Optimal Algorithms for Generating Discrete Random Variables with Changing Distributions

T. Hagerup* K. Mehlhorn† I. Munro‡

Abstract

We give optimal algorithms for generating discrete random variables for changing distributions. We discuss two models of how distributions may change. In both models, we obtain a solution with constant update time, constant expected generate time, and linear space.

1 Introduction

A *pseudo-distribution* on integers $[1\,..\,n]$ is an n-tuple $(a_1, \ldots, a_n)$ of non-negative real numbers. The elements a_i are called *frequencies*. A pseudo-distribution is called a *distribution* if the frequencies sum up to one. We consider the problem of generating a discrete random variable X which for $i \in [1\,..\,n]$ attains value i with probability proportional to a_i. This operation is one of the most fundamental and frequently used in discrete event simulation [BFS83]. In the case of static frequencies the well-known 'alias-algorithm' [BFS83, pp.147–148] takes $O(n)$ preprocessing time, after which it can generate a value in time $O(1)$.

In this paper, we develop efficient algorithms for the case of the a_i's changing over time, extending work of Fox [Fox91] and Rajasekaran and Ross [RR91]. Our results lead to more efficient simulations of an important class of queuing networks, the so–called open Jackson networks (see [Kle75]). We sketch this application in section 5.

We study two versions of the problem. In the first version of the problem, called the problem of *maintaining a distribution*, $(a_1, \ldots, a_n)$ is a distribution and an update changes two probabilities a_i and a_j but leaves their sum invariant. In the second version of the problem, called the problem of *maintaining a pseudo-distribution*, $(a_1, \ldots, a_n)$ is a pseudo-distribution and an update changes one of the frequencies a_i. We obtain the following results.

Theorem 1 *A distribution can be maintained with constant update and constant expected generate time. The space requirement and preprocessing time is linear.*

A pseudo-distribution $(a_1, \ldots, a_n)$ is called *polynomially-bounded* if $a_i \in \mathbb{N}$ and $\sum_i a_i \leq n^c$ for some fixed integer c (independent of n).

Theorem 2 *A polynomially-bounded pseudo-distribution on $[1\,..\,n]$ can be maintained with constant update and constant expected generate time. The space requirement and preprocessing time is linear.*

*Max–Planck–Institut für Informatik, Im Stadtwald, D-6600 Saarbrücken; part of the work was carried out during an appointment to the Departament de LSI of the Universitat Politècnica de Catalunya in Barcelona, Spain

†Max–Planck–Institut für Informatik and Fachbereich Informatik, Universität des Saarlandes, Im Stadtwald, D-6600 Saarbrücken

‡Department of Computer Science, University of Waterloo, Canada; this research was carried out while the author was visiting the MPI für Informatik

Both theorems substantially improve upon previous results. The only previous method which could deal with general update operations and put no restriction on the distribution (or pseudo-distribution) had generate and update time $O(\log n)$ [BFS83]. Sublogarithmic methods were only known under the assumption that there are a-priori bounds $\underline{a}_1, \overline{a}_1, \underline{a}_2, \ldots, \underline{a}_n, \overline{a}_n$ such that $a_i \in [\underline{a}_i, \overline{a}_i]$ for all $i \in [1\,..\,n]$. Under this assumption, Rajasekaran and Ross [RR91] achieve update and generate time $O(\overline{\alpha}/\underline{\alpha})$ where $\overline{\alpha} = \sum_i \overline{a}_i/n$ and $\underline{\alpha} = \sum_i \underline{\alpha}_i/n$.

Our results are valid in the standard real number RAM-model. More specifically, we assume that the following operations take constant time:

1. arithmetic (addition, subtraction, multiplication, integer division) on integers whose absolute value is bounded by a polynomial in n,
2. real arithmetic and the floor function,
3. generating a uniformly distributed real number between 0 and 1.

This paper is structured as follows. In section 2 we develop some basic tools, in section 3 we show how to maintain distributions and in section 4 we show how to maintain pseudo-distributions.

2 Basic Building Blocks

2.1 Bounded Pseudo-distributions

Lemma 1 *A pseudo-distribution can be maintained with constant update time and expected generate time $O(nB/A)$ where $A = \sum_{1 \le i \le n} a_i$ and B is a possibly time–dependent known upper bound on max a_i. The space requirement is $O(n)$.*

Proof: We maintain the bound B and an array $a[\,]$ with $a[i] = a_i$ for $1 \le i \le n$. Then update time is clearly constant.

To generate a value for X, repeat the following process until a value is output: choose $i \in [1\,..\,n]$ uniformly at random and output i with probability a_i/B (To do so, choose $X \in [0, 1]$ uniformly at random and output i if $X \le a_i/B$).

This algorithm clearly outputs integer i with probability proportional to a_i, i.e., the method is correct. Also, a particular iteration leads to an output with probability $A/(Bn)$, i.e., expected generate time is $O(nB/A)$. ∎

We refer to the method described above as the *bounded distribution method*. For the application in section 4 the following observation is useful: The array $a[\,]$ need not be maintained explicitly; it suffices to assume that given i the frequency a_i can be determined in time $O(1)$.

2.2 Flat Pseudo-Distributions

Let $A \ge 0$. A pseudo-distribution $(a_1, \ldots, a_n)$ on $[1\,..\,n]$ is called *A-flat* if $a_i \in \{0\} \cup [A, 2A]$, for $i = 1, \ldots, n$.

Lemma 2 *For all given $A \ge 0$, an A-flat pseudo-distribution $(a_1, \ldots, a_n)$ can be maintained with constant update and constant expected generate time.*

Proof: We will store the set $S := \{i : 1 \le i \le n \text{ and } a_i \ne 0\}$ in a data structure Q that supports *insert* and *delete* operations in constant time and *random* queries, which return a random element of the set stored in Q drawn from the uniform distribution over that set, in constant expected time. In order to produce a random

variate, repeat the following until a number is output: Compute $i = \mathit{random}(Q)$ and output i with probability $a_i/(2A)$. The correctness of the procedure is obvious, and it can be seen to work in constant expected time.

We now describe the implementation of the data structure Q. The static version of the problem, i.e., if only *random* queries on a set $S \subseteq \{1, \ldots, n\}$ must be supported, is extremely easy: Simply store the elements of S in an array $H[1\,..\,|S|]$ and execute a *random* query by drawing a random number from the uniform distribution over $[1\,..\,|S|]$ and returning the contents of the corresponding cell of H.

In order to cope with updates, first introduce an array $D[1\,..\,n]$ such that for all $x \in S$, $D[x]$ stores the index of the unique cell in H that contains x. (The value of $D[x]$ for $x \notin S$ is arbitrary.) Also allow H to be a *padded* array, i.e., allow $|H|$, the size of H, to exceed $|S|$ and fill all unused cells with a special *null* value. We will maintain the invariant that precisely the first $|S|$ cells of array H are used. This implies that a *random* query can still be performed as described above. An insertion simply stores the new element at position $|S| + 1$ of array H and a deletion, say at position i of H, simply moves $H[|S|]$ to $H[i]$. ∎

We refer to the method described above as the *flat distribution method*. In the applications of the flat distribution method in Sections 3 and 4 we will partition $[1\,..\,n]$ into groups and apply the flat distribution method to each group (with different values of A). We will now discuss how this can be done without allotting space $O(n)$ to each group. The array D will always be shared among all groups. Note that this is possible since each element $x \in [1\,..\,n]$ belongs to exactly one group. For the arrays H we have to work harder.

In section 3 we partition $[1\,..\,n]$ into groups $G_0, \ldots, G_k$ where $k = \lceil \log n \rceil$. Moreover $|G_i| \leq n/2^{i-1}$ for all $i \in [0\,..\,n]$. We can therefore use an array H_i of size $n/2^{i-1}$ for the i–th group. The total space requirement for all arrays H_i is thus $O(n)$.

In section 4 we partition $[1\,..\,n]$ into groups $G_0, \ldots, G_k$ where $k = O(\log n)$. In contrast to section 3 no non–trivial bound on the size of $|G_i|$ will be available. We will now describe two schemes: a simple scheme which uses $O(n)$ space and guarantees constant amortized update time and a more complicated scheme with constant worst–case update time. Both schemes are suggested by the known methods for dynamizing static data structures [OvL81].

The simple scheme operates in cycles. In each cycle $2n$ insertion into and deletions from groups are processed. Consider a cycle. For i, $0 \leq i \leq k$, let s_i be the current size of the i–th group and let h_i be the current size of the array H_i used for the i–th group. Then $h_i \geq s_i$. We have an array $K[1\,..\,11n]$ which is divided into a used and an unused part. All arrays H_i are contained in the used part of K. At the beginning of a cycle set h_i to $2s_i$ (to 1 if $s_i = 0$) and allocate the arrays H_i in an initial segment of K of length $3n$ (note that $\sum_i \max(1, 2s_i) \leq 3n$).

We now process updates as described in Lemma 2 except when s_i exceeds h_i. When some s_i exceeds h_i double h_i and allocate a new array H_i for the i–th group in the unused part of K. This has cost $O(h_i)$. Since at least $h_i/2$ insertions into G_i took part since the last allocation of H_i the amortized cost per insertion is constant. At the end of a cycle the H_i's are compacted into the first $3n$ cells of K. We still need to argue that K never overflows. This follows from the observation that initially $8n$ cells of K are unused and that whenever a new H_i is allocated, say of size $2h_i$, there were at least $h_i/2$ insertions into G_i since the previous allocation. Thus 4 cells are used per insertion for a total of at most $8n$. This completes the description of the simple scheme.

Amortization is used at two levels in the simple scheme: when a new array H_i is allocated (inner level) and when a cycle is completed (outer level). We deal with

the two levels in turn.

To cope with the inner level of amortization we increase the size of K to $27n(= 3n + 8n + 16n)$, have an additional array $\overline{H}_i[1\,..\,2h_i]$ for each $i \in [1\,..\,n]$, and maintain the invariant that the content of $\overline{H}_i$ is exactly the content of the new H_i in the simple scheme whenever h_i doubles. This is ensured as follows. Whenever h_i doubles, make the old $\overline{H}_i$ the new H_i and allocate space for the new $\overline{H}_i$. Do not initialize $\overline{H}_i$. For each update executed on H_i perform the same update on $\overline{H}_i$ and in addition initialize 4 cells of $\overline{H}_i$ by either copying values from H_i or storing *null* whatever is appropriate. This ensures that $\overline{H}_i$ is completely initialized by the time H_i overflows.

We next deal with the outer level of amortization. Consider the midpoint of a cycle, i.e., when n updates are processed. Process the next n updates as described above but in addition perform the following actions:

- when an element of K is written keep also the last value before the midpoint,
- record the sequence of updates in a log–file, and
- compact the content of K at the midpoint into an initial segment of length $3n$ and perform the updates recorded in the log–file on it.

The last action takes $O(n)$ time. It is performed in piecemeal fashion, $O(1)$ steps per update, such that it is completed within the second half of the cycle. Note that we have now a version of K on which exactly n updates were performed, i.e., this K is exactly at the midpoint of its cycle. We operate on this K exactly as described above. This completes the description of the scheme with constant worst–case update time.

2.3 Small Ground Sets

We deal with pseudo-distributions over small ground sets and with small integer-valued frequencies. More specifically, we make the following assumption: k is an integer with $(k+1)^{k+3} \leq n$ and $(a_1, \ldots, a_k)$ is an integer-valued pseudo-distribution on $[1\,..\,k]$ with $a_i \in [0\,..\,k]$ for all i, $1 \leq i \leq k$. Under this assumption, a pseudo-distribution can be encoded as an integer fitting into a single word and generating random values and updating the pseudo-distribution can be done by table lookup.

Lemma 3 *Under the assumption above a pseudo-distribution can be maintained with constant update and generate time. The solution requires space and preprocessing time $O(n)$.*

Proof: We encode a pseudo-distribution $(a_1, \ldots, a_k)$ as the integer $A = A(a_1, \ldots, a_k)$ with $(k+1)$–ary expansion $(a_1, \ldots, a_k)$, i.e., $A(a_1, \ldots, a_k) = \sum_{1 \leq i \leq k} a_i (k+1)^{i-1}$, and store A and $S = a_1 + \ldots + a_k$. We also precompute tables $L[0\,..\,(k+1)^k - 1, 1\,..\,k^2]$ and $U[0\,..\,(k+1)^k - 1, 1\,..\,k, 0\,..\,k]$ such that $L[A(a_1, \ldots, a_k), l] = i$ for exactly a_i values $l \in [1\,..\,S]$
and
$U[A(a_1, \ldots, a_k), i, b] = A(a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_k)$ for all $(a_1, \ldots, a_k) \in [0\,..\,k]^k, i \in [1\,..\,k]$ and $b \in [0\,..\,k]$. These tables can be computed in time $O((k+1)^{k+3}) = O(n)$.

To generate a value for X we generate $l \in [1\,..\,S]$ uniformly at random and return $L[A, l]$.

To change the value of a_i to a_i' we replace A by $U[A, i, a_i']$ and S by $S - a_i + a_i'$. ∎

We refer to the method described above as the *small ground set method*.

2.4 Bounded Ordered Sets

Let k be an integer. The *bounded ordered set problem* on $[1\,..\,k]$ is to maintain a set $S \subseteq [1\,..\,k]$ under the operations $Insert(i)$, $Delete(i)$, $Member(i)$, $Pred(i)$, and $Suc(i)$ where the last two operations return the nearest predecessor and successor of i in S respectively (0 if there is no such element).

Lemma 4 *If $k = (\log n)^{O(1)}$ then an ordered set on $[1\,..\,k]$ can be maintained in constant time per operation. Preprocessing time and space requirement is $O(n)$.*

Proof: Let k_0 be the largest integer with $2^{k_0}k_0^2 \leq n$. We first show how to maintain a subset of $[1\,..\,k_0]$. Encode a set as a number in $[0\,..\,2^{k_0}-1]$ and prepare a table for each operation. These tables have $2^{k_0}k_0$ entries and can be set up in time $O(k_0)$ per entry.

For $k = (\log n)^{O(1)}$ let $c \in \mathbb{N}$ be such that $k \leq k_0^c$. Represent $[1\,..\,k] \subseteq [1\,..\,k_0^c]$ as a k_0-ary tree of depth c and use the structure described in the preceding paragraph for each node of the tree. ∎

2.5 Logarithms

The methods to be described in sections 2.6, 3 and 4 require the computation of binary logarithms. We describe how this is done.

For the integers i between 1 and n we precompute a table $L[1\,..\,n]$ with $L[i] = \lceil \log i \rceil$. This table requires linear space and can be set up in time $O(n)$.

To determine $\lceil \log pn \rceil$ for $p \in [1/n, 1]$ compute $\lceil \log \lceil pn \rceil \rceil$.

To compute $\lceil \log x \rceil$ for $x \in [1\,..\,n^c]$ and c a fixed integer (independent of n), let $i_0 = max\{i \geq 0 : \overline{n}^i \leq x\}$, where $\overline{n}$ is the largest power of 2 no larger than n, and then use $\lceil \log x \rceil = i_0 \log \overline{n} + \lceil \log \lceil x/\overline{n}^{i_0} \rceil \rceil$.

2.6 Small Ground Sets Revisited

We refine the small ground set method of section 2.3. Let c be a fixed integer constant. We make only the following weaker assumption: k is an integer with $(k_0+1)^{k_0+3} \leq n$, where $k_0 = 2^{2\lceil \log k \rceil} - 1$, and $(a_1, \ldots, a_k)$ is a pseudo-distribution on $[1\,..\,k]$ with $a_i \in [0\,..\,n^c - 1]$ for all i, $1 \leq i \leq k$.

Lemma 5 *Under the assumption above a pseudo-distribution can be maintained with constant update and generate time. The solution requires space and preprocessing time $O(n)$.*

Proof: The idea is as follows. Split each probability into a *rounded part* and a *remainder* such that the small ground set method is applicable to the pseudo-distribution of the rounded parts and the bounded distribution method is applicable to the pseudo-distribution of the remainders. To generate a random value first choose one of the two distributions with appropriate probability and then apply the respective method. To update a frequency recompute the rounded part and the remainder and make the appropriate changes in the two distributions. We now give the details.

Let $s = \lceil \log k \rceil$. Split the binary representations of the frequencies a_i into blocks of s bits each, i.e., write $a_i = \sum_j a_{ij} 2^{sj}$ where $a_{ij} \in [0\,..\,2^s - 1]$ for $0 \leq j \leq L := \lceil (c \log n)/s \rceil$. Let $j(a_i)$ be the maximal j with $a_{ij} \neq 0$ and define A_{ij} by $A_{ij} = a_{ij}$ if $j \in \{j(a_i), j(a_i) - 1\}$ and $A_{ij} = 0$ otherwise. Let $a_{max} = max_{1 \leq i \leq k} a_i$ and let $j_{max} = j(a_{max})$. Then $A_{ij} = a_{ij}$ for all i and all $j \geq j_{max} - 1$. Let $S = a_1 + \ldots + a_k$ and let $S_j = \sum_i A_{ij}$ for $0 \leq j \leq L$.

For $i \in [1\,..\,k]$, the rounded part b_{1i} of a_i is now given by $b_{1i} = A_{ij_{max}}2^s + A_{ij_{max}-1}$ and the remainder b_{2i} is given by $b_{2i} = a_i - b_{1i}2^{s(j_{max}-1)}$. Let $b^{(1)} = (b_{11}, \ldots, b_{1k})$ be the pseudo-distribution of the rounded parts and let $b^{(2)} = (b_{21}, \ldots, b_{2k})$ be the pseudo-distribution of the remainders. Note that

- $\sum_i b_{1i} = S_{j_{max}} \cdot 2^s + S_{j_{max}-1}$
- $b_{1i} < 2^{2s}$ for all i, i.e., the small ground set method is applicable to $b^{(1)}$, and
- $b_{2i} \leq a_{max}/2^s \leq S/2^s \leq S/k$ for all i, i.e., the bounded distribution method with bound S/k is applicable to $b^{(2)}$ (It will become clear below that, given i, b_{2i} can be computed in constant time.).

We can now describe how to generate a value for X. First select pseudo-distribution $b^{(1)}$ with probability $p_1 := (S_{j_{max}} \cdot 2^s + S_{j_{max}-1}) \cdot 2^{s(j_{max}-1)}/S$ and pseudo-distribution $b^{(2)}$ with probability $p_2 := 1 - p_1$, and then select a value i according to either $b^{(1)}$ or $b^{(2)}$ using either the small ground set or the bounded distribution method. The expected generate time is $O(p_1 \cdot 1 + p_2 \cdot k \cdot (S/k) \cdot (1/\sum_i b_{2i})) = O(1)$.

We next turn to updates. We maintain

- the values a_i, A_{ij}, S_j, S for all $i \in [1\,..\,k]$ and $j \in [0\,..\,L]$.
- for each $j \in [1\,..\,L]$ the pseudo-distribution $(A_{ij}2^s + A_{ij-1})_{1 \leq i \leq k}$ according to the small ground set method. (Note that $b^{(1)}$ as defined above is identical to the pseudo-distribution $(A_{ij_{max}}2^s + A_{ij_{max}-1})_{1 \leq i \leq k}$. We maintain these pseudo–distributions for all j and not only for $j = j_{max}$ to allow for efficient updates.)
- the value j_{max}, for each $j \in [0\,..\,L]$ the number n_j of non-zero A_{ij}'s, and the subset I of $[0\,..\,L]$ consisting of all indices j with $n_j \neq 0$. The set I is maintained using the bounded ordered set structure of section 2.4.

Suppose now that frequency a_i is to be changed to a_i'. We may assume w.l.o.g. that either a_i or a_i' is zero. We discuss the former case and leave the latter to the reader.

Compute $l := j(a_i')$ (according to the formula $j(a_i') = \lfloor \lfloor \log a_i' \rfloor / s \rfloor$). Compute a_{il}' and a_{il-1}' and set A_{il} and A_{il-1} to these values (according to the formulae $a_{il}' = \lfloor a_i'/2^{sl} \rfloor$ and $a_{il-1}' = \lfloor (a_i' - a_{il}'2^{sl})/2^{s(l-1)} \rfloor$). Increment S_l by A_{il} and S_{l-1} by A_{il-1}, insert l and $l-1$ into I, increment n_l and n_{l-1}, increment S by a_i, and update j_{max} using the bounded set structure on I. Finally, update the distributions $(A_{ij}2^s + A_{ij-1})_{1 \leq i \in k}$ for $j = l$ and $j = l-1$. All of this takes constant time. ∎

We refer to the method described above as the *refined small ground set method*.

3 Maintaining Distributions

We proceed in two steps. We first describe a reduction process which reduces the problem of maintaining a distribution on $[1\,..\,n]$ to the problem of maintaining a distribution on $[0\,..\,\lceil \log n \rceil]$ at an additive cost of $O(1)$ in update and generate time. From this we derive a solution with constant time performance (by combination with the small ground set method).

3.1 A Reduction

The idea for the reduction process is to split a distribution $(p_1, \ldots, p_m)$ into a distribution for the small probabilities (value $\leq 1/m$) and a distribution for the large probabilities (value $> 1/m$) and then in turn to split the distribution of the large probabilities into $\log m$ flat groups. To generate a value for random variable X, first choose a group with the appropriate probability and then use either the bounded or the flat distribution method to determine an actual value for X. The details are as follows.

Let m be an integer and let $(p_1, \ldots, p_m)$ be a distribution on $[1\,..\,m]$. For $j \in [1\,..\,\lceil \log m \rceil]$ let $G_j = \{i \in [1\,..\,m];\ 2^{j-1}/m < p_i \leq 2^j/m\}$ and let $G_0 = \{i \in [1\,..\,m];\ p_i \leq 1/m\}$. For $j \in [0\,..\,\lceil \log m \rceil]$ let $g_j = \sum_{i \in G_j} p_i$.

We maintain:

1. The distribution $g = (g_0, g_1, \cdots, g_{\lceil \log m \rceil})$ on $[0\,..\,\lceil \log m \rceil]$. We call g the *group distribution.*

2. The pseudo-distribution $a^{(0)} = (a_{01}, \ldots, a_{0m})$, where $a_{0i} = p_i$ if $i \in G_0$ and $a_{0i} = 0$ otherwise, according to the small distribution method.

3. For $1 \leq j \leq \lceil \log m \rceil$, the pseudo-distribution $a^{(j)} = (a_{j1}, \cdots, a_{jm})$, where $a_{ji} = p_i$ if $i \in G_j$ and $a_{ji} = 0$ otherwise, according to the flat distribution method. Denote by H_j the array used in the flat distribution method for G_j.

4. An array D which associates with every $i \in [1\,..\,m]$ the probability p_i, the group index j with $i \in G_j$, and, if $j > 0$, the index l with $H_j[l] = i$.

To generate a value for X, choose $j \in [0\,..\,\lceil \log m \rceil]$ according to group distribution g and then i according to pseudo-distribution $a^{(j)}$. Output i.

Consider an update next. Say p_i, p_j are changed into p'_i, p'_j. This amounts to an update of the group distribution g and to up to four updates on pseudo-distributions $a^{(l)}$. Note that the indices h and k with $p'_i \in G_h$ and $p'_j \in G_k$ can be computed in constant time as described in section 2.5.

We summarize in:

Lemma 6 *The method above produces values according to distribution $(p_1, \ldots, p_m)$. Update time is constant and generate time is constant expected time plus the update and generate time for the group distribution. Space requirement is $O(m)$ plus the space for the group distribution.*

Proof: Correctness is obvious. The space bound follows directly from Lemmas 1 and 2. For the time bound we observe that some $i \in G_0$ is produced with probability g_0 and that generating an $i \in G_0$ takes expected time $O(1/\sum_i a_{0i}) = O(1/g_0)$ (by Lemma 1). For values of i in $\cup_{j \geq 1} G_j$ the generate time is constant expected according to Lemma 2. ∎

3.2 A Constant Time Solution

We combine the reduction with the small ground set method. More precisely, we apply the reduction method twice. This leaves us with a distribution $(p_1, \ldots, p_k)$ on $[1\,..\,k]$ where $k = \lceil \log(1 + \lceil \log n \rceil) \rceil + 1$. Note that $(k+1)^{k+3} \leq n$ for sufficiently large n, i.e., Lemma 3 is applicable. We maintain:

1. The number $p_{small} = \sum\{p_i; p_i \leq 1/k\}$.

2. The pseudo-distribution $(a_1, \ldots, a_k)$, where $a_i = p_i$ if $p_i \leq 1/k$ and $a_i = 0$ otherwise, according to the bounded distributions method with bound $1/k$.

3. The pseudo-distribution $(b_1, \cdots, b_k)$, where $b_i = \lceil kp_i \rceil$ if $p_i > 1/k$ and $b_i = 0$ otherwise, according to the small ground set method.

To generate a value $i \in [1\,..\,k]$ according to $(p_1, \ldots, p_k)$ we first choose $x \in [0, 1]$ uniformly at random. If $x \leq p_{small}$ then we choose i according to $(a_1, \ldots, a_k)$ using the bounded distributions method and output i. If $x > p_{small}$ then we repeat the following process until a value is output. Choose i according to $(b_1, \cdots, b_k)$ using the small ground set method and output i with probability kp_i/b_i.

Theorem 1 *A dynamic distribution on $[1\,..\,n]$ can be maintained with constant update and constant expected generate time. The space requirement and preprocessing time is $O(n)$.*

Proof: We only need to argue that the method described above generates values according to distribution $(p_1, \ldots, p_k)$ in constant expected time. All other claims follow directly from Lemma 6.

Let $Small = \{i; p_i \leq 1/k\}$ and $Large = [1\,..\,k] \setminus Small$. Then $p_{small} = \sum_{i \in Small} p_i$ and hence any $i \in Small$ is generated with probability p_i. Also given that $x \leq p_{small}$ it takes expected time $O(k(1/k)/\sum_{i \in Small} p_i) = O(1/p_{small})$ to generate an output, i.e., the contribution to the overall generate time is constant.

Consider $i \in Large$ next. Observe first that $kp_i/b_i \geq 1/2$ for $i \in Large$ and hence the values in $Large$ are also generated in constant expected time. For $i \in Large$ let $g_i = \frac{b_i}{b_1 + \cdots + b_k} \cdot \frac{kp_i}{b_i}$. Given that $x > p_{small}$ any value $i \in Large$ is output with probability $g_i/\sum_{i \in Large} g_i = p_i/(1 - p_{small})$. ∎

4 Maintaining Pseudo-Distributions

As in the previous section we proceed in two steps. We first describe a reduction and then combine the reduction with the refined small ground set method.

Let c be a fixed integer constant and let $A = n^c$. Recall that a pseudo-distribution $(a_1, \ldots, a_n)$ is called polynomially–bounded if $\sum_i a_i \leq n^c$.

4.1 A Refined Reduction

We describe a reduction process refining the one used in section 3.1. The additional complexity arises from the fact that in contrast to section 3.1 the definition of small frequency depends on the current pseudo-distribution.

Let m be an integer and let $a = (a_1, \ldots, a_m)$ be a pseudo-distribution with $a_1 + \ldots + a_m \leq A$. For $j \in [0 \ldots \lceil \log A \rceil]$ let $G_j = \{i \in [1\,..\,m]; 2^{j-1} < a_i \leq 2^j\}$ and $g_j = \sum_{i \in G_j} a_i$. Let $S = a_1 + \ldots + a_m$, let $Large = \{j; g_j \neq 0$ and there are less than $\log m$ indices $k > j$ with $g_k \neq 0\}$, and let $Large = \{j_1, j_2, \ldots, j_h\}$ where $h = \lceil \log m \rceil$ (If $|Large| < \lceil \log m \rceil$ then fill up $Large$ to size $\lceil \log m \rceil$ using the indices $\lceil \log A \rceil$, $\lceil \log A \rceil - 1$, $\ldots$). Note that $Large$ contains the $\log m$ largest non-empty groups. This means that one update can move only $O(1)$ groups from large to non-large and vice versa. We maintain:

1. A bijection π between $[1\,..\,h]$ and $Large$. π is stored as an array $\Pi[1\,..\,h]$ with $\Pi[i] = \pi(i)$ and an array $\Pi'[0\,..\,\lceil \log A \rceil]$ with $\Pi'[j] = \pi^{-1}(j)$ for $j \in Large$ and $\Pi'[j] = nil$ otherwise.

2. The pseudo-distribution $g = (S - \sum_{j \in Large} g_j, g_{\pi(1)}, \ldots, g_{\pi(h)})$ on $[0\,..\,h]$. We call g the *group pseudo-distribution.*

3. For each j, the pseudo-distribution on $b^{(j)} = (b_{j1}, \ldots, b_{jm})$, where $b_{ji} = a_i$ if $i \in G_j$ and $b_{ji} = 0$ otherwise, according to the flat distributions method. Denote by H_j the array H used in the flat distribution method for G_j.

   Remark: We maintain the flat distribution method on all groups G_j and not only on the groups G_j with $j \in Large$. The generating algorithm uses only the structures for $j \in Large$. The advantage for the update algorithm of keeping all structures is that a group can be moved from large to non-large and vice versa at constant cost.

4. The pseudo-distribution $b^{(0)} = (b_{01}, \ldots, b_{0m})$, where $b_{0i} = a_i$ if $i \in \bigcup_{j \notin Large} G_j$ and $b_{0i} = 0$ otherwise, according to the bounded distribution method.

   Remark: If $i \in \bigcup_{j \notin Large} G_j$ then $a_i \leq a_{max} 2^{-\log m} \leq S/m$ where $a_{max} = \max\{a_i; 1 \leq i \leq m\}$, i.e., S/m can be used as a bound in the bounded distribution method.

5. A array D which associates with every $i \in [1\,..\,n]$ its frequency a_i, the group index j with $i \in G_j$, and the index l with $H_j[l] = i$.

6. The minimal element min in $Large$ and a bitvector $B[0\,..\,\lceil \log A \rceil]$ with $B[j] = 1$ iff $G_j \neq \emptyset$. The bitvector B is organized as the bounded ordered set structure of section 2.4.

   Remark: For every i the j with $i \in G_j$ is given by $D[i]$. Comparing j with min decides whether $j \in Large$, i.e., membership of i in $\bigcup_{j \notin Large} G_j$ can be decided in constant time. The bounded distribution method is therefore applicable to $b^{(0)}$. Updates can make non-empty groups empty and vice versa. Suppose that a group G_j with $j \geq min$ becomes empty. We then use the bounded set structure on B to find the index to be added to $Large$ and to update the value of min. Similarly, if some G_j with $j \geq min$ becomes nonempty we use the bounded set structure on B to update min.

To generate a value for X we first generate a $j \in [0\,..\,h]$ and then use either the bounded distribution method (if $j = 0$) to generate i or the flat distribution method (if $j > 0$) on $G_{\pi(j)}$ to generate i.

To change a frequency a_i, we need to update the group pseudo-distribution and up to two of the pseudo-distributions $b^{(j)}$. Note that given the new frequency a_i' its group can be computed in constant time as described in section 2.5. We also may have to add or remove an index from $Large$.

Lemma 7 *The method above produces values according to the pseudo-distribution* $(a_1, \ldots, a_m)$. *Update time is constant and generate time is constant expected plus the update and generate time for the group pseudo-distribution.*

Proof:
Similar to the proof of Lemma 6 and therefore left to the reader.

∎

4.2 A Constant Time Solution

Let $(a_1, \ldots, a_n)$ be a polynomially bounded pseudo-distribution. We combine the refined reduction method of section 4.1 with the refined small ground set method of section 2.6. More precisely, we apply the refined reduction method twice. This leaves us with a pseudo-distribution $(a_1, \ldots, a_k)$ on $[1\,..\,k]$ where $k = \lceil \log(1 + \lceil \log n \rceil) \rceil + 1$ and $\sum_i a_i \leq A$. Note that $(k_0 + 1)^{k_0+3} \leq n$, where $k_0 = 2^{2\lceil \log k \rceil} - 1$, for sufficiently large n, i.e., Lemma 5 is applicable. We apply the refined small ground set method to $(a_1, \ldots, a_k)$. Altogether we obtain:

Theorem 2 *An integer-valued polynomially-bounded pseudo-distribution can be maintained with constant update and constant expected generate time. The space requirement is $O(n)$.*

5 An Application

We briefly discuss an application of our results to the simulation of queuing networks. This application is discussed in more detail in [Fox91, RR91]. Consider an open Jackson network (see e.g. [Kle75]). It consists of n single–server queues. New customers are generated according to a Poisson process with rate λ. A new customer is routed to queue i with probability r_{0i} for $1 \leq i \leq n$. The service time at queue i is exponentially distributed with parameter μ_i. When a customer completes service at queue i, it is routed to queue j, $1 \leq j \leq n$, with probability r_{ij}. It leaves the system with probability $r_{i0} = 1 - \sum_j r_{ij}$. Service times are assumed to be independent from each other and from the arrival process.

In the standard future event scheduling simulation method a priority queue of events is maintained. Whenever an event is processed zero or more new events are generated and inserted into the event queue. Thus processing an event takes time $\Omega(\log m)$ where m is the current size of the event queue. In a heavily loaded system, i.e., a significant fraction of the server queues is non–empty, this is $\Omega(\log n)$.

Fox [Fox91] suggested an alternative simulation strategy. Consider the Markov chain associated with the queuing network (the states are n–tuples $(s_1, \ldots, s_n)$ where s_i is the number of customers in the i–th queue). What are the transition probabilities? Note that the rate a_i associated with the i–th queue is either μ_i or 0 depending on whether the i–th queue is non–empty or not. Also the rate associated with external arrivals is $a_0 = \lambda$. Given that the current rates are $a_0, a_1, \ldots, a_n$ the time to the next event (inter–event time) is exponentially distributed with parameter $S := a_0 + a_1 + \ldots + a_n$. This event is a service completion at queue i, $1 \leq i \leq n$, with probability a_i/S and a new arrival with probability a_0/S. Once the type $C \in [0\,..\,n]$ of the event is determined a customer is moved from queue C according to the routing probabilities r_{Cj}, $0 \leq j \leq n$. The alias–algorithm [BFS83, 147-148]) can be used for the last step. After an event is processed at most two the a_i's change. We can therefore apply the results of section 4 and simulate an event in constant expected time (provided that the rates are polynomially bounded).

Fox [Fox91] and Rajasekaran and Ross [RR91] previously obtained weaker versions of this result: Fox guarantees constant simulation time per event only for very heavily loaded systems and Rajasekaran and Ross only for heavily loaded systems (In technical terms: If $(\lambda_1, \ldots, \lambda_n)$ is a solution to the so–called 'traffic equations' $\lambda_i = \lambda_0 r_{0i} + \sum_j \lambda_j r_{ji}$, $1 \leq i \leq n$, then $\lambda_i \approx \mu_i$ (Fox) and $\lambda_i/\mu_i \geq c$ for some fixed constant $c > 0$ (Rajasekaran and Ross) was required.)

6 Conclusion

We have presented constant expected time algorithms to generate random variables according to time–varying distributions and pseudo–distributions. In the case of distributions our solution is completely general but in the case of pseudo–distributions our solution works only for polynomially–bounded pseudo–distributions. This is due to the fact that we need to compute binary logarithms of integers and know of no constant time implementation of this operation for integers which are not polynomially bounded. If binary logarithms of arbitrary integers could be computed in constant time then our solution would extend to frequencies bounded by $2^{(\log n)^{O(1)}}$. The limiting factor would then be the bounded ordered set problem.

References

[BFS83] P. Bratley, B.L. Fox, and L.E. Schrage. *A Guide to Simulation*. Springer Verlag, 1983.

[Fox91] B.L. Fox. Generating Markov-chain transitions quickly: II. *Operations Research Society of America Journal on Computing*, 2.1:3–11, 1991.

[Kle75] L. Kleinrock. *Queuing Systems*. John Wiley & Sons, 1975.

[OvL81] M. Overmars and J. van Leeuwen. Worst-case optimal insertion and deletion methods for decomposable searching problems. *IPL*, 12:168–173, 1981.

[RR91] S. Rajasekaran and K.W. Ross. Fast algorithms for generating discrete random variates with changing distributions. Technical Report MS-CIS-91-52, Dept. of CIS, Univ. of Pennsylvania, 1991.