JOURNAL OF ALGORITHMS 11, 185–207 (1990)

Generating Random Combinatorial Objects

V. G. KULKARNI*

Department of Operations Research, University of North Carolina, Chapel Hill, North Carolina 27519

Received September 21, 1988; accepted March 23, 1989

Let E be a finite set and $\mathscr{B}$ be a given set of subsets of E. Let $w(B)$ be a positive weight of $B \in \mathscr{B}$. In this paper we develop an algorithm to generate a random subset $U \in \mathscr{B}$ such that the $P(U = B)$ is proportioned to $w(B)$. If the complexity of computing sums of the weights of B contained in set-intervals in $\mathscr{B}$ is $O(f(|E|))$, then the algorithm developed here has complexity $O(|E|f(|E|))$. The algorithm is specialized to the following cases: (i) $\mathscr{B}$ = all subsets of E, (ii) $\mathscr{B}$ = all subsets of E with k elements, (iii) $\mathscr{B}$ = all spanning trees in a network, and (iv) $\mathscr{B}$ = all paths in a directed acyclic network. Different types of weight functions are considered. It is shown that this proposed algorithm unifies many existing algorithms. © 1990 Academic Press, Inc.

1. INTRODUCTION

In this paper we develop a general algorithm to randomly generate subsets of a given finite set with a given probability distribution. To be specific, let $E = \{1, 2, \ldots, N\}$ be a given set and $\mathscr{B}$ be a given set of subsets of E. Let $w(B)$ be a positive weight associated with the subset $B \in \mathscr{B}$. The algorithm is designed to generate a subset $B \in \mathscr{B}$ with probability proportional to $w(B)$. The algorithm assumes that a perfect random number generator is available that generates independent uniform random numbers over the interval [0, 1].

We now describe several examples where such random subsets are needed. The first example comes from weighted sampling without replacement. Consider an urn containing N balls numbered $1, 2, \ldots, N$. Each trial consists of picking one ball from the urn. Probability of picking the ith ball

*This research was supported by the Air Force of Scientific Research under grant AFOSR-84-0140. The U.S. Government's right to retain a nonexclusive royalty-free license in and to the copyright covering this paper, for governmental purposes, is acknowledged.

is proportional to a fixed positive constant α_i. When a ball is drawn on a trial it is not available for drawing on subsequent trials. The experiment ends when $k(\leq N)$ balls are drawn. One needs to generate random outcomes (the sets of size k) from this experiment in statistical sampling plans. The problem fits our model if we take $\mathscr{B}$ to be the set of all subsets of $E = \{1, 2, \ldots, N\}$ with cardinality k and define the weight $w(B)$ of $B \in \mathscr{B}$ as

$$w(B) = \prod_{i \in B} \alpha_i.$$

The next situation arises in the simulation of a system consisting of N independent components. Component i operates (is up) with probability p_i and fails (is down) with probability $q_i = 1 - p_i$. Depending upon which components are up and which are down, the system itself either operates or fails. One common model for system operation is as follows: suppose the system can be represented as a network with the arcs representing the components. The system is said to operate if there is a path of operating arcs connecting two specified nodes s and t. This is called the s–t connectedness model. Since computing the probability of s–t connectedness is a NP hard problem (see Ball and Provan [2]) Monte Carlo simulation has been used as a method of estimating it (see Fishman [6]). In the simulation experiments one needs to generate a random set of operating components with the following distribution:

$$\begin{aligned} &P\{B \text{ is the set of operating components}\} \\ &= \Big(\prod_{i \in B} p_i\Big)\Big(\prod_{i \notin B} q_i\Big) \\ &= \Big(\prod_{i=1}^{N} q_i\Big)\Big(\prod_{i \in B} \frac{p_i}{q_i}\Big). \end{aligned}$$

Thus this problem fits our model with E as the set of arcs and $\mathscr{B} = 2^E$ and

$$w(B) = \prod_{i \in B} \alpha_i,$$

where $\alpha_i = p_i/q_i$. Note that the constant $\prod_{i=1}^{N} q_i$ need not be included in $w(B)$, since the probability of generating B is **proportional** to $w(B)$, not equal to $w(B)$.

Another model of system operation is called the all points connectedness. In this model the system is assumed to operate if all the nodes in its network representation are connected by paths of operating arcs. Again, computing the probability of all points connectedness is an NP hard problem (see Ball [1]) and hence recourse has been taken to simulation (see Nel and Colbourn [11]). In the Monte Carlo simulation one needs to generate random spanning trees in the network representation with the following distribution:

$$P\{A \text{ spanning tree } B \text{ is generated}\} = \prod_{i \in B} p_i.$$

(We think of a spanning tree B as being described by its arc set.) This problem also fits our model with E as the set of all arcs, $\mathscr{B}$ being the set of all spanning trees and

$$w(B) = \prod_{i \in B} p_i.$$

Tinhofer [13] considers the problem of generating random graphs from a set of graphs having a given property. We shall show that his algorithm is a special case of the algorithm developed here.

Our algorithm depends, in a crucial way, upon the ability to count the sum of the weights of the elements of special subsets of $\mathscr{B}$, called the intervals. If this computation can be done in $O(f(N))$ time, then the proposed algorithm will generate a random $B \in \mathscr{B}$ with the desired distribution in at most $O(Nf(N))$ time.

The algorithm is described in detail in the next section. We illustrate it with a simple example and prove that it works in general. In Sections 3–6, we apply the algorithm to several special cases: (i) $\mathscr{B} = 2^E$, (ii) $\mathscr{B} =$ subsets of E of cardinality k, (iii) $\mathscr{B} =$ spanning trees in graphs, and (iv) $\mathscr{B} =$ paths in a directed acyclic network. We also discuss various tractable weight functions. Some of these special-case algorithms have already appeared in literature, while some seem to be new. The paper ends with a few concluding remarks in Section 7.

2. The General Algorithm

Let $E = \{1, 2, \ldots, N\}$ be a given finite set and $\mathscr{B}$ be an arbitrary non-empty set of subsets of E, i.e., $\mathscr{B} \subseteq 2^E$. Let $w(B)$ be the weight of the subset $B \in \mathscr{B}$. Without loss of generality, we assume that $w(B) > 0$ for all $B \in \mathscr{B}$, since if $w(B) = 0$, the subset B will never be generated and hence may be deleted from $\mathscr{B}$. In this section we describe an algorithm that generates a random variable U that takes values in $\mathscr{B}$ such that

$$P\{U = B\} = \frac{w(B)}{\sum_{D \in B} w(D)} \qquad (B \in \mathscr{B}). \tag{2.1}$$

In other words, the algorithm generates random subsets of E that belong to $\mathscr{B}$ such that the probability of generating a given $B \in \mathscr{B}$ is proportional to $w(B)$.

EXAMPLE 1. Let $E = \{1, 2, 3\}$ and $\mathscr{B} = \{\{1\}, \{3\}, \{2, 3\}\}$. The weights and the probabilities are given in the following table:

B	$w(B)$	$P(U = B)$
$\{1\}$	4	0.16
$\{3\}$	16	0.64
$\{2, 3\}$	5	0.20

Thus the algorithm generates $B = \{2, 3\}$ with probability $5/(4 + 16 + 5) = 0.20$, etc.

Before we describe the algorithm we introduce the following notation. Let U and V be subsets of E (not necessarily in $\mathscr{B}$) such that $U \subseteq V$. (We use U as a subset of E and also as the random subset generated by the algorithm. This need not cause any confusion.) The interval $[U, V]$ is defined as the following set

$$[U, V] = \{B \in \mathscr{B}\colon U \subseteq B \subseteq V\}. \tag{2.2}$$

Also, define the weight of the interval $[U, V]$ as

$$w[U, V] = \sum_{B \in [U, V]} w(B). \tag{2.3}$$

The algorithm described below includes a statement "generate $Z \sim U[0, 1]$." Every time this statement is executed a random variable is generated having the following properties:

(i) it is uniformly distributed over the interval $[0, 1]$;

(ii) it is independent of all the random variables generated so far.

With these conventions we are now ready to describe the algorithm.

ALGORITHM A.

```
$U = \varnothing$, $V = E$, $i = 0$.
Do while ($i < N$);
        $i = i + 1$.
        Generate $Z \sim U[0,1]$.
        If $Z \le \dfrac{w[U \cup \{i\}, V]}{w[U, V]}$
        then $U = U \cup \{i\}$
        else $V = V - \{i\}$.
end.
Stop. $U$ is the desired random subset.
```

Before we give a proof of correctness of the algorithm we apply it to Example 1.

EXAMPLE 2. (Algorithm A applied to Example 1).

(0) $U = \phi$, $V = \{1,2,3\}$, $i = 0$.
$[U, V] = \{\{1\}, \{3\}, \{2,3\}\}$, $w[U, V] = 25$.
Go to (1).

(1) $i = 1$. $w[U \cup \{1\}, V] = 4$.
Generate $Z \sim U[0,1]$.
If $Z \leq 4/25$ go to (2), else go to (3).

(2) The probability of reaching this stage is $4/25$.
$i = 2$, $U = \{1\}$, $V = \{1,2,3\}$, $w[U, V] = 4$.
$w[U \cup \{2\}, V] = 0$.
Generate $Z \sim U[0,1]$.
If $Z \leq 0$ go to (4), else go to (5).

(3) The probability of reaching this stage is $21/25$.
$i = 2$, $U = \phi$, $V = \{2,3\}$, $w[U, V] = 21$.
$w[U \cup \{2\}, V] = 5$.
Generate $Z \sim U[0,1]$.
If $Z \leq 5/21$ go to (6), else go to (7).

(4) The probability of reaching this stage is 0.
Hence we shall not pursue it further.

(5) The probability of reaching this stage is $4/25$.
$i = 3$, $U = \{1\}$, $V = \{1,3\}$, $w[U, V] = 20$.
$w[U \cup \{3\}, V] = 0$.
Generate $Z \sim U[0,1]$.
If $Z \leq 0$ go to (8), else go to (9).

(6) The probability of reaching this stage is $\frac{21}{25} \times \frac{5}{21} = 5/25$.
$i = 3$, $U = \{2\}$, $V = \{2,3\}$, $w[U, V] = 5$.
$w[U \cup \{3\}, V] = 5$.
Generate $Z \sim U[0,1]$.
If $Z \leq 5/5$, go to (10), else go to (11).

(7) The probability of reaching this stage is $\frac{21}{25} \times \frac{16}{21} = \frac{16}{25}$.
$i = 3$, $U = \varnothing$, $V = \{3\}$, $w[U, V] = 16$.
$w[U \cup \{3\}, V] = 16$.
Generate $Z \sim U[0,1]$.
If $Z \leq 16/16$ go to (12), else go to (13).

(8) The probability of reaching this state is 0.
Hence we shall not pursue it further.

(9) Probability of reaching this stage is $4/25$
$U = \{1\}$, $V = \{1\}$. Stop. $\{1\}$ is the desired set.

(10) Probability of reaching this stage is $\frac{5}{25}$
$U = \{2,3\}$, $V = \{2,3\}$. Stop. $\{2,3\}$ is the desired set.
(11) Probability of reaching this stage is 0.
Hence we shall not pursue it further.
(12) Probability of reaching this stage is 16/25,
$U = \{3\}$, $V = \{3\}$. Stop. $\{3\}$ is the desired set.
(13) Probability of reaching this stage is 0.
Hence we shall not pursue it further.

From stages (9), (10), and (12) it is clear that the algorithm produces the set $\{1\}$ with probability 4/25, $\{2,3\}$ with probability 5/25, and $\{3\}$ with probability 16/25; as desired.

The exhaustive analysis of the algorithm in the above example should make it clear why the algorithm works. A formal proof is given below:

THEOREM 2.1. *Algorithm* A *produces a random variable U taking values in $\mathscr{B}$ with the distribution*

$$P\{U = B\} = \frac{w(B)}{\sum_{D \in \mathscr{B}} w(D)} \qquad (B \in \mathscr{B}). \tag{2.4}$$

Proof. Let U_i and V_i be the values of the sets U and V after the do loop in Algorithm A has been executed i times $(0 \le i \le N)$. Thus $U_0 = \varnothing$, $V_0 = E$.

Since $\mathscr{B}$ is assumed to be nonempty, $[U_0, V_0] = [\varnothing, E] = \mathscr{B}$ is also nonempty. As an induction hypothesis assume that $[U_i, V_i]$ is nonempty for some i, $0 \le i \le N$. From the algorithm it is clear that $i + 1 \in V - U$ and

$$[U_i, V_i] = [U_i \cup \{i+1\}, V_i] \cup [U_i, V_i - \{i+1\}] \tag{2.5}$$

and that $[U_i \cup \{i+1\}, V_i]$ and $[U_i, V_i - \{i+1\}]$ are disjoint. Hence

$$w[U_i, V_i] = w[U_i \cup \{i+1\}, V_i] + w[U_i, V_i - \{i+1\}]. \tag{2.6}$$

Thus the algorithm will generate

$$U_{i+1} = U_i \cup \{i+1\} \text{ and } V_{i+1} = V_i$$

$$\text{with probability } \frac{w[U_i \cup \{i+1\}, V_i]}{w[U_i, V_i]}$$

and

$$U_{i+1} = U_i \text{ and } V_{i+1} = V_i - \{i+1\}$$
$$\text{with probability } \frac{w[U_i, V_i - \{i+1\}]}{w[U_i, V_i]}. \tag{2.7}$$

In any case $w[U_{i+1}, V_{i+1}] > 0$, i.e., $[U_{i+1}, V_{i+1}] \neq \varnothing$. Hence, by induction, $[U_i, V_i] \neq \varnothing$ for all $0 \leq i \leq N$. In particular $[U_N, V_N] \neq \varnothing$. But clearly, $U_N = V_N$. Hence we must have $U_N \in \mathscr{B}$. Hence the algorithm terminates with an element of $\mathscr{B}$ as output.

Next we show that $P\{U_N = B\}$ satisfies (2.4). We have

$$P\{U_0 = \varnothing, V_0 = E\} = 1 = \frac{w[\varnothing, E]}{\sum_{D \in \mathscr{B}} w(D)}. \tag{2.8}$$

As an induction hypothesis suppose that

$$P\{U_i = U, V_i = V\} = \frac{w[U, V]}{\sum_{D \in \mathscr{B}} w(D)}. \tag{2.9}$$

From Eq. (2.7) we see that

$$P\{U_{i+1} = U \cup \{i+1\}, V_{i+1} = V | U_i = U, V_i = V\}$$
$$= \frac{w[U \cup \{i+1\}, V]}{w[U, V]} \tag{2.10}$$

and

$$P\{U_{i+1} = U, V_{i+1} = V - \{i+1\} | U_i = U, V_i = V\}$$
$$= \frac{w[U, V - \{i+1\}]}{w[U, V]}. \tag{2.11}$$

Hence, combining (2.9), (2.10), and (2.11) we obtain

$$P\{U_{i+1} = U, V_{i+1} = V\} = \frac{w[U, V]}{\sum_{D \in \mathscr{B}} w(D)}. \tag{2.12}$$

Thus by induction, (2.9) is true for all $i = 0, 1, \ldots, N$. Hence

$$\begin{aligned} &P\{\text{Algorithm A generates the set } B\} \\ &= P\{U_N = B, V_N = B\} \\ &= \frac{w[B, B]}{\sum_{D \in \mathscr{B}} w(B)} = \frac{w(B)}{\sum_{D \in \mathscr{B}} w(B)} \end{aligned}$$

as desired. □

The algorithm generates N $U(0,1)$ random variables. It is possible to modify the algorithm slightly so that it terminates when $[U, V]$ consists of a single element of $\mathscr{B}$. With this modification the algorithm will need at most N uniform random variables. Of course the complexity of the algorithm depends upon the complexity of the computation of $w[U, V]$. As mentioned in the introduction, if $w[U, V]$ can be computed in $O(f(N))$ time, the algorithm generates a desired random subset in at most $O(Nf(N))$ time.

Another feature of the algorithm that can be exploited for more efficient implementation is that the actual order in which elements of E get processed is immaterial. In effect, we can sequence the elements of E in any way we want. We can even resequence the unprocessed elements of E depending upon the (random) outcome of the processing of the processed elements. A good adaptive ordering of the elements of E can produce a more efficient implementation of the algorithm.

We next compare our algorithm to Tinhofer's algorithm [13]. He considers a graph G with node set $\{1, 2, \ldots, N\}$, and a set $\mathscr{B}$ of subgraphs of G having a given property (connectivity, for example). His algorithm uniformly generates subgraphs from $\mathscr{B}$. We describe it briefly below.

A graph can be described by its adjacency lists $A_1 A_2 \ldots A_N$, where A_j is the set of neighbors of node j. The algorithm generates a subgraph by generating its adjacency lists $A_{v_1} A_{v_2} \ldots A_{v_N}$, where $v_1 v_2 \ldots v_N$ is a random permutation of $12 \ldots N$. After generating $A_{v_1} A_{v_2} \ldots A_{v_{j-1}}$, the next A_{v_j} is generated according to the distribution

$$P\{A_{v_j} = A \,|\, A_{v_1} A_{v_2} \ldots A_{v_{j-1}}\} = \frac{N(A_{v_1} A_{v_2} \ldots A_{v_{j-1}} A)}{N(A_{v_1} A_{v_2} \ldots A_{v_{j-1}})}, \qquad (2.13)$$

where the numerator is the number of subgraphs in $\mathscr{B}$ which contain the adjacency lists $A_{v_1} A_{v_2} \ldots A_{v_{j-1}} A$ and the denominator is the number of subgraphs in $\mathscr{B}$ which contain the adjacency lists $A_{v_1} A_{v_2} \ldots A_{v_{j-1}}$.

The differences and similarities between Algorithm A and Tinhofer's algorithm should now be clear. Tinhofer's algorithm is specifically designed for subgraphs, while Algorithm A works for any $\mathscr{B}$. Tinhofer's algorithm generates subgraphs *uniformly* from $\mathscr{B}$, while Algorithm A can handle *any* distribution over $\mathscr{B}$. A more important difference is that Tinhofer's algorithm needs to generate the entire conditional distribution of A_{v_j} at iteration j, while Algorithm A only needs to decide whether the element j belongs to U or $\bar{V}$.

In the following sections we discuss several applications of Algorithm A to special cases where the computation of $w[U, V]$ is relatively easy.

3. Random Sets

Let $E = \{1, 2, \ldots, N\}$ and suppose $\mathscr{B} = 2^E$, the set of all subsets of E. Let α_i be a positive weight of $i \in E$. Suppose

$$w(B) = \prod_{i \in B} \alpha_i \qquad (B \in \mathscr{B}). \tag{3.1}$$

We want to randomly generate sets in $\mathscr{B}$ so that the probability of generating a subset $B \in \mathscr{B}$ is proportional to $w(B)$. In order to use Algorithm A we need the following lemma.

Lemma 3.1. *Let $U \subseteq V \subseteq E$. Then*

$$w[U, V] = \prod_{i \in U} \alpha_i \prod_{j \in V - U} (1 + \alpha_j). \tag{3.2}$$

For $i \in V - U$,

$$w[U \cup \{i\}, V] = \frac{\alpha_i}{1 + \alpha_i} w[U, V]. \tag{3.3}$$

Proof. Equation (3.2) follows from direct computation using

$$w[U, V] = \sum_{U \subseteq B \subseteq V} w(B).$$

Equation (3.3) follows from Eq. (3.2). □

Using the above lemma we get the following version of Algorithm A.

Algorithm A1.

```
$U = \varnothing$
Do $i = 1$ to $N$;
    Generate $Z \sim U[0, 1]$
    If $Z \leq \frac{\alpha_i}{1 + \alpha_i}$, set $U = U \cup \{i\}$
    end;
Stop. $U$ is the desired set.
```

Note that we do not explicitly keep track of the set V, since at the beginning of the ith iteration of the loop we have $V = E - (\{1, 2, \dots, i-1\} - U)$.

When $\alpha_i = 1$, we get $w(B) = 1$, for all $B \in \mathscr{B}$. Thus in this case the algorithm generates all subsets of E in a uniform fashion. When $\alpha_i = \alpha$ for all $i \in E$, the number of elements in the generated set is binomially distributed with parameters N and $\alpha/(1+\alpha)$. In this case Algorithm A1 yields the well-known method of generating binomial random variables (see Devroye [4]).

The weight function in Eq. (3.1), for obvious reasons, is called a multiplicative weight function. Next we consider an additive weight function,

$$w(B) = \sum_{i \in B} \alpha_i \qquad (B \in \mathscr{B}). \tag{3.4}$$

For this case we need the following lemma.

Lemma 3.2. *Let $U \subseteq V \subseteq E$. Then*

$$w[U, V] = 2^{|V-U|}\left[w(U) + \tfrac{1}{2}w(V-U)\right]. \tag{3.5}$$

For $i \in V - U$,

$$w[U \cup \{i\}, V] = \tfrac{1}{2}\left(w[U, V] + 2^{|V-U|-1}\alpha_i\right). \tag{3.6}$$

Proof. Let $B \in [U, V]$. Then B can be written as $U \cup B'$, where $B' \subseteq V - U$. Hence,

$$\begin{aligned} w[U, V] &= \sum_{B \in [U,V]} w(B) \\ &= \sum_{B' \subseteq V-U} w(U \cup B') \\ &= \sum_{B' \subseteq V-U} (w(U) + w(B')) \\ &= 2^{|V-U|}w(U) + \sum_{B' \subseteq V-U} w(B'). \end{aligned} \tag{3.7}$$

Now each α_i $(i \in V - U)$ appears in exactly half of the $2^{|V-U|}$ terms in the last sum in Eq. (3.7). Hence,

$$\begin{aligned} \sum_{B' \subseteq V-U} w(B') &= 2^{|V-U|}\tfrac{1}{2} \sum_{i \in V-U} \alpha_i \\ &= 2^{|V-U|}\tfrac{1}{2}w(V-U). \end{aligned} \tag{3.8}$$

Combining (3.7) and (3.8) yields (3.5). Manipulating (3.5) we get (3.6). □

The above lemma produces the following version of Algorithm A. In this algorithm the variable m contains the current value of $w[U, V]$. As before, we do not explicitly keep track of the set V.

ALGORITHM A2.

```
$U = \varnothing$, $m = 2^{N-1} \sum_{i=1}^{N} \alpha_i$.
Do $i = 1$ to $N$.
    Generate $Z \sim U[0,1]$.
    If $Z \le \frac{1}{2}(1 + 2^{N-i}\alpha_i/m)$
        then $\{U = U \cup \{i\},\ m = \frac{1}{2}(m + 2^{N-i}\alpha_i)\}$
        else $m = \frac{1}{2}(m - 2^{N-i}\alpha_i)$.
    End.
Stop. $U$ is the desired set.
```

When $\alpha_i = \alpha$ for all $i \in E$, the above algorithm generates random sets $B \in \mathscr{B}$ with probability proportional to $|B|$. In this special case the above algorithm can be simplified even further. We omit the details and only describe the algorithm below.

ALGORITHM A3.

```
$U = \varnothing$
Do $i = 1$ to $N$;
    Generate $Z \sim U[0,1]$
    If $Z \le \frac{1}{2}\frac{N - i + 2 + 2|U|}{N - i + 1 + 2|U|}$ then $U = U \cup \{i\}$.
    end.
Stop, $U$ is the desired set.
```

Note that Algorithms A1, A2, and A3 are all $O(N)$ algorithms. Next we consider another weight function. Let $f(k)$, $0 \le k \le N$, be a positive function of k. Let

$$w(B) = f(|B|), \qquad B \in \mathscr{B}. \tag{3.9}$$

The next lemma gives the required results about $w[U, V]$.

LEMMA 3.3. *Let $U \subseteq V \subseteq E$, with $m = |U|$, $n = |V|$. Then*

$$w[U,V] = g(m,n) = \sum_{k=m}^{n} \binom{n-m}{k-m} f(k). \tag{3.10}$$

Proof. The number of sets of cardinality k that belong to the interval $[U, V]$ is $\binom{n-m}{k-m}$ and the weight of each one of these sets is $f(k)$. Hence the lemma follows. □

We get the following algorithm when the above lemma is used in conjunction with Algorithm A. In the algorithm below, $m = |U|$ and $n = |V|$ at all times. Again, we need to keep track of U alone.

ALGORITHM A4.

```
$U = \emptyset$
$m = 0$, $n = N$.
Do $i = 1$ to $N$.
        Generate $Z \sim U[0,1]$.
        If $Z \le g(m+1, n)/g(m, n)$
                then $\{U = U \cup \{i\},\ m = m + 1\}$
        end.
Stop. $U$ is the desired set.
```

Algorithm A1 is well known in literature (see Devroye [4]). However, Algorithms A2, A3, and A4, to the best of author's knowledge, are new.

4. RANDOM SUBSETS OF SIZE k

Let $E = \{1,2,\ldots,N\}$ and $\mathscr{B} = \{B \subseteq E: |B| = k\}$, where k is a fixed number such that $0 \le k \le N$. Let $w(B)$ be the multiplicative weight function defined by Eq. (3.1) (restricted to $B \in \mathscr{B}$). As before, we want to randomly generate subsets $B \in \mathscr{B}$ such that the probability of generating B is proportional to $w(B)$. The relevant results about $w[U, V]$ are given in the lemma below. We need the following notation. For $U \subseteq E$ with $|U| = m$, let $H(U)(x)$ be the mth degree polynomial defined

$$H(U)(x) = \prod_{i \in U} (1 + \alpha_i x) = \sum_{r=0}^{m} a_r(U) x^r. \tag{4.1}$$

LEMMA 4.1. *Let $U \subseteq V \subseteq E$ and $m = |U|$. Then, assuming $m \le k \le n = |V|$,*

$$w[U, V] = w(U) a_{k-m}(V - U). \tag{4.2}$$

For $i \in V - U$,

$$w[U \cup \{i\}, V] = \alpha_i a_{k-m-1}(V - U \cup \{i\})/a_{k-m}(V - U). \tag{4.3}$$

Proof. It is clear by direct multiplication that, for $0 \le r \le m$,

$$a_r(U) = \sum_{B} \prod_{i \in B} \alpha_i = \sum_{B} w(B), \tag{4.4}$$

where the sum is taken over all $B \subseteq U$ with $|B| = r$. Now, a $B \in [U, V]$

can be written as $B = B' \cup U$, where $B' \in V - U$ and $|B'| = k - m$. Conversely, if $B' \in V - U$ and $|B'| = k - m$ then $B' \cup U \in [U, V]$. Hence, letting $\mathscr{B}' = \{B' \subseteq V - U, |B'| = k - m\}$,

$$\begin{aligned} w[U,V] &= \sum_{B \in [U,V]} w(B) \\ &= \sum_{B' \in \mathscr{B}'} w(B' \cup U) \\ &= w(U) \sum_{B' \in \mathscr{B}'} w(B') \\ &= w(U) a_{k-m}(V - U), \end{aligned}$$

as desired. The last equality follows from Eq. (4.4). Equation (4.3) follows directly from (4.2). □

It is possible to compute $a_r(U)$ $(0 \le r \le k)$ in $O(k|U|)$ time as follows:

Algorithm to Compute $a_r(U)$.

```
$a_0(U) = 1$, $a_r(U) = 0$ $(1 \le r \le k)$.
For each $i \in U$, do the following:
Do $r = 1$ to $k$
        $a_r(U) = a_r(U) + \alpha_i a_{r-1}(U)$.
end; end;
Stop. $a_r(U)$ $(0 \le r \le k)$ are the desired coefficients.
```

Using Eqs. (4.2) and (4.3) we obtain the $O(kN^2)$ algorithm as a special case of Algorithm A.

Algorithm A5.

```
$U = \varnothing$, $V = E$, $i = 0$, $m = 0$
Do while $(m < k)$
        $i = i + 1$
        Generate $Z \sim U[0,1]$
        If $Z \le \alpha_i a_{k-m-1}(V - U \cup \{i\}) / a_{k-m}(V - U)$
                then $\{U = U \cup \{i\}, m = m + 1\}$
                else $V = V - \{i\}$
        end.
        Stop. $U$ is the desired set.
```

It is possible to construct a more efficient version of Algorithm A5 as follows: In the above algorithm the set $V - U$ is replaced by the set $(V - U) - \{i\}$ after the ith execution of the loop. Hence, if we can compute $a_r(U - \{i\})$ $(0 \le r \le k)$ from $a_r(U)$ $(0 \le r \le k)$ in an efficient fashion, the efficiency of the algorithm will increase. It is indeed possible to do so as

$$a_0(U - \{i\}) = a_0(U)$$
$$a_r(U - \{i\}) = a_r(U) - a_{r-1}(U - \{i\})\alpha_i \qquad 1 \le r \le k.$$

With this improvement we get the following version of Algorithm A5. Now we do not need to keep track of V.

ALGORITHM A6.

```
$U = \varnothing$, $m = 0$, $i = 0$.
Let $a_r$ be the coefficient of $x^r$ in $H(E)(x)$, $0 \le r \le k$.
Do while $(m < k)$
        $i = i + 1$
        $b_0 = a_0$; $b_r = a_r - a_{r-1}\alpha_i$,    $1 \le r \le k - m$;
        Generate $Z \sim U[0,1]$
        If $Z \le \alpha_i b_{k-m-1}/a_{k-m}$
                then $\{U = U \cup \{i\}, m = m + 1\}$
        $a_r = b_r$, $0 \le r \le k - m$.
        end.
Stop. $U$ is the desired set.
```

It is easy to see that the above algorithm is $O(kN)$. Several algorithms exist in literature to solve this problem. We mention a few: Fagin and Price [5] have an $O(kN)$ algorithm in which, at each step, an index i is chosen from $E - U$ to be added to U. This step is repeated k times. Wong and Easton [14] describe an $O(N + k \log N)$ procedure for the same problem. The improvement in efficiency in their algorithm comes from using sophisticated data structures. Ross and Schechner [12] describe another $O(N + k \log N)$ procedure for this problem. Their procedure uses special properties of exponential distribution and Poisson processes.

When $\alpha_i = 1$ for all $i \in E$, then the algorithm generates a random subset of size k, in a uniform fashion. In this case we can use the following results to improve Algorithm A5:

$$H(U)(x) = \sum_{r=0}^{m} \binom{m}{r} x^r \qquad (m = |U|) \tag{4.5}$$

$$w[U, V] = \binom{n - m}{k - m} \qquad (n = |V|,\ m = |U|,\ m \le k \le n) \tag{4.6}$$

$$w[U \cup \{i\}, V] = \frac{k - m}{n - m} w[U, V] \qquad (n \ne m). \tag{4.7}$$

All these results follow as special cases of Lemma 4.1. Using these results we get the following version of Algorithm A5. We use $m = |U|$ and $n = |V|$.

ALGORITHM A7.

```
$U = \varnothing$, $m = 0$, $n = N$, $i = 0$.
Do while $(m < k)$
      $i = i + 1$
      Generate $Z \sim U[0,1]$
      If $Z \le \frac{k - m}{n - m}$
            then $\{U = U \cup \{i\},\ m = m + 1\}$
            else $n = n - 1$.
      end.
Stop. $U$ is the desired set.
```

This is an $O(N)$ algorithm and is identical to algorithm S of Section 3.4.2 of Knuth [8].

All the algorithms in this section are for the multiplicative weight function. Similar algorithms can be developed for the additive weight function. We only give the results about $w[U, V]$ and skip the algorithms.

LEMMA 4.2. *For $B \in \mathscr{B}$ let*

$$w(B) = \sum_{i \in B} \alpha_i. \tag{4.8}$$

Let $U \subseteq V \subseteq E$ with $|U| = m \le k \le n = |V|$. Then

$$w[U, V] = \binom{n - m}{k - m} w(U) + \binom{n - m - 1}{k - m - 1} w(V - U). \tag{4.9}$$

Proof. Following the same argument as in the proof of Lemma 4.1 we obtain

$$\begin{aligned} w[U, V] &= \sum_{B' \in \mathscr{B}'} w(U \cup B') \\ &= \sum_{B' \in \mathscr{B}'} (w(U) + w(B')) \\ &= |\mathscr{B}'| w(U) + \sum_{B' \in \mathscr{B}'} \sum_{i \in B'} \alpha_i. \end{aligned}$$

Now each $i \in V - U$ appears in exactly $\binom{n - m - 1}{k - m - 1}$ sets $B' \in \mathscr{B}'$, and

$|\mathscr{B}'| = \binom{n-m}{k-m}$. Hence

$$w[U, V] = \binom{n-m}{k-m} w(U) + \binom{n-m-1}{k-m-1} \sum_{i \in V-U} \alpha_i.$$

Equation (4.9) follows from this. □

5. Random Spanning Trees

Let $G = (\mathscr{V}, E)$ be an undirected network of M nodes and N arcs. A spanning tree in G is a connected subgraph of G with M nodes and $M - 1$ arcs. It is completely described by its constituent arcs, hence we shall think of a spanning tree as a set of arcs. Let $\mathscr{B}$ be the set of all spanning trees in G. Let α_i be the positive weight of arc $i \in E$. Defined the weight $w(B)$ of a spanning tree $B \in \mathscr{B}$ as

$$w(B) = \prod_{i \in B} \alpha_i. \tag{5.1}$$

Also define

$$n(G) = \sum_{B \in \mathscr{B}} w(B). \tag{5.2}$$

In this section we describe an algorithm to generate $B \in \mathscr{B}$ so that

$$P\{B \text{ is generated}\} = \frac{w(B)}{n(G)}. \tag{5.3}$$

First we develop a method of computing $n(G)$. Let $E(i, j)$ be the set of all arcs incident on both nodes i and j and let $E(i)$ be the set of arcs incident on node i. (Note that there may be parallel arcs in G. We assume, however, that G contains no self-loops.) For the network G, construct an M by M symmetric matrix $D(G)$ as follows:

$$[D(G)]_{ij} = \begin{cases} -\sum_{r \in E(i,j)} \alpha_r & \text{if } i \neq j \\ \sum_{r \in E(i)} \alpha_r & \text{if } i = j. \end{cases} \tag{5.4}$$

Lemma 5.1. *All cofactors of $D(G)$ (determinant of $M - 1 \times M - 1$ submatrices of $D(G)$ obtained by deleting ith row and column for any $i \in \mathscr{V}$) are equal and their common value is equal to $n(G)$.*

Proof. Construct a matrix P of size M by N as follows:

$$[P]_{ir} = \begin{cases} -\alpha_r & \text{if arc } r \text{ joins node } i \text{ to some node } j > i \\ \alpha_r & \text{if arc } r \text{ joins node } i \text{ to some node } j < i \\ 0 & \text{if arc } r \text{ is not incident on node } i. \end{cases} \tag{5.5}$$

Let Q be another matrix of size M by N which is obtained from P by setting $\alpha_r = 1$ for all $r \in E$. It is easy to see by direct multiplication that

$$D(G) = PQ^{\mathrm{T}} \tag{5.6}$$

where Q^{T} denotes the transpose of Q. Consider the $M-1 \times M-1$ submatrix D_1 of $D(G)$ obtained by deleting the first row and first column of $D(G)$. Let P_1 and Q_1 be the $M-1 \times N$ matrices obtained by deleting the first row of P and Q, respectively. Then

$$D_1 = P_1 Q_1^{\mathrm{T}}. \tag{5.7}$$

From Binet–Cauchy theorem (see Harary [7, p. 153]),

$$\begin{aligned} \det(D_1) &= \det(P_1 Q_1^{\mathrm{T}}) \\ &= \sum \det(RS^{\mathrm{T}}), \end{aligned} \tag{5.8}$$

where the sum is taken over all R such that R is a $M-1 \times M-1$ submatrix of P_1 and S is a corresponding $M-1 \times M-1$ submatrix of Q_1. (Corresponding means that the same columns are deleted to obtain S from Q_1 as for R from P_1.) Now it is easy to see that

$$\det(RS^{\mathrm{T}}) = \begin{cases} w(B) & \text{if the columns of } R \text{ form a spanning tree } B \in \mathscr{B}, \\ 0 & \text{otherwise.} \end{cases} \tag{5.4}$$

Hence the theorem follows. □

The above theorem yields an efficient way of computing $w[U, V]$, as is given in the following lemma.

LEMMA 5.2. *Let $U \subseteq V \subseteq E$. Suppose U is a forest of trees in G. Let $G(U, V)$ be a subgraph of G obtained by deleting arcs that are not in V, and collapsing arcs that are in U (i.e., identifying the end nodes of arcs in U) and deleting all self-loops resulting from these deletions and collapsing. Then*

$$w[U, V] = n(G(U, V))\Big(\prod_{i \in U} \alpha_i\Big). \tag{5.10}$$

Proof. We have $[U, V]$ = the set of all spanning trees in G that contain all the arcs in U and none of the arcs in $E - V$. Now let B be a spanning tree in $G(U, V)$. Then $B \cup U$ is a spanning tree in G. Conversely, if B is a spanning tree in $G(U, V)$ then $B - U$ is a spanning tree in $G(U, V)$. Hence, with $\mathscr{B}'$ as the set of all spanning trees in $G(U, V)$

$$\begin{aligned} w[U, V] &= \sum_{B \in [U, V]} w(B) \\ &= \sum_{B' \in \mathscr{B}'} w(U \cup B') \\ &= \left(\prod_{i \in U} \alpha_i\right) \sum_{B' \in \mathscr{B}'} w(B') \\ &= \left(\prod_{i \in U} \alpha_i\right) n(G(U, V)). \square \end{aligned}$$

With the help of above lemma, we can now specialize Algorithm A to the spanning tree problem.

ALGORITHM A8.

```
$U = \varnothing$, $V = E$.
Do $i = 1$ to $N$;
        Let $a = n(G(U, V))$
            $a' = n(G(U \cup \{i\}, V))$
        Generate $Z \sim U[0, 1]$
        If $Z \le \alpha_i a'/a$
                then $U = U \cup \{i\}$,
                else $V = V - \{i\}$
        end.
Stop. $U$ is the required spanning tree.
```

Note that computation of a and a' is an $O(M^3)$ step. Hence the above algorithm is $O(M^3 N)$. When $\alpha_i = 1$ for all $i \in E$, the above algorithm generates spanning trees of G in a uniform fashion. This special case has been treated in Kulkarni [10] and Nel and Colbourn [11]. In another paper Colbourn *et al.* [3] provide a compact hash function for all spanning trees in G. Computing this hash function and its inverse is an $O(M^3)$ problem. Hence, if one can generate a random integer over the interval 1 to $n(G)$ = number of spanning trees in G, this hash function yields an efficient method of uniformly generating random spanning trees. However, typically $n(G)$ is a very large number and naive random number generation techniques on finite precision machine would require $O(M \log M)$ steps to generate a desired random integer. Thus using the hash function approach may not be very suitable for large problems. Second, extending the hash function approach to non-uniform generation seems problematic. When all $\alpha_i = 1$ and the graph is a complete graph, Kulkarni [10] also gives an $O(N)$ algorithm for this problem. When α_i are not all equal, such a simplification does not seem possible for the complete graph case.

Extension of the above algorithm to rooted spanning trees in directed networks is straightforward. We only give the main result below. Let G be a directed network. Let $E(i, j)$ be the set of all arcs with tail node i and head node j and $E(i)$ be the set of all arcs with tail node i. Consider node 1 to be the root and let $\mathscr{B}$ be the set of all rooted spanning trees in G. Let $w(B)$ be the product weight function as before. Define a $M \times M$ matrix $D(G)$ as

$$[D(G)]_{ij} = \begin{cases} -\sum\limits_{r \in E(i, j)} \alpha_r & \text{if } i \neq j \\ \sum\limits_{r \in E(i)} \alpha_r & \text{if } i = j. \end{cases} \tag{5.11}$$

Let $n(G)$ be as defined in (5.2).

LEMMA 5.3. *Let D_1 be the $M - 1$ by $M - 1$ submatrix obtained from $D(G)$ by deleting its first row and column. Then*

$$n(G) = \det(D_1). \tag{5.12}$$

Proof. Omitted. □

LEMMA 5.4. *Let U be a subset of arcs of a rooted spanning tree in G. Let $U \subseteq V \subseteq E$. Let $G(U, V)$ be the subgraph of G as defined in Lemma 5.2. Then*

$$w[U, V] = \Big(\prod_{i \in U} \alpha_i\Big) n(G(U, V)). \tag{5.13}$$

Proof. Omitted. □

Using Lemma 5.3 to compute $n(G)$, and Lemma 5.4, we see that Algorithm A8 can generate rooted spanning trees in G.

So far we have discussed the product weight function. All the results can be extended to the additive weight function. We briefly mention the relevant results below.

Let $G = (\mathscr{V}, E)$ be an undirected network. Let α_i be the positive weight of arc $i \in E$. Let B be a spanning tree in G. Define

$$w(B) = \sum_{i \in B} \alpha_i \qquad (B \in \mathscr{B}). \tag{5.14}$$

Let $t(G)$ be the number of spanning trees in G and let

$$n(G) = \sum_{B \in \mathscr{B}} w(B). \tag{5.15}$$

Then it is easy to see that

$$n(G) = \sum_{i=1}^{N} \alpha_i t(G(\{i\}, E)) \tag{5.16}$$

and

$$n(G(U,V)) = \sum_{i \in V-U} \alpha_i t(G(U \cup \{i\}, V)). \tag{5.17}$$

It can be easily seen that

$$w[U,V] = \left(\sum_{i \in U} \alpha_i\right) t(G(U,V)) + n(G(U,V)). \tag{5.18}$$

These results produce an $O(M^4N)$ algorithm to generate random spanning trees with additive weight function.

6. Random Paths in Directed Acyclic Networks

Let $G = (V, A)$ be a directed acyclic network with a set of nodes $V = \{1, 2, \ldots, N\}$ and a set of arcs $A \subseteq V \times V$. Let s and t be two prespecified nodes. An (s,t) path B in G is a sequence of nodes $\{v_0, v_1, \ldots, v_k\}$ such that $v_0 = s$, $v_k = t$, and $(v_i, v_{i+1}) \in A$ for $i = 0, 1, \ldots, k-1$. Since G is acyclic, it is clear that no node appears more than once in B. Let $\mathscr{B}$ be the set of all (s,t) paths in G. Let $w(u,v)$ be a positive weight of arc $(u,v) \in A$. Define the weight of path $B \in \mathscr{B}$ as

$$w(B) = \sum_{i=0}^{k-1} w(v_i, v_{i+1}). \tag{6.1}$$

In this section we develop an algorithm to randomly generate paths from $\mathscr{B}$ in such a way that the probability of generating a path $B \in \mathscr{B}$ is proportional to $w(B)$. Note that (6.1) defines an additive weight function.

Assume that nodes are indexed in a topological order, i.e., $(u,v) \in A \Rightarrow u < v$. Now define, for $i \in V$,

$$n(i) = \text{number of } (i,t) \text{ paths in } G$$
$$d(i) = \text{sum of the weights of all the } (i,t) \text{ paths in } G.$$

It is easy to compute $n(i)$ and $d(i)$ recursively as shown below. Without loss of generality we assume $s = 1, t = N$.

ALGORITHM TO COMPUTE $n(i)$.

```
$n(N) = 1$
Do $i = N - 1$ to 1 by $-1$.
    $n(i) = \sum_{j: (i,j) \in A} n(j)$
    end.
```

ALGORITHM TO COMPUTE $d(i)$.

```
$d(N) = 0$.
Do $i = N - 1$ to 1 by $-1$
    $d(i) = \sum_{j: (i,j) \in A} [w(i, j)*n(j) + d(j)]$
    end.
```

Instead of deriving the expressions for $w[U, V]$ we directly give the algorithm. The version presented here is closer to Tinhofer's algorithm than to Algorithm A, but the differences are minor. It is easy to see that it is an $O(N)$ algorithm.

ALGORITHM A9.

```
$U = \{1\}, k = 1, i_1 = 1$.
Do while $(N \notin U)$
    Let $\{j_1, j_2, \ldots, j_r\}$ be nodes in $V$ such that $(i_k, j_m) \in A$ for $m = 1, 2, \ldots, r$.
    Generate an index $m \in \{1, 2, \ldots, r\}$
    with probability $d(j_m) / \sum_{n=1}^{r} d(j_n)$
    Set $k = k + 1$, $i_k = j_m$, $U = U \cup \{i_k\}$.
    end.
Stop. $U$ is the desired path.
```

Next we consider the product weight function:

$$w(B) = \prod_{i=0}^{k-1} w(v_i, v_{i+1}).$$

In this case the quantities $n(i)$ are not needed. The computation of $d(i)$ takes the form:

```
$d(N) = 1$
Do $i = N - 1$ to 1 by $-1$
$d(i) = \sum_{j: (i,j) \in A} w(i, j)*d(j)$
end.
```

Algorithm A9 remains unchanged with this modified computation of $d(i)$. Again, the algorithm is $O(N)$. When all $w(i, j) = 1$, the product weight function makes $w(B) = 1$ for all $B \in \mathscr{B}$ and thus in this case Algorithm A9 produces random (s, t) paths uniformly from $\mathscr{B}$. Another $O(N)$ algorithm, based on compact hash functions, is described in Kulkarni [9]. When the number of paths in the network gets very large, the algorithms based upon hash functions do not work well, while Algorithm A9 works correctly for all N.

7. Conclusions

In this paper we have developed an algorithm to randomly generate subsets of a finite set according to a given probability distribution. Such generation techniques are needed in simulation of reliability systems, statistical sampling procedures, etc. The technique developed here depends upon the ability to compute the probability that the generated set lies in a given interval of sets. The technique is then applied to the special case of generating all subsets, subsets of size k, spanning trees, and paths. Some of the special algorithms have appeared in literature before, while some algorithms are reported here for the first time.

Acknowledgments

I thank Professor G. S. Fishman who stimulated my interest in this problem. I also thank Professor J. S. Provan for helpful comments and bringing the Binet–Cauchy theorem to my attention.

References

1. M. O. Ball, The complexity network reliability computations, *Networks* **10** (1980), 153–165.
2. M. O. Ball, and J. S. Provan, Calculating bounds on reachability and connectedness in stochastic networks, *Networks* **13** (1983), 253–278.
3. C. J. Colbourn, R. P. J. Day, and L. D. Nel, "Unranking and Ranking Spanning Trees of a Graph," Research Report CORR 88-11, University of Waterloo, Ontario, Canada, 1988.
4. L. Devroye, "Nonuniform Random Variate Generation," Springer-Verlag, New York, 1986.
5. R. Fagin and T. G. Price, Efficient calculation of expected miss ratios in the independent reference model, *SIAM J. Comput.* **7** (1978), 288—297.
6. G. S. Fishman, A Monte Carlo sampling plan for estimating network reliability, *Oper. Res.* **34** (1986), 581–594.
7. F. Harary, "Graph Theory," Addison-Wesley, Reading, MA, 1972.

8. D. E. KNUTH, "The Art of Computer Programming, Vol. 2. Seminumerical Algorithms," Addison-Wesley, Reading, MA, 1980.

9. V. G. KULKARNI, A compact hash function for paths in PERT networks, *Oper. Res. Lett.* **3** (1984), 137–140.

10. V. G. KULKARNI, "Generating Random Spanning Trees," Technical Report No. UNC/OR/88-1, Department of Operations Research, UNC, Chapel Hill, NC 27599, 1985.

11. L. D. NEL AND C. J. COLBOURN, "Combining Monte Carlo Estimates and Bounds for Network Reliability," Technical Report No. UW/ICR 88-03, University of Waterloo, Waterloo, Ontario, Canada, 1988.

12. S. M. ROSS AND Z. SCHECHNER, "Simulation Uses of the Exponential Distribution," Technical Report No. ORC-84-6, Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA, 1984.

13. G. TINHOFER, On the generation of random graphs with given properties and known distribution, *in* "Graphs, Data Structures, Algorithms" (M. Nagl, Ed.), pp. 265–297.

14. C. K. WONG AND M. C. EASTON, An efficient method for weighted sampling without replacement, *SIAM J. Comput.* **9** (1980), 111–113.