the generation of X is $O(1)$.

In the context of a queueing network, our assumptions not only hold for the case of globally heavy traffic, but also hold when the traffic is substantially lighter. Moreover in many applications, both $\underline{\alpha}$ and $\bar{a}$ are independent of n, thus naturally satisfying (A3).

In the paragraph that follows we collect a few results from the literature that will be of use in this paper. In Section 2 we give our most basic algorithm for generating variates in $O(1)$ expected time. However, the constant associated with this algorithm can be quite large for many interesting applications. In Section 3 we develop a generalized version of the algorithm that can significantly reduce the magnitude of the constant. We also derive an explicit expression characterizing the magnitude of the constant. In Section 4 we discuss the application of the generalized algorithm to the simulation of queueing networks. We obtain explicit expressions for the constant associated with the algorithm in terms of the service rates and utilizations at each of the queues. In Section 5 we discuss three modifications of the generalized algorithm which have the potential of offering additional improvements in performance.

1.1 Some Preliminaries

Let $Y = \sum_{i=1}^{m} \eta_i$ where η_i's are independent and identically distributed geometric random variables with parameter p. (Y can be thought of as the number of times a coin has to be flipped before a head appears for the mth time, p being the probability that a head appears in a single flip).

Chernoff bounds (see e.g., [6] [pp. 388-393]) can be used to obtain tight bounds on probabilities in the tail ends of Y. In particular we can show the following:

$$\mathrm{P}\left[Y \geq (1+\epsilon)\frac{m}{p}\right] \leq \exp\left(\frac{-\epsilon^2 m}{1-p}\right),$$

for any fixed $\epsilon << 1$ (see [10]).

Also, if η is a binomial random variable $B(n,p)$, then the probabilities in the tail ends of η can be bounded using Chernoff bounds. The following facts can be proven (e.g., see [1] or [3]):

$$\mathrm{P}\left[\eta \geq (1+\epsilon)np\right] \leq e^{-\epsilon^2 np/2}, \text{ and}$$

$$\mathrm{P}\left[\eta \leq (1-\epsilon)np\right] \leq e^{-\epsilon^2 np/3},$$

for any $0 < \epsilon < 1$. We will make use of these bounds in our analysis.

Just like the $O()$ function is used to specify the asymptotic resource bounds of deterministic algorithms, $\tilde{O}()$ is used to specify resource (like time, space etc.) bounds of randomized algorithms. $\tilde{O}()$ is defined as follows.

Definition. [10] We say a function $f(.)$ is $\tilde{O}(g(.))$ if there exists a constant c such that $f(n) \leq c\alpha g(n)$ with probability $\geq (1 - n^{-\alpha})$ on any input of size n, for any $\alpha > 0$ and any $n \geq n_0$ (n_0 being a constant).

2 A Simple Algorithm

In this section we present a simple algorithm for generating variates with changing distributions. Suppose we need to generate a variate for X, where $P(X = i) = a_i/(a_1 + \cdots + a_n)$. Our algorithm makes use of n buckets, i.e., an array B of size n. Each bucket has a 'capacity' of 1. For each bucket i we initially set $B[i] = a_i/\bar{a}$. This completes preprocessing. Clearly this processing can be completed in $O(n)$ time. Notice that the initial contents of each bucket does not exceed unity. Here is an algorithm to generate a variate for X: (Here after let U stand for the uniform random variable in the interval $(0, 1)$.)

Algorithm **Generate1**

Step 1. Generate U, and compute $I = \lceil nU \rceil$.
Step 2. Let $R = I - nU$.
Step 3. If $R \leq B[I]$, accept and output I and quit;
Else go to Step 1.

Observe that Generate1 picks a bucket at random (uniformly). If I is the bucket chosen, then I is output with probability $B[I]$. (Notice that R is a uniform variable in the interval $(0, 1)$ and is independent of I.)

The algorithm Generate1 can be thought of as the rejection algorithm (e.g., see [2] [pp. 140-141]) for generating discrete random variates with fixed distributions. (The rejection algorithm is typically discussed in the context of continuous random variables.) The output of the algorithm Generate1 has the same distribution as that of X.

The key observation we make here is that the rejection algorithm can also be used to generate variates with changing distributions. If an a_i changes to a_i', then we simply reset

$B[i] = a_i'/\bar{a}$. The procedure is valid since, by Assumption (A1), the contents of each bucket never exceeds unity.

In order to characterize the performance of Generate1, let "acceptance" be the event that $R \leq B[I]$ the first time Step 3 is reached. We refer to $P(\text{"acceptance"})$ as the *acceptance probability.* Note that the expected number of uniform random variables U needed to generate a variate for X by the algorithm Generate1 is equal to the reciprocal of the acceptance probability. Let $\alpha := \frac{1}{n}\sum_{i=1}^{n} a_i$. Then, we have the following

Theorem 2.1 *The acceptance probability for Generate1 is given by*

$$P(\text{"acceptance"}) = \frac{\alpha}{\bar{a}}. \tag{1}$$

Furthermore, expected $O(1)$ time per variate is guaranteed by the algorithm Generate1 in order to generate variates for X with changing distributions.

Proof: We have

$$\begin{aligned} P(\text{"acceptance"}) &= P(R \leq B[I]) \\ &= \sum_{i=1}^{n} \frac{1}{n} P(R \leq B[i]) \\ &= \sum_{i=1}^{n} \frac{1}{n}\frac{a_i}{\bar{a}} = \frac{\alpha}{\bar{a}}, \end{aligned}$$

establishing the first claim. It follows from (1) and assumption (A2) that $P(\text{"acceptance"}) \geq \underline{\alpha}/\bar{a}$. This fact along with assumptions (A3) and (A4) establishes the second claim. □

Even though Generate1 (and Generate2 given in the next section) only has an *expected* $O(1)$ time bound, if it is invoked many times (which is typically the case in any stochastic discrete event simulation), the total amount of time spent will deviate from the expected value only by a small amount with high probability, as proven in the following theorems.

Theorem 2.2 *Let h be a lower bound on the acceptance probability of Generate2. If Generate2 is called m times, then the total number of U's generated by Generate2 is no more than $(1+\epsilon)\frac{m}{h}$ with probability $\geq 1 - e^{[-\epsilon^2 m/(1-h)]}$ for any fixed $\epsilon << 1$.*

Proof. Follows immediately from the remarks made in Section 1.1, and the observation that the number of U's generated by Generate2 to obtain a variate for X is upper bounded by a geometric random variable with parameter h. □

Corollary 2.1 *If $m = \omega(\log n)$, the number of U's generated is no more than $(1+\epsilon)\frac{m}{h}$ with probability $\geq \left(1 - \frac{1}{n^\alpha}\right)$ for any constant $\epsilon << 1$ and any fixed $\alpha \geq 1$. Thus the number of U's generated is $\tilde{O}(\frac{m}{h})$.*

2.1 Probabilistic Structure for the Rates

The assumption (A2) can be made weaker if we assume that a probabilistic structure is available for the a_i's. Consider generating a sequence of variates $X(1), X(2), \ldots$ from Generate1. Let $\mathbf{A}(s) := (A_1(s), \ldots, A_n(s))$ be the vector of rates used to generate $X(s)$. Let us assume that $\mathbf{A}(1), \mathbf{A}(2), \ldots$ are random variables and are supported on a known probability space. We shall continue to make the assumption (A1) so that $0 \leq A_i(s) \leq \bar{a}$ for all i and s. But we now replace the assumption (A2) with the following assumption:

(A2′) There is an $\underline{\alpha} > 0$ such that

$$\frac{1}{n}\sum_{i=1}^{n} E[A_i(s)] \geq \underline{\alpha} \qquad \text{for all } s = 1, 2, \ldots$$

Note that this assumption allows for the possibility of $\underline{a}_i = 0$ for all i.

Let "acceptance at time s" be the event that $R \leq B(I)$ the first time Step 3 is reached when generating X_s. From Theorem 2.1 we have

$$P(\text{"acceptance at time s"}|\mathbf{A}(s)) = \frac{\frac{1}{n}\sum_{i=1}^{n} A_i(s)}{\bar{a}}.$$

Taking the expectation of both sides of this expression and mimicking the proof of Theorem 2.1 gives the following result.

Theorem 2.3 *We have*

$$P(\text{ "acceptance at time } s\text{"}) \;=\; \frac{\frac{1}{n}\sum_{i=1}^{n} E[A_i(s)]}{\bar{a}}. \tag{2}$$

Furthermore, under assumptions (A1), (A2′), (A3), and (A4), expected $O(1)$ time per variate is ensured by the algorithm Generate1 in order to generate variates for X with changing distributions.

3 Increasing the Acceptance Probability

Although the algorithm Generate1 produces variates in expected $O(1)$ time, its performance could be poor in applications for which the average of the a_i's is significantly less than the uniform upper bound for the a_i's. However, in many applications (see Section 4 where queueing networks are discussed) we may be able to specify an upper bound, $\bar{a}_i$, for each of

the a_i's individually, and we may also have $\bar{a}_i << \bar{a}$ for many of the a_i's. In this section we give an algorithm that exploits this special structure.

In Generate1 one bucket is assigned to each outcome i, $i = 1, \ldots, n$. The idea behind our modified algorithm is to assign one or more buckets to any outcome i. The number of buckets assigned to i is proportional to the upper bound of the ith rate. Specifically, let d be a positive number and let the number of buckets assigned to outcome i, denoted by l_i, be given by

$$l_i = \left\lceil \frac{\bar{a}_i}{d} \right\rceil, \qquad i = 1, \ldots, n.$$

Also let $l = l_1 + \cdots + l_n$ be the total number of buckets and let $\bar{\alpha}$ be the average of the upper bounds, i.e., $\bar{\alpha} := 1/n \sum_{i=1}^{n} \bar{a}_i$. Note that we have the following bounds on the total number of buckets:

$$n\frac{\bar{\alpha}}{d} \leq l \leq n(\frac{\bar{\alpha}}{d} + 1).$$

Thus if $d = \bar{\alpha}$, then $n \leq l \leq 2n$.

In the modified algorithm, the number of buckets assigned to any i is held fixed even when the a_i's change. Once having chosen at random one of the l buckets, if that bucket is assigned to i, then we will output i with a certain probability. This probability, however, will change as the a_i's change.

To be more precise, we make use of two arrays B and C of size l and n, respectively. For C we set

$$C[i] = \frac{a_i}{dl_i}, \qquad i = 1, \ldots, n.$$

We set $B[j]$ equal to the outcome i assigned to bucket j (for $1 \leq j \leq l$).

Preprocessing

Step 1. For each outcome i compute l_i.
Step 2. Compute the prefix sums of $(l_1, l_2, \ldots, l_n)$. Let the sums be $(m_1, m_2, \ldots, m_n)$.
Step 3. Initialize B as follows: Fill cells 1 through m_1 with 1, cells $m_1 + 1$ through m_2 with 2, and so on.
Step 4. Set $C[i] = \frac{a_i}{dl_i}$ for $1 \leq i \leq n$.

Next we present the procedure for generating a variate for X.

Algorithm **Generate2**

Step 1. Generate U and compute $J = \lceil lU \rceil$.
Step 2. Let $R = J - lU$.
Step 3. Let $I = B[J]$. If $R \leq C[I]$, output I and quit; Else go to Step 1.

After having generated a variate for X, if an a_i changes to a_i' we simply reset $C[i] = a_i'/dl_i$. Note that, by assumption (A1), $C[i]$ never exceeds unity. Also note that we can always set $\bar{a}_i = \bar{a}$, $i = 1, \ldots, n$, and $d = \bar{a}$, in which case Generate2 reduces to Generate1. In a manner entirely analogous to that for Generate1, we can also define the event "acceptance" for the algorithm Generate2.

Theorem 3.1 *The output of Generate2 has the same distribution as that of X. Moreover, for Generate2 we have*

$$P(\text{"acceptance"}) = \frac{\sum_{i=1}^{n} \frac{a_i}{d}}{\sum_{i=i}^{n} \lceil \frac{\bar{a}_i}{d} \rceil}. \tag{3}$$

Proof: We have

$$P(\text{"acceptance"}) = \sum_{i=1}^{n} \frac{l_i}{l} C[i],$$

from which (3) follows. Also note that

$$P(I = i, \text{"acceptance"}) = \frac{l_i}{l} C[i].$$

The two above expressions together give $P(I = i | \text{"acceptance"}) = a_i/(a_1 + \cdots + a_n)$, which completes the proof. □

Theorem 3.1 directly gives the following result.

Corollary 3.1 *We have the following bounds on the acceptance probability for Generate2:*

$$\frac{\alpha}{\bar{\alpha} + d} \leq P(\text{"acceptance"}) \leq \frac{\alpha}{\bar{\alpha}}.$$

Furthermore, if d is a divisor of $\bar{a}_i$, $i = 1, \ldots, n$, then $P(\text{"acceptance"}) = \alpha/\bar{\alpha}$.

Since $\bar{\alpha}$ can be significantly less than $\bar{a}$, the algorithm Generate2 can give a much higher acceptance probability as compared with Generate1. The cost of Generate2 is the additional memory that it requires. Corollary 3.1 implies the following.

Corollary 3.2 *Suppose the rates of X never change. Then the acceptance probability for Generate2 can be made arbitrarily close to unity by decreasing d. Furthermore, if d is a divisor of $\bar{a}_i$, $i = 1, \ldots, n$, then the acceptance probability equals unity.*

Note that under the conditions of Corollary 3.2, with d being a divisor of $\bar{a}_i$, $i = 1, \ldots, n$, Generate2 can be simplified because $C[i] = 1$ for all i. In particular, each variate for X would only require the generation of one uniform variate, one multiplication, one 'upper floor' operation, and one memory reference. This compares favorably with the Alias algorithm, which requires one of each of the above operations (sometimes two memory references) in addition to a subtraction and a comparison. We also note that this simplified algorithm is similar, but not identical, to the algorithm given in [2] [pp. 141-143].

Before concluding this section we remark that if the rates can be viewed as random variables, as in Section 2.1, then the acceptance probability for Generate2 becomes:

$$P(\text{“acceptance at time s”}) = \frac{\sum_{i=1}^{n} \frac{E[A_i(s)]}{d}}{\sum_{i=i}^{n} \lceil \frac{\bar{a}_i}{d} \rceil}. \tag{4}$$

Suppose that in this probabilistic setting for the rates we further invoke the following assumption:

(A5) The following limit exists for all $i = 1, \ldots, n$:

$$\lim_{s \to \infty} E[A_i(s)] := E[A_i].$$

¿From (4) we have the following.

Lemma 3.1 *Under assumption (A5) we have*

$$\lim_{s \to \infty} P(\text{“acceptance at time } s\text{”}) = \frac{\sum_{i=1}^{n} \frac{E[A_i]}{d}}{\sum_{i=i}^{n} \lceil \frac{\bar{a}_i}{d} \rceil}.$$

This result will be of use in the subsequent sections.

4 Application to Queueing Networks

Queueing networks are used extensively in the modeling and analysis of computer systems and networks (e.g., see [7]). To illustrate our ideas for variate generation, consider the open Jackson network (e.g., see [11]), one of the most fundamental queueing networks. There are n

single-server queues. Customers arrive from the outside according to a Poisson process with rate λ and are routed to queue i with probability r_{0i}, $i = 1, \ldots, n$. Service times at queue i are exponentially distributed with parameter μ_i, $i = 1, \ldots, n$. When a customer completes service at queue i, it is routed to queue j with probability r_{ij}; it leaves the network with probability $r_{i0} := 1 - \sum_{j \neq i} r_{ij}$. Service times are assumed to be independent of each other and of the arrival process.

Now consider simulating the Markov process associated with this queueing network. Note that the rate, a_i, associated with queue i is either μ_i or 0 depending on whether queue i is occupied or empty. Note that the rate associated with external arrivals is $a_0 := \lambda$. Given that the current rates are $a_0, \ldots, a_n$, the elapsed time (the inter-event time) until the next arrival or service-completion event is exponentially distributed with parameter $a := a_0 + \cdots + a_n$. This event is a service completion at queue i with probability a_i/a, $i = 1, \ldots, n$, and an arrival with probability a_0/a. After having determined the elapsed time and event type, the alias algorithm (see [4]) can be used to determine, in $O(1)$ time, the queue to which the customer is to be routed (including the possibility of being routed to the outside). A new iteration then begins, where we determine the next inter-event time, the next event-type, and the the next queue to which the customer is routed. Note that at most two a_i's change from iteration to iteration, so that the assumption (A4) is satisfied.

Consider the central component of the above simulation procedure: determining the event type according to the probabilities a_i/a, $i = 0, \ldots, n$. Clearly, either Generate1 or Generate2 can be applied. Let us now characterize the performance of Generate2. We shall do this in the setting of the rates being random, as discussed in Secton 2.1. Note that in this application we have $\bar{a}_i = \mu_i$, $i = 1, \ldots, n$, and $\bar{a}_0 = \lambda$. Also note that, in the context of queueing networks, Assumption (A5) is tantamount to assuming that the following limit exists for all $i = 1, \ldots, n$:

$$\gamma_i := \lim_{s \to \infty} P(\text{“ queue } i \text{ is occupied at time } s\text{”}). \tag{5}$$

We shall obtain an explicit expression for the long-run acceptance probability under the following assumption.

(A6) There exists a nonnegative solution $(\lambda_1, \ldots, \lambda_n)$ to the ‘traffic equations’,

$$\lambda_i = \lambda r_{0i} + \sum_{j=1}^{n} \lambda_j r_{ji}, \quad i = 1, \ldots, n,$$

such that $\rho_i := \lambda_i/\mu_i < 1$ for all $i = 1, \ldots, n$.

Theorem 4.1 *Suppose that d is a divisor for μ_i, $i = 1, \ldots, n$, and for λ. Then under assumption (A6), the long-run acceptance probability for Generate2 for the Jackson network is given by*

$$\lim_{s \to \infty} P(\text{"acceptance at time } s\text{"}) = \frac{\lambda + \sum_{i=1}^{n} \mu_i \gamma_i}{\lambda + \sum_{i=1}^{n} \mu_i}. \tag{6}$$

Moreover, we have

$$\gamma_i = 1 - (1 - \rho_i)\left(\frac{\lambda + \sum_{j \neq i} \lambda_j}{\lambda + \sum_{j=1}^{n} \lambda_j}\right). \tag{7}$$

Consequently,

$$\lim_{s \to \infty} P(\text{"acceptance at time } s\text{"}) \geq \frac{\lambda + \sum_{i=1}^{n} \lambda_i}{\lambda + \sum_{i=1}^{n} \mu_i}. \tag{8}$$

Proof: Under (A6) the associated Markov process for the Jackson network is irreducible, positive recurrent, and aperiodic. Hence, the limit in (5) exists so that (A5) holds true. Thus, (6) follows from Lemma 3.1 and the facts: $E[A_i(s)] = \mu_i P(\text{"queue } i \text{ is occupied at time } s\text{"})$, $i = 1, \ldots, n$, $E[A_0(s)] = \lambda$.

It remains to determine γ_i, the limit of the queue occupancy probability at *event times*. (Note that this does not necessarily equal the queue occupancy averaged over time.) To this end, let L_i denote the random variable for the number of customers present at queue i in steady state. It is well known that

$$P(L_1 = l_1, \ldots, L_n = l_n) = \prod_{i=1}^{n} (1 - \rho_i)\rho_i^{l_i}. \tag{9}$$

It follows from Theorem 2.10.6 of [11] and (9) that the total rate at which events occur is

$$\begin{aligned} \beta &:= \lambda + \sum_{j=1}^{n} \mu_j P(L_j > 0) \\ &= \lambda + \sum_{j=1}^{n} \lambda_j \end{aligned}$$

and that

$$\begin{aligned} \beta(1 - \gamma_i) &= \mu_i(1 - r_{ii})P(L_i = 1) + \lambda(1 - r_{0i})P(L_i = 0) \\ &+ P(L_i = 0) \sum_{j \neq i} P(L_j > 0)\mu_j(1 - r_{ji}) \\ &= (1 - \rho_i)(\lambda + \sum_{j \neq i} \lambda_j). \end{aligned}$$

Equation (7) directly follows. □

We see from (8) that if the time average utilization, ρ_i, at each queue i is at least $1/2$ (as is the case in most applications of practical interest), then the long-run acceptance probability is greater than $1/2$. Therefore, under these traffic conditions, the algorithm Generate2 should be faster than the future event schedule for networks of even moderate size. As compared with the algorithm in [4], Generate2 has about the same performance in the case of globally heavy traffic (i.e., $\rho_i \approx 1$ for all $i = 1, \ldots, n$), but significantly better performance in lighter traffic. Equation (8) also tells us that it is particularly desirable to have high values of ρ_i for those queues i with high service rates.

Let us now attempt to characterize the performance for some other classes of queueing networks. First consider the case of multiple classes, where each class c, $c = 1, \ldots, C$, has exogenous arrival rate λ^c and has routing probabilites r_{ij}^c, $0 \leq i, j \leq n$. Suppose that the service rate at each queue is given by μ_i and does not depend on the class. Also suppose that the service discipline at each queue is First-Come-First-Serve (FCFS). There are now C sets of traffic equations; suppose each equation has a nonnegative solution $(\lambda_1^c, \ldots, \lambda_n^c)$. Let $\lambda_i := \lambda_i^1 + \cdots + \lambda_i^C$, and $\lambda := \lambda^1 + \cdots + \lambda^C$, and suppose that $\rho_i := \lambda_i/\mu_i < 1$ for all $i = 1, \ldots, n$. With these modified definitions and assumptions, it is well known that (9) continues to hold true for this multiclass network. Hence, Theorem 4.1 holds unchanged for this multiclass network.

Second consider the multiclass network described above, but with s_i servers at queue i. Now suppose that $\rho_i = \lambda_i/\mu_i < s_i$. Under these conditions, the associated Markov process is again irreducible, positive recurrent, and aperiodic. Therefore, the following limit exits for all $i = 1, \ldots, n$:

$$\gamma_i := \lim_{s \to \infty} E[\# \text{ of busy servers at queue } i \text{ at time } s]. \tag{10}$$

Note that, with this new definition of γ_i, we have $0 < \gamma_i < s_i$. It is easily seen that

$$\lim_{s \to \infty} P(\text{"acceptance at time } s\text{"}) = \frac{\lambda + \sum_{i=1}^n \mu_i \gamma_i}{\lambda + \sum_{i=1}^n s_i \mu_i}.$$

As in (7), γ_i can be expressed in terms of the defining parameters of the network. The derivation of this complicated expression is left to the reader.

Third consider the multiclass, multiserver network discussed above, but now suppose that it is closed, i.e., customers neither enter nor leave the network, so that the population size in the network is fixed for each class. The finite-state Markov process associated with this network is irreducible and aperiodic. Thus assumption (A5) holds and the long-run

acceptance probability is given by

$$\lim_{s\to\infty} P(\text{"acceptance at time s"}) = \frac{\sum_{i=1}^{n} \mu_i \gamma_i}{\sum_{i=1}^{n} s_i \mu_i}$$

with γ_i defined as in (10). In this case γ_i can be expressed in terms of the defining parameters through 'normalization constants'.

The networks discussed above are 'product-form' queueing networks. Let us now consider a non-product-form network. In particular, consider the closed, multiclass, multiserver network discussed in the paragraph above, but now suppose that the service rate for class c customers at queue i is μ_i^c (i.e., the service rates now depend on the class as well as the queue). The maximum service rate at queue i is now given by $s_i\bar{\mu}_i$ where $\bar{\mu}_i := \max\{\mu_i^c : c = 1, \ldots, C\}$. Let

$$\gamma_i^c := \lim_{s\to\infty} E[\#\text{ of servers busy with class } c \text{ customers at queue } i \text{ at time } s].$$

Assumption (A5) again holds and we have

$$\lim_{s\to\infty} P(\text{"acceptance at time s"}) = \frac{\sum_{i=1}^{n} \sum_{c=1}^{C} \mu_i^c \gamma_i^c}{\sum_{i=1}^{n} s_i \bar{\mu}_i}.$$

However, the current tools of queueing theory do not offer a means for expressing γ_i^c in terms of the defining parameters for this non-product-form network.

We conlcude this section by mentioning that the methods discussed above, and the method of Fox [4], are based on simulating the transitions of the Markov process, which is different from the more traditional method based on a future event schedule. However, as Fox points out, many of the features of the traditional method are also available for the 'Markov process' method. For example, we can estimate sojourn time distribtions by keeping an ordered list for each queue and moving customers from queue to queue at event times.

5 Improving Performance

Although the algorithm Generate2 should be useful for many applications, there are situations for which its performance will be less than satisfactory. In particular, this will occur when the expected rates are far from their respective upper bounds, i.e., when $\sum_{i=1}^{n} E[A_i] << \sum_{i=1}^{n} \bar{a}_i$ (see Lemma 3.1). In this section we discuss three modifications of Generate2 which are designed to alleviate or even overcome this problem. In a subsequent

paper ([9]) we will discuss how some of these modifications can be employed in the efficient simulation of large-scale telephone networks.

Before discussing these modifications, it will be beneficial to introduce yet another algorithm for generating variates with changing distributions. To our knowledge, this algorithm cannot be found in the simulation literature. However a similar algorithm has been given in [8] [Ex. 4.27 and page 422] in a different context. An informal description of the algorithm is as follows. (For the sake of convenience, suppose that $\log_2 n$ is an integer in this discussion.) The algorithm is based on a binary tree with $1 + \log_2 n$ levels and n leaf nodes at the bottom level. The value associated with the ith leaf node is a_i. The value associated with any other node is the sum of the values associated with the two sons of that node. Thus, the value of the node at the root of the binary tree will be $a := a_1 + \cdots + a_n$. Note that $O(n)$ preprocessing time is needed to set up the binary tree. Now to generate a variate for X, we first generate a uniform number over $(0, a)$. We then compare this number with the value associated with the left son of the root. If it is less, we know that the variate is in $\{1, 2, \ldots, n/2\}$, so we proceed with the algorithm on the left side of the binary tree. If it is more, we proceed on the right side of the tree. Note that the number of comparisons needed to generate a variate for X is $\log_2 n$. If an a_i changes to a_i' after generating a variate for X, we can update the binary tree with $O(\log n)$ operations as follows. We first reset the value associated with leaf node i to a_i'. We then move up the path between this node and the root and reset the sums accordingly.

In summary, the above 'binary-tree algorithm' requires $O(n)$ preprocessing time and $O(\log n)$ time per variate to generate variates for X with changing distributions (with assumption (A4) still in force). The advantage of this algorithm, as compared to Genearte2, is that its performance does not depend on how close the a_i's are to their upper bounds. Its disadvantage is that it can take significantly more time to generate a variate for X when n is large and the acceptance probability for Generate2 is not small.

5.1 Method I: Partitioning

For any subset S of $\{1, \ldots, n\}$, let

$$h(S) := \frac{\sum_{i \in S} \frac{E[A_i]}{d}}{\sum_{i \in S} \lceil \frac{\bar{a}_i}{d} \rceil},$$

$$\hat{a}(S) := \sum_{i \in S} E[A_i],$$

and

$$a(S) := \sum_{i \in S} a_i,$$

Now suppose there exists a partition (S_1, S_2) of $\{1, \ldots, n\}$ with the following properties: (i) $h(S_1)$ is not 'small'; (ii) $h(S_2)$ is 'small'; (iii) $\hat{a}(S_1)$ is in the 'vicinity of' or larger than $\hat{a}(S_2)$.

Under the above conditions, the following scheme makes sense. First draw a uniform random variate U. If $U < a(S_1)/[a(S_1)+a(S_2)]$ then we declare that the variate for X belongs to S_1 and we use Generate2 (across S_1) to determine it. If $U \geq a(S_1)/[a(S_1) + a(S_2)]$, then we declare that the variate for X belongs to S_2 and we use the binary-tree method (across S_2) to find it. Of course, in order to utilize this method, we need to get a handle on $E[A_i]$, $i = 1, \ldots, n$. This can perhaps be done with analytical analysis or with pilot runs.

In the context of queueing networks, the above method may be suitable when a fraction of the queues are in heavy traffic and the remaining queues are in light traffic.

5.2 Method II: Pseudo Upper Bounds

Here, for those indices i such that $E[A_i] << \bar{a}_i$, we replace $\bar{a}_i$ with a smaller value, perhaps with $E[A_i]+3\sigma(A_i)$ where $\sigma(A_i)$ is the long-run standard deviation of $A_i(s)$. (This, of course, assumes that one can get a handle on $\sigma(A_i)$ as well as on $E[A_i]$). Now we may have $a_i > \bar{a}_i$ for some indices i. When this occurs, the algorithm Generate2 is no longer correct since we may have $C[i] > 1$ for some i. In order to rectify the algorithm, we keep track of the set $\Phi := \{i : a_i > \bar{a}_i\}$ and of the counter $\delta := \max\{a_i/\bar{a}_i : i \in \Phi\}$. (If Φ is empty, set $\delta = 1$.) Then in Step 3 of Generate2 we replace the test '$R \leq C[I]$' with the test '$R \leq C[I]/\delta$'. We leave it to the reader to verify the correctness of the procedure.

But when is it necessary to update Φ and δ, and how much effort is required for each update? Suppose that an a_i changes to an a_i'. It is only necessary to perform an update in the following circumstances: (i) $a_i/\bar{a}_i \leq 1$ and $a_i'/\bar{a}_i > 1$; (ii) $a_i/\bar{a}_i > 1$ and $a_i'/\bar{a}_i > \delta$; $(iii)a_i/\bar{a}_i = \delta$ and $a_i'/\bar{a}_i < \delta$; (iv) $a_i/\bar{a}_i > 1$ and $a_i'/\bar{a}_i \leq 1$. In order to minimize the effort to update Φ and δ, Φ can be implemented as a priority queue. With such a data structure, for any one of the four events occurs, $O(\log|\Phi|)$ operations are sufficient for the update. The pseudo upper bounds should be chosen large enough so that $|\Phi|$ and δ are typically small. However, they should not be chosen so small that the acceptance probability becomes undesirably low.

5.3 Method III: Global Updates

For Method II it may not be possible to determine, a priori, good pseudo upper bounds $\bar{a}_i$, $i = 1, \ldots, n$. Or it may be the case that the appropriate choice of psuedo upper bounds changes with the evolution of the underlying process that we are simulating. In these cases we may want to consider occasional global updates of our pseudo upper bounds and reinitializations of the arrays B and C. The reinitialization will make sure that all the entries in C are close to 1. Of course, the process of global update is very costly (requiring $\Omega(n)$ time). But if this has to be done very rarely, it may be advantageous.

For illustration, suppose α is small when compared to $\bar{\alpha}$ (the average of our pseudo upper bounds). It may be worthwhile to perform a global update. But how do we detect at any given time if α is low or not? When α is small, many entries in the array C will be much less than 1. We can make use of the following sampling process: We pick say $10 \log n$ random outcomes. If a major fraction of these outcomes have an entry much less than (say) $\frac{1}{2}$ in the C array, we perform a global update. Using Chernoff bounds we can show that if a major fraction in the sample has a low $C[]$ value, then a major fraction of all the $C[]$ values will be low with high probability. Also, we perform this checking only every $20 \log n$ samples (thus making sure that the total additional cost per sample due to this checking is no more than a fraction).

6 Conclusions

In this paper we have presented $O(1)$ expected time algorithms for generating a nonuniform discrete random variate (when the distribution changes). Though these algorithms have the potential of finding many applications, an important open question is if there exists a constant time algorithm for discrete random variate generation which does not make any assumptions on the way the rates of X change.

References

[1] Angluin, D., and Valiant, L., 'Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings,' Journal of Computer Systems and Science 18(2), 1979, pp. 155-193.

[2] Bratley, P., Fox, B.L., and Schrage, L.E., *A Guide to Simulation*, Springer-Verlag Publications, 1983.

[3] Chernoff, H., 'A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations,' Annals of Mathematical Statistics 23, 1952, pp. 493-507.

[4] Fox, B.L., 'Generating Markov-Chain Transitions Quickly: I,' Operations Research Society of America Journal on Computing, vol. 2, no. 2, Spring 1990, pp. 126-135.

[5] Fox, B.L., and Young, A.R., 'Generating Markov-Chain Transitions Quickly: II,' Operations Research Society of America Journal on Computing, vol. 2, no. 1, Winter 1991, pp. 3-11.

[6] Kleinrock, L., *Queueing Systems, Volume 1: Theory*, John-Wiley and Sons Publishers, 1975.

[7] Heidelberger, P., and Lavenberg, S.S., 'Computer Performance Evaluation Methodology,' IEEE Transactions on Computers 33, 1984, pp. 1195-1220.

[8] Manber, U., *Introduction to Algorithms: A Creative Approach*, Addison-Wesley Publishing Company, 1989.

[9] Prindiville, M., Rajasekaran, S., and Ross, K.W., 'Efficient Simulation of Large-Scale Loss Networks,' *in preparation.*

[10] Rajasekaran, S., and Reif, J.H., 'Derivation of Randomized Sorting and Selection Algorithms,' Technical Report, Aiken Computing Lab, Harvard University, March 1987.

[11] Walrand, J., *An Introduction to Queueing Networks*, Prentice Hall, 1988.