Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

GENERATION OF RANDOM OBJECTS

Luc Devroye

School of Computer Science
McGill University
Montreal, Canada H3A 2A7

ABSTRACT

We illustrate the paradigm that various random objects defined in terms of random processes can be generated quite efficiently without actually "running" or "simulating" the defining random process. Examples include the generation of sums of independent random variables, of random trees, of random convex hulls, and of absorption times in finite Markov chains. A simple method with one design parameter is presented.

1 INTRODUCTION

Generating random variables is like creating chaos from nothing. It is the world in reverse: we are used to looking at data, creating models, estimating parameters, and molding data in terms of well-defined functions. In all these endeavours, some degree of determinism is squeezed out of random data, often at tremendous computational costs. The inverse process should thus be simpler as we create disorder, increase the temperature and descend into the hell of random objects. Computationally, it should thus be easier to create such random objects than to derive distributional and other information from existing random objects. A random object (such as a random variable, a random vector, a random process, a random graph, a random set, and so forth) can be described in a myriad of ways. Consider for example a random variable X with a mixture density

$$f(x) = \sum_{i=1}^{n} p_i f_i(x) ,$$

where the p_i's form a probability vector, and the f_i's are known densities. X can be generated as X_N, where N is a random integer with $\mathbf{P}\{N = i\} = p_i$, and X_N has density f_N. The random integer N can be generated in time $O(1)$ if we allow preprocessing, using either Walker's method (Walker, 1974, 1977; Kronmal and Peterson, 1979; Peterson and Kronmal, 1983) or the guide table method (Chen and Asau, 1974; Fishman and Moore, 1984). If, e.g., the f_i's are normal densities with different means and variances, X_N can thus be generated in constant time. Take now the reverse viewpoint: given x, how fast can we compute $f(x)$ (exactly, of course)? It seems that we cannot avoid a time complexity that grows at least linearly with n. Thus, deterministic information is dramatically more expensive than random variate generation for this distribution.

We did not have to work hard on the finite mixture example. Let us briefly exhibit an example in which random variate generation seems to be challenging even though the description of the distribution is simple. Consider a random variable X with a symmetric stable distribution of parameter $\alpha \in (0, 2]$: it is uniquely determined by its characteristic function $\varphi(t) = \exp(-|t|^\alpha)$. The computation of $\varphi(t)$ is trivial; it seems difficult to beat. It was not until 1976 that we were even presented with an exact generator (see Chambers, Mallows and Stuck, 1976). Later (see for example, Devroye, 1982, 1986) more efficient methods were proposed, such as the one in which X is generated as Y/Z, where $Z = (E_1 + E_2 I_{[U<\alpha]})^{1/\alpha}$, E_1, E_2 are independent exponential random variables, U is a uniform $[0,1]$ random variable, and Y has the de la Vallée-Poussin density $(1/(2\pi))(\sin(x/2)/(x/2))^2$ and can be generated quite easily via the rejection method.

Still, even now, there are serious challenges ahead. In general, there are no satisfactory methods for infinitely divisible distributions when described via their Kolmogorov canonical representation for example: given is a nondecreasing function K with $K(-\infty) = 0$ (see Durrett, 1991), and the characteristic function is given by

$$\log \varphi(t) = i\mu t + \int_{-\infty}^{\infty} \frac{e^{itx} - 1 - itx}{x^2}\, dK(x) ,$$

where the parameter μ is a shift parameter. With $dK(x) = \sigma^2\delta(x)$, we obtain the normal distribution with variance σ^2. For $dK(x) = \lambda\delta(x-1)$, we have the Poisson ($\lambda$) distribution. The general K case remains a serious challenge. An approximate generator was proposed by Bondesson (1982). But even for particular K, simple random variate generators seem elusive, despite simple descriptions. An example is given by

$$\log\varphi(t) = it + \int_0^1 \frac{e^{itx} - 1 - itx}{x}\,dx ,$$

which describes the limit distribution of $(1/n)\sum_{i=1}^n X_i$, where the X_i's are independent, and $X_i = i$ with probability $1/i$, and $X_i = 0$ otherwise. Of course, strictly speaking, $\varphi(t)$ takes an infinite amount of time to compute, so the description of the distribution is not computationally simple.

In this note, we will take one paradigm, that of simulating quantities defined in terms of random processes without actually running the random process. This will be illustrated on the time until absorption in Markov chains, on random convex hulls, on sums of i.i.d. random variables, and on random trees.

2 ABSOPRPTION TIMES IN MARKOV CHAINS

2.1 Naive simulation.

Consider an N-state discrete-time Markov chain with starting state X_0, and realization $X_1, X_2, \ldots$. Let P be the $N \times N$ matrix of transition probabilities $p_{ij} = \mathbf{P}\{X_{n+1} = j | X_n = i\}$. We wish to simulate the time T until absorption in one of a number of absorbing states, or, the time until we first reach one of the states in a designated collection of states. For simulation purposes, we can always modify the transition probabilities to make any subset of states absorbing, so both problems are in fact equivalent. Needless to say, if we have a good grip on the absorption time simulation, we will in fact have solved a fundamental problem in Markov chain simulation. In a renewal process framework, we can simulate the time of first return to the starting state X_0 by generating the first transition (to X_1), and then generating the time until we first reach X_0 from X_1.

For simulating Markov chains, one typically constructs N alias tables (Walker, 1974, 1977; Kronmal and Peterson, 1979) or N guide tables (Chen and Asau, 1974; Fishman and Moore, 1984), one per state. This allows one to generate transitions in constant expected time. Deleting multiplicative constants, the total expected cost of such a direct simulation is $\mathbf{E}T + N^2$, where T is the time until absorption. The quadratic term accounts for the set-up of the alias or guide tables. This is the standard method one uses to proceed; see, e.g., the survey in Popescu and Vaduva (1991). Additional improvements are possible if one assumes a certain structure on the transition matrix, such as when the transition probability vectors for the states do not differ much (see for example Fox (1990, 1991) or Fishman and Yarberry (1990)). If one is just interested in the generation of T and not the sequence of states per se, then additional savings are possible. The purpose of this section is to explore this possibility.

The waiting time W until absorption in a continuous-time Markov chain, in which the distribution of the waiting time until a transition is exponentially distributed is simply gamma (T), where T is the waiting time until absorption in a discrete-time Markov chain with the same transition matrix. Random variables of this form are called phase type random variables. Neuts and Pagano (1981) generate such random variables by running the Markov process. For more discussion on this point, and minor improvements, see Devroye (1986, pp. 757–758). The vastness of the problem we are facing can be grasped by the following result: given any density f on the positive halfline, and any $\epsilon > 0$, there exists a continuous-time finite-state Markov chain with a starting state and an absorbing state such that the density f_T of T, the time until absorption, satisfies $\int |f - f_T| < \epsilon$. Thus, the family of phase-type distributions is phenomenally rich.

2.2 An Adaptive Algorithm: Acceleration, Deceleration

Introduce the matrix P^k with entries

$$p_{ij}^{(k)} = \mathbf{P}\{X_k = j | X_0 = i\} .$$

We propose in a first step, the acceleration phase, to generate the sequence $X_0, X_1, X_2, X_4, X_8, \ldots, X_{2^k}$, where X_{2^k} is the first time an absorbing state is reached. This can be achieved in $O(N^2)$ space and $O(kN^3)$ time, since at every iteration, we square the last transition matrix and recompute the alias tables. Because $2^{k-1} < T \leq 2^k$, we see that this method takes expected time bounded from above by a constant times $1 + N^3\mathbf{E}\log_2 T$.

In the deceleration phase, we put $l = 2^{k-1}$, $r = 2^k$, and repeat the following steps:

```
while $r - l > 1$ do
```

```
        $m \leftarrow (r+l)/2$
        generate $X_m$ (given $X_l, X_r$)
        if $X_m$ is absorbing
                then $r \leftarrow m$
                else $l \leftarrow m$
return $T \leftarrow r$
```

In the generation of X_m given X_l and X_r, we must in fact use a matrix of transition probabilities corresponding to a conditional Markov chain (Seneta, 1981) in which we have the following transition probabilities:

$$\begin{aligned}\mathbf{P}\{X_m = i \mid X_l = j, X_r = s\} \\ = \frac{\mathbf{P}\{X_m = i, X_r = s \mid X_l = j\}}{\mathbf{P}\{X_r = s \mid X_l = j\}} \\ = \frac{p_{ji}^{(m-l)} p_{is}^{(r-m)}}{p_{js}^{(r-l)}} .\end{aligned}$$

The matrix of probabilities only requires the matrices P^{r-l} and $P^{(r-l)/2}$. Thus, we could naively store the matrices $P, P^2, \ldots, P^k$ that were computed during the acceleration phase. Every step in both phases requires $O(N^3)$ time for transition matrix computations. The total number of steps is bounded by k, as the original interval is of size 2^{k-1}. The conditional Markov chain run until absorption costs no more than kN^3 in set-up time for the matrices and tables, kN for computing the appropriate transition probabilities, and $O(k)$ on average for actually generating the conditional Markov chain. The time complexity of deceleration is $O(N^3 \log_2 T)$. This nicely balances out the acceleration step. The method given here is reminiscent of binary search, as each time the interval size is halved. The overall expected time is thus bounded by

$$C_1 + C_2 N^3 \mathbf{E} \log_2 T$$

where C_1 and C_2 are universal constants. This is better than straightforward simulation when

$$N^3 \mathbf{E} \log_2 T < \max(N^2, \mathbf{E}T) .$$

Roughly speaking, this happens when

$$N^3 < \frac{\mathbf{E}T}{\mathbf{E}\log_2 T} .$$

The total expected storage is $O(N^2 \mathbf{E} \log_2 T)$. This is only slightly more than N^2.

3 RANDOM CONVEX HULLS

Let $X_1, \ldots, X_n$ be i.i.d. random vectors in the plane, and let C_n be a (random) subset of indices from $\{1, \ldots, n\}$ corresponding to those X_i's that are on the convex hull of the n data points. In computational geometry and in pattern recognition, one works frequently with convex hulls of data sets. It is thus natural to ask how we can generate a random convex hull efficiently. The above definition of a random convex hull has the drawback that the cardinality of the convex hull, $|C_n|$, is also random. Nevertheless, for many distributions of X_1, $|C_n|$ is rather concentrated about its mean, $\mathbf{E}|C_n|$, and thus nearly deterministic.

The method based upon the definition would call for the generation of the data and the application of a convex hull finding algorithm, which would typically lead to a time complexity of $\Theta(n \log n)$ (see, e.g., Preparata and Shamos, 1985). A small improvement follows if we generate X_1, X_2 and X_3, denote the geometric center by X_c, generate $X_4, \ldots, X_n$ in angular order with respect to X_c, and find the convex hull by the linear-time Graham march (Graham, 1972). The generation of order statistics in order in linear time is well-known: see for example section V.3 of Devroye (1986). Still, our time complexity grows linearly with n. Contrast this with an obvious (distribution-dependent) lower bound for the expected time given by $\mathbf{E}|C_n|$. For the uniform distribution on any polygon with a finite number of vertices, $\mathbf{E}|C_n| = \Theta(\log n)$. For the uniform distribution on the unit sphere, it is $\Theta(n^{1/3})$ (Rényi and Sulanke, 1963, 1964). In the same papers, we find that for the standard normal distribution, $\mathbf{E}|C_n| \sim 2\sqrt{2\pi \log n}$. In other words, the expected size of a normal random cloud with $n = 10^3$ is about 13. For $n = 10^6$, 10^9, 10^{12} and 10^{15}, we obtain about 18, 22, 26 and 29 respectively. The implication is simple: if we wish to simulate a fair-sized normal random convex hull, we will require phenomenal values for n. Linear expected time algorithms just won't do. There are even distributions for which $\mathbf{E}|C_n|$ grows slower with n: Carnal (1970) first pointed out the existence of heavy-tailed radially symmetric distributions for which $\mathbf{E}|C_n|$ tends to a finite constant as $n \to \infty$. We will make our point here with the aid of the normal sample, though.

The method of Devroye (1982) can be applied to any radially symmetric distribution. The idea is to select a threshold radius r_n and to bet on the event that the circle S_{0,r_n} lies entirely in the convex hull. If that is indeed the case, one would only need to generate the points that fall outside the said circle. By choice of r_n, this number is small.

```
[set-up]
$r \leftarrow \sqrt{2\log n - 3\log\log n + 2c}$
  where $c \le -(1/2)\log(4\pi)$
```

```
[generation]
generate a binomial $(n, e^{-r^2/2})$
  random variate $N$
generate an ordered random sample $\Theta_1, \dots \Theta_N$
  with a uniform distribution on $[0, 2\pi]$
generate i.i.d. exponential random
  variates $E_1, \dots, E_N$
for the points with polar coordinates
  $(\sqrt{2(r^2 + E_1)}, \Theta_1), \dots, (\sqrt{2(r^2 + E_N)}, \Theta_N)$
  find the convex hull CH by a
  linear time Graham march
if $S_{0,r} \subseteq$ CH
  then return CH
  else determine the largest radius $R$
          such that $S_{0,R} \subseteq$ CH
        generate a binomial $(n - N, p)$
          random variate $M$ where
          $p = (e^{-R^2/2} - e^{-r^2/2})/(1 - e^{-r^2/2})$
        generate an angularly ordered
          random sample of $M$
          independent points drawn from
          the uniform distribution on
          $S_{0,r} - S_{0,R}$
        merge the samples into an angularly
          ordered list of size $N + M$
        find the convex hull CH
          by a Graham march
        return CH
```

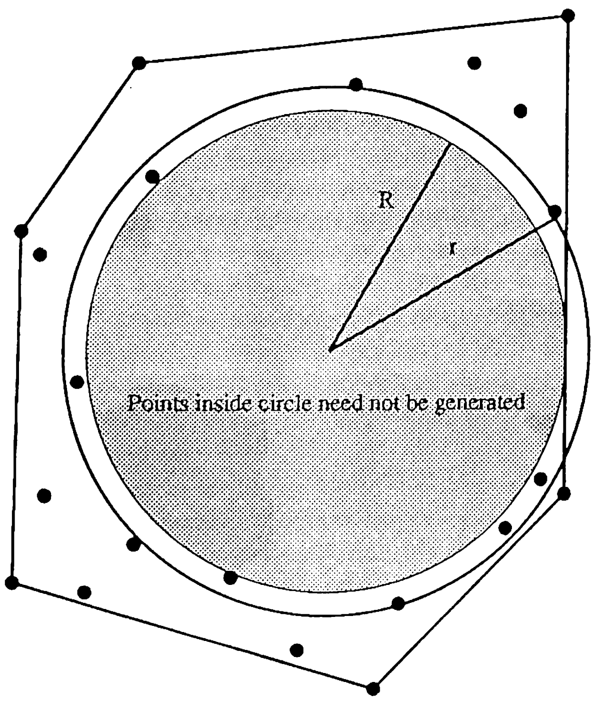

Figure 1: Random Convex Hull Generator

In the algorithm, we used the fact that $\sqrt{N_1^2 + N_2^2}$ has distribution function $1 - \exp(-r^2/2)$ on $(0, \infty)$, when N_1, N_2 are independent standard normal random variables. The ordered random sample of angles can be obtained in time $O(N)$ using the algorithms of section V.3 of Devroye (1986). Therefore, given N, the time taken by the first part of the algorithm is proportional to N. To determine R takes time proportional to N as well. Next, if the inclusion check fails, we need an additional sample of size M. In that case, there is an additional cost proportional to $N + M$, since two sorted lists can be merged in linear time. With a little effort we then obtain

Theorem 1. *The expected time taken by the above algorithm does not exceed a constant times* $(\log n)^{3/2}$ *if we take* $r = r_n = \sqrt{2 \log n - 3 \log \log n + 2c}$ *for a constant* c *satisfying* $c \leq -(1/2) \log(4\pi)$.

Proof. Binomial random variates can be generated in constant expected time, uniformly bounded over all parameters (Devroye, 1986, section X.4). For what follows, we do not even need such sophistication. Everything below remains true if N for example is generated by the inversion method so that the expected time is bounded by a constant times $1 + \mathbf{E}N$. Up to the generation of M, the expected time is thus bounded by a constant times

$$1 + \mathbf{E}N = 1 + ne^{-r^2/2} = 1 + e^{-c}(\log n)^{3/2} .$$

The contribution of the second part to the expected time is bounded from above in a rather crude manner by assuming that $R = 0$. The cost of the generation of the M-sized angularly ordered sample, its merge with the N-sized sample, and the Graham march does not exceed a constant times n. The expected complexity due to that part of the algorithm is not larger than a constant times nQ, where Q is the probability that $S_{0,r}$ is not contained in CH. This is bounded as follows: for $X_i \notin S_{0,r}$, let T_i be the collection of X_j, $j \neq i$, that are such that the segment linking X_j with the origin crosses one of the tangents of $S_{0,r}$ through X_i. Then,

$$\begin{aligned} Q &\leq \mathbf{P}\{N \leq 2\} + \mathbf{E}\left\{\sum_{i=1}^{n} I_{[X_i \notin S_{0,r}]} I_{[|T_i|=0]}\right\} \\ &\leq \left(1 - e^{-r^2/2}\right)^{n-1} (1 - e^{-r^2/2} + n) \\ &\quad + n\mathbf{P}\{X_1 \notin S_{0,r} \ , \ |T_1| = 0\} \\ &\leq (n+1) \exp\left(-(n-1)e^{-r^2/2}\right) \\ &\quad + 2ne^{-r^2/2}(1-p)^{n-1} \end{aligned}$$

where p is the probability that the first coordinate of X_1 exceeds r. With our choice of r, we note that the bound becomes

$$\begin{aligned} Q &\le (n+1)\exp\left(-(1-1/n)e^{-c}(\log n)^{3/2}\right) \\ &\quad +2e^{-c}(\log n)^{3/2}e^{-p(n-1)} \\ &= o(1/n)+2e^{-c}(\log n)^{3/2}e^{-p(n-1)} . \end{aligned}$$

Collecting bounds, we note that the overall expected time is not greater than a constant times

$$1+o(1)+e^{-c}(\log n)^{3/2}\left(1+2ne^{-p(n-1)}\right) .$$

It suffices thus to establish that $p \to 0$ and that $ne^{-pn} = O(1)$. But from the properties of the normal distribution, we recall that

$$\begin{aligned} p &= \int_r^\infty \frac{1}{\sqrt{2\pi}}e^{-t^2/2}\,dt \\ &\le \frac{1}{r\sqrt{2\pi}}e^{-r^2/2} \\ &= \frac{e^{-c}\log n}{n\sqrt{4\pi}} \end{aligned}$$

so that indeed $ne^{-pn} = O(1)$ provided that $e^{-c} \ge \sqrt{4\pi}$. □

The algorithm given above is thus acceptably efficient. For other radially symmetric distributions, one can argue similarly, but all of them require the computation of a threshold value for r. There is, fortunately, a different strategy for uniform distributions on convex sets. Take for example the uniform distribution in the unit circle. We generate the random convex hull incrementally, while keeping track at all times of the partial convex hull in angular order, and the probability (p) that a new point falls outside the partial convex hull. Since this is just an area, it is easy to calculate and update. After having generated $X_1, X_2, \ldots, X_T$ where T is the first index such that the convex hull of $X_1, \ldots, X_T$ contains the origin, we start the following process: we keep a counter for the index (initially T), and increase the counter by a geometric (p) random variate. If the counter exceeds n, we stop. Otherwise, a point is generated uniformly in $S_{0,1}$ but outside the present convex hull, and both the convex hull and p are updated. With a bucket structure for the angles, we can update all in $O(1)$ expected time. To facilitate the generation, it helps to partition the space into a fan of infinite triangles with top at the origin and with sides going through the points on the convex hull. Intersect this with $S_{0,1}$ and with the outside of the convex hull. Each piece thus obtained has an easy-to-compute area. We could coin this the waiting time algorithm, in analogy with the paradigm explained in Devroye (1986, p. 71). The overall expected time is seen to be bounded by

$$O\left(\sum_{i=1}^n \frac{\mathbf{E}N_i}{i}\right) ,$$

where N_i is the size of the convex hull of $X_1, \ldots, X_i$. For the uniform distribution on the unit circle, we obtain an unimprovable expected complexity of $O(n^{1/3})$, while for the uniform distribition on the unit square (where $\mathbf{E}N_n \sim (8/3)\log n$), the overall expected complexity becomes $O(\log^2 n)$.

4 GENERATING SUMS

The idea of simulating random variables via shortcuts that bypass the definition of the random variables has been exploited by many. For example, a binomial (n, p) random variable representing the outcome of n coin flips is nowadays routinely generated in $O(1)$ expected time uniformly bounded over n and p (Kachitvichyanukul and Schmeiser, 1988; Stadlober, 1988, 1989). Assume more generally that we wish to generate many independent copies of the random variable $S_n = \sum_{i=1}^n X_i$, where the X_i's are i.i.d. random variables having a given density f. Obviously, we could always do this in time $O(n)$ by generating and summing the X_i's. For very large n, this is rather inefficient. At the same time, it is unacceptable to generate a random variate with a properly normalized limit law for S_n. However, local central limit theorems can be helpful in the design of an exact generator. Section XIV.4 of Devroye (1986) deals with precisely this problem. Basically, one could in general develop generators for distributions with known characteristic function (see also Devroye, 1986), and apply these to the situation at hand. This is promising if we know the characteristic function φ of f, since S_n must then have characteristic function φ^n. However, the resulting algorithms are rather cumbersome.

The strategy explored in Devroye (1988) rests on the simple principle of solving a complex problem by solving many easier problems, e.g., by first developing a generator for the sum S_n of n i.i.d. uniform $[-1, 1]$ random variables. This generator takes $O(1)$ expected time per random variate. From this, the user can build at will. as densities can be written as mixtures

$$f(x) = \sum_{i=1}^\infty p_i f_i(x) ,$$

where the f_i's are uniform densities on intervals $[a_i, b_i]$. The sum S_n of n i.i.d. random variables with density f can be generated as follows.

```
generate a multinomial $(n, p_1, p_2, \ldots)$
  random sequence $N_1, N_2, \ldots$
  (note that the $N_i$'s sum to $n$)
  (let $K$ be the index of the largest nonzero $N_i$)
$X \leftarrow 0$
for $i := 1$ to $K$ do
  generate $S$, the sum of $N_i$ i.i.d.
  random variables with common density $f_i$.
  $X \leftarrow X + S$
return $X$
```

Developing a uniformly fast generator for the sum of uniform $[-1, 1]$ random variables is not a sinecure as the density f_n of S_n can only be computed at $\Omega(n)$ time cost. This is best seen by noting first that S_n has characteristic function $(\sin(t)/t)^n$. For all $n \geq 2$, the density f_n can be obtained by the inversion formula

$$f_n(x) = \frac{1}{2\pi} \int \left(\frac{\sin(t)}{t}\right)^n \cos(tx)\, dt .$$

This yields

$$f_n(x) = n2^{-n} \sum_{k=0}^{\lfloor (n-|x|)/2 \rfloor} \frac{(-1)^k}{k!\,(n-k)!} (n - 2k - |x|)^{n-1} \quad (0 \leq |x| < n) .$$

The evaluation of the density f_n of S_n takes time proportional to n. Let us formalize this, and assume that the algorithm has a random integer cost associated with it, consisting of the number of uniform random variates needed before the algorithm halts, and of n times the number of evaluations of f_n (thus, reflecting the fact that each evaluation takes time proportional to n). These two components will be called R and N respectively. The algorithm we are after has the following desirable properties: uniformly over all n, $\mathbf{E}R \leq c < \infty$ for some constant c; and as $n \to \infty$, $\mathbf{E}N \to 0$. This means that the contribution from the evaluation of f_n is asymptotically negligible. In other words, one could be rather sloppy in the implementation of these evaluations, and barely notice any impact on the expected time per random variate. Furthermore, the overall expected time is uniformly bounded over n. To be able to avoid the evaluations of f_n nearly all the time means that we must in fact derive a relatively accurate expression for the actual density of S_n. For the solution, we use the truncated Gram-Charlier series:

$$g_n(x) = \frac{e^{-x^2/2}}{\sqrt{2\pi}} \left(1 + \frac{6x^2 - 3 - x^4}{20\, n}\right) ,$$

which approximates the density $f_n(x)$ of the normalized sum $\sqrt{3/n}S_n$. This normalization will be assumed throughout the remainder of this note. We need to know how good the approximation is. Devroye (1988) has shown the following:

$$\sup_x |f_n(x) - g_n(x)| \leq \frac{A}{n^2} ,$$

where $A = 3.9608280445\ldots$. This is used to derive a dominating curve in the rejection method. Indeed, we know that $\sqrt{3/n}S_n$ has support on $[-\sqrt{3n}, \sqrt{3n}]$. Thus, we have on this support set, maximizing the quartic polynomial in the definition of g_n with respect to x,

$$f_n(x) \leq \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \left(1 + \frac{6}{20\, n}\right) + \frac{A}{n^2} .$$

The dominating curve has integral not exceeding

$$1 + \frac{6}{20\, n} + \frac{2A\sqrt{3}}{n^{3/2}} = 1 + O\left(\frac{1}{n}\right) .$$

Next, we introduce a squeeze step, both for rejection and acceptance. An evaluation of f_n is only necessary when both squeeze tests fail. It is known that the expected number of evaluations of f_n is the the area between the squeeze curves (Devroye, 1986, p. 54). In our case, this yields a value not exceeding $4A\sqrt{3}/n^{3/2}$. Hence, $\mathbf{E}N$, which equals n times this value, is $O(1/\sqrt{n})$. We have thus achieved our goals stated above. We summarize the algorithm:

```
[set-up]
$A \leftarrow 3.9608280445\ldots$
compute $p \leftarrow 1 + \frac{6}{20\,n}$ and $q \leftarrow \frac{2A\sqrt{3}}{n^{3/2}}$
[generator]
repeat
  generate a uniform $[0, 1]$ random variate $U$
  if $U \leq \frac{q}{p+q}$
    then generate a uniform $[-n, n]$
           random variate $X$
         (or set $X \leftarrow -n + 2n(p+q)U/q$)
    else generate a normal random variate $X$
  generate a uniform $[0, 1]$ random variate $V$.
  $T \leftarrow V\left(\left(1 + \frac{6}{20\,n}\right)\frac{e^{-X^2/2}}{\sqrt{2\pi}} + An^{-2}\right)$
  if $|X| > \sqrt{3n}$ then Accept $\leftarrow$ False
  else if $T \leq g_n(X) - An^{-2}$ then Accept $\leftarrow$ True
  else if $T \geq g_n(X) + An^{-2}$ then Accept $\leftarrow$ False
  else Accept $\leftarrow [T < f_n(X)]$
until Accept
return $X$ (note: $X$ is $S_n\sqrt{3/n}$).
```

The properties of the algorithm are well understood: the expected number of iterations is $1 + (6/(20\,n)) + 2A\sqrt{3}n^{-3/2}$, the expected number of evaluations of f_n is not greater than $4A\sqrt{3}/n^{3/2}$, and thus $\mathbf{E}N \leq 4A\sqrt{3/n}$.

5 RANDOM BINARY SEARCH TREES

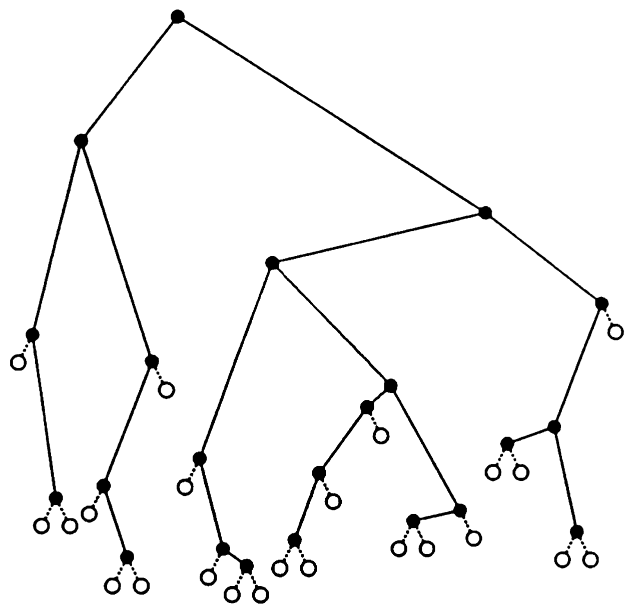

Figure 2: Random Binary Search Tree
(Internal nodes are black, external nodes are white)

The binary search tree is one of the most frequently used structures in computer science, see e.g. Aho, Hopcroft and Ullman (1983), Knuth (1973) or Cormen, Leiserson and Rivest (1990). A random binary search tree is defined as the random binary tree obtained by consecutive insertion of $X_1, \ldots, X_n$ into an initially empty tree, where $X_1, \ldots, X_n$ is either an i.i.d. sequence of random variables with a fixed density, or an equiprobable random permutation of $\{1, \ldots, n\}$. The height H_n of the tree is the maximal distance between any node and the root (thus, $H_1 = 0$, as the root is at distance 0 from itself, and $H_2 = 1$). Early studies of H_n include Robson (1979, 1982), Pittel (1984) and Mahmoud and Pittel (1984). See also Mahmoud (1992). While it is known that

$$\frac{H_n}{\log n} \to c \overset{\text{def}}{=} 4.33107\ldots \text{a.s.}$$

as $n \to \infty$ (Devroye, 1986, 1987), very little additional information is available regarding H_n, and one is led to simulation in order to study the second order properties of H_n such as its variance, its asymptotic distribution, and so forth. Such simulations require formidable values of n in view of the logarithmic growth of H_n with n. It is thus of great importance to assure that the time and especially the space requirements grow slowly with n. For one thing, to simulate H_n, it is simply out of the question that a random tree is generated.

One solution to this problem is to generate a random vector $(n_0, \ldots, n_m)$ where n_i is the number of external nodes at level i. Clearly, for a random binary search tree with n nodes, we have $n+1$ external nodes, and thus $\sum_i n_i = n + 1$. We obtain H_n as $m - 1$. Various paradigms are studied in Devroye and Robson (1992), including depth-first search with pruning, incremental methods in which the tree grows with random-sized jumps, and a tree growing procedure gleaned from birth-and-death processes. The last method takes $O(\log^4 n)$ expected time.

5.1 A Simple Linear Time Algorithm

Constructing the tree by consecutive insertions leads to a $\Theta(n \log n)$ expected time and $\Theta(n)$ space algorithm. In Devroye (1986, p.650), a linear time method is given for generating a random binary search tree. The basic algorithm is shown below.

```
$m \leftarrow 0$ ($m$ is the number of levels)
$n_0 \leftarrow 1$
for $N := 1$ to $n$ do
  ($N$ is the number of external nodes)
  generate $L$ randomly in $\{0, \ldots, m\}$
    according to the frequencies $n_0, \ldots, n_m$
  $n_L \leftarrow n_L - 1$
  if $L = m$ then $m \leftarrow m + 1$
  $n_{L+1} \leftarrow n_{L+1} + 2$
return $(n_0, n_1, \ldots, n_m)$
(the height of the tree is $m - 1$)
```

In this algorithm, we keep the number of external nodes at each level in an array $(n_0, n_1, \ldots, n_m)$. The expected storage needed for this is $O(\log n)$ since the expected height is $O(\log n)$. The algorithm is based upon the fact that when adding a new node, each of the external nodes is equally likely to receive the node. To generate the random integer L according to the vector of frequencies $(n_0, \ldots, n_m)$, one can trivially proceed in time $O(1 + m)$, but this would result in overall expected time $O(n \log n)$. To generate L in $O(1)$ expected time, one has to either use more space or more programming resources. For example, keeping an array with $n_0 + n_1 + \cdots n_m$ entries, of which n_i entries have label i, would enable us to generate L in $O(1)$ time. The overall time and space both are $O(n)$. By a dynamic version of the guide table method, $O(\log n)$ expected space is achievable provided that we can update the guide table in $O(1)$ amortized time. This is easy to achieve if we take care to rebuild the table every $\log n$-th (or m-th) entry. Between rebuilding, the new entries in the table are collected in a simple overflow list; this does not affect the overall linear expected time.

5.2 Discrete Jumps in the Simulation

Probabilistic shortcuts based upon waiting times allow us to reduce the expected time to $O(\sqrt{n}\log n)$. This requires efficient generators for a multivariate hypergeometric and a certain waiting time distribution, thereby rendering the programming effort and the overhead a bit heavier. Nevertheless, for medium-sized values of n, the method is very useful.

Consider a vector $(n_0, n_1, \ldots, n_m)$ representing the number of external nodes at levels $0, 1, \ldots, m$ respectively, and assume that $n_0 + \ldots + n_m = n$ for now. The previous linear time algorithm is based upon the selection of a uniform random external node, say one at level $k \in \{0, \ldots, m\}$, and the updating of the vector by the rule

$$(n_k, n_{k+1}) \leftarrow (n_k - 1, n_{k+1} + 2) .$$

Imagine that the n original external nodes are white balls in an urn, and that the label of each ball is its level number. A randomly selected (white) ball is removed. If its label is k, two black balls with label $k+1$ are added. This process can be repeated until we pick for the first time a black ball. The number of draws required until this happens is a random variable $T = T_n \in \{2, \ldots, n+1\}$. We say that T_n has the *waiting-time distribution* with parameter n. We can let the tree growing process jump ahead by T steps at once if we are given T. Indeed, given T, it suffices to draw $T-1$ white balls uniformly and without replacement from the urn. The vector $(D_0, D_1, \ldots, D_m)$ represents the number of balls drawn with labels $0, 1, \ldots, m$ respectively. The distribution of this vector is multivariate hypergeometric, and it can be generated in $O(m)$ expected time uniformly over all n. The vector of external nodes is updated by the rule:

$$\begin{aligned}(n_0, \ldots, n_m, n_{m+1}) \quad \leftarrow \quad & (n_0, \ldots, n_m, 0) \\ - \quad & (D_0, D_1, \ldots, D_m, 0) \\ + \quad & 2(0, D_0, \ldots, D_m) .\end{aligned}$$

The single black ball is taken care of by selecting a label L at random from the $T-1$ white ball labels just selected, the k-th label being picked with probability proportional to D_k. This label can be chosen in time $O(m)$. A further update is required:

$$(n_{L+1}, n_{L+2}) \leftarrow (n_{L+1} - 1, n_{L+2} + 2) ,$$

where, if $L = m+1$, we define $n_{m+2} = 0$ before the update and $n_{m+2} = 2$ after the change. The number of external nodes now is $n + T$ instead of n. Iterate this process until we obtain more external nodes than needed. It is a simple matter to get the exact size one wants by simply truncating T in the last iteration. The following remaining details are ironed out in Devroye and Robson (1992):

- Determine the waiting time distribution for T and show how a random variate T can be generated in constant expected time, bounded uniformly over n.
- Show how one can generate a multivariate hypergeometric random variate.
- Show that the algorithm takes $O(\sqrt{n}\log n)$ expected time units.

5.3 A Birth-and-Death Process Method

A random binary search tree can also be grown by imagining that each external node is a living organism that will bear two children and die according to a simple Poisson point process. We then let the time grow by constant amounts, so that the tree grows at an exponential rate. At any given moment, we have a correctly distributed binary search tree, but the size is random. When one stops as soon as the size of the tree is n or larger, the expected time complexity is $O(\log^4 n)$. A modification of the algorithm is introduced to obtain the right size. The constant in $O(\log^4 n)$ is rather large due to the overhead in a multinomial random vector generator used in a bottleneck portion of the algorithm. This last method is useful for extremely large n, such as $n \approx 10^{40}$. In Robson (1979, 1982), simulations were reported that were based upon an ultra-fast algorithm that produces random binary search trees of random size. Once again, consider a vector $(n_0, n_1, \ldots, n_m)$ representing the number of external nodes at levels $0, 1, \ldots, m$ respectively, and assume that $n_0 + \ldots + n_m = n$ for now. Every external node should be considered as a living element in a birth-and-death process with unit reproduction rate for each element. When an external node gives birth, it produces two new elements (which live at one level below their parent), and it immediately dies, for a net gain of one element. The n nodes at time t will thus spawn families of offspring at time $t + \Delta$ of i.i.d. sizes $S_1, \ldots, S_n$. The key to the solution is to find the distribution of these S_i's. We claim that the common distribution is that of S, where

$$\mathbf{P}\{S = k\} = \left(1 - e^{-\Delta}\right)^{k-1} e^{-\Delta} \ (k \geq 1) .$$

If the state at time t is described by $(n_0, n_1, \ldots, n_m)$, then our purpose is to efficiently generate an updated state at time $t + \Delta$, where Δ is a constant to be selected. The first step is to generate the sizes of the subtrees of external nodes (at time $t + \Delta$) with roots at the elements alive at time t. This leads to the generation of the triangular array of random integers $N(i, j)$, each representing the number of size j subtrees with original root at level i. Thus,

$$\sum_{j=1}^{\infty} N(i, j) = n_i \ , \ 0 \leq i \leq m \ .$$

In fact, $(N(i, 1), N(i, 2), N(i, 3), \ldots)$ is multinomial with parameter n_i and probabilities given by $p_1(\Delta), p_2(\Delta), \ldots$. By repeatedly appealing to a uniformly fast binomial generator, we can generate this vector in expected time bounded by a constant times the expected value of the maximal size subtree $M(n_i)$ for any of the n_i roots. Now, the maximum of n_i i.i.d. geometric random variables described by the probabilities $p_i(\Delta)$, $i \geq 1$, has expected value not exceeding

$$1 + \frac{1 + \log n_i}{\log(1/(1 - e^{-\Delta}))} \ .$$

Since each n_i does not exceed n, we see that, given m, all $N(i, j)$ can be generated in expected time $O(m \log n)$.

The next step in the algorithm consists of generating the correct numbers of external nodes at all levels. This can be done in one sweep from 0 to m. Assume that we are given $N(i, j)$, $j \geq 1$, for fixed i. This leads to $N(i, 1)$ external nodes at level i. For fixed $j \geq 2$, we obtain no external nodes at level i. Rather, it is possible to determine how many subtrees rooted at nodes of level $i + 1$ this leads to. Indeed, a node at level i with a subtree having j external nodes yields a left child at level $i + 1$ which itself has a subtree of (random) size $S \geq 2$, where S is equally likely to take any value between 1 and $j - 1$. The size of the subtree rooted at the right child is $j - S$. Of course, we won't have to do this for each node separately. Rather, it is easy to see that we need only generate a multinomial random vector $w_1, w_2, \ldots, w_{j-1}$ with $\sum w_l = N(i, j)$, and equal probabilities. Here w_l represents the number of left child nodes at level $i + 1$ having l external nodes in their subtrees. Adding in the sizes of the right subtrees, we see that at level $i + 1$, the $N(i, j)$ level i nodes spawn $2N(i, j)$ nodes with $w_l + w_{j-l}$ nodes of size l. A uniformly fast binomial generator can "split" all $N(i, j)$ at level i in this manner in expected time $O(\mathbf{E}M(n_i)^2)$. For fixed Δ, this is $O(\log^2 n)$.

The sizes of the nodes at level $i + 1$ can now be updated, and they in turn are split. An iteration thus takes expected time not exceeding $\mathbf{E}H_N$ times $O(\log^2 n)$ where N is the number of external nodes at the end of the iteration. Devroye and Robson (1992) showed that the above algorithm halts in expected time $O(\log^4 n)$.

REFERENCES

Aho, A. V., J. E. Hopcroft and J. D. Ullman. 1983. Data Structures and Algorithms. Addison-Wesley: Reading, Mass.

Bondesson, L. 1982. On simulation from infinitely divisible distributions. *Advances in Applied Probability* 14: 855–869.

Carnal, H. 1970. Die konvexe Hülle von n rotationssymmetrisch verteilten Punkten. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* 15: 168–176.

Chambers, J. M., C. L. Mallows and B. W. Stuck. 1976. A method for simulating stable random variables. *Journal of the American Statistical Association* 71: 340–344.

Chen, H. C. and Y. Asau. 1974. On generating random variates from an empirical distribution. *AIIE Transactions* 6: 163–166.

Cormen, T. H., C. E. Leiserson and R. L. Rivest. 1990. Introduction to Algorithms. MIT Press: Boston.

Devroye, L. 1982. On the computer generation of random convex hulls *Computers and Mathematics with Applications* 8: 1–13.

Devroye, L. 1984. Methods for generating random variates with Polya characteristic functions. *Statistics and Probability Letters* 2: 257–261.

Devroye, L. 1986. *Non-Uniform Random Variate Generation*. Springer-Verlag. New York.

Devroye, L. 1986. A note on the height of binary search trees. *Journal of the ACM* 33: 489–498.

Devroye, L. 1987. Branching processes in the analysis of the heights of trees. *Acta Informatica* 24: 277–298.

Devroye, L. 1988. An automatic method for generating random variables with a given characteristic function. *SIAM Journal of Applied Mathematics* 46: 698–719.

Devroye, L. 1988. Generating sums in constant average time. In *Proceedings of the 1988 Winter Simulation Conference*, ed. M. A. Abrams, P. L. Haigh and J. C. Comfort, 425–431. IEEE, San Diego, CA.

Devroye, L. and M. Robson. 1992. On the generation of random binary search trees. Technical Report, School of Computer Science, McGill University, Montreal.

Durrett, R. 1991. Probability: Theory and Examples. Wadsworth and Brooks: Pacific Grove, CA.

Fishman, G. S. and L. R. Moore. 1984. Sampling from a discrete distribution while preserving monotonicity. *The American Statistician* 38: 219–223.

Fishman, G. S. and L. S. Yarberry. 1990. Generating a sample from a k-cell table with changing probabilities in $O(\log_2 k)$ time. Technical Report UNC/OR/TR-90/10, Department of Operations Research, University of North Carolina, Chapel Hill, NC.

Fox, B. L. 1990. Generating Markov-chain transitions quickly: I. *ORSA Journal on Computing* 2: 126–135.

Fox, B. L. 1991. Generating Markov-chain transitions quickly: II. *ORSA Journal on Computing* 2: 3–11.

Graham, R. 1972. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters* 1: 132–133.

Kachitvichyanukul, V. and B. W. Schmeiser. 1988. Binomial random variate generation. *Communications of the ACM* 31: 216–222.

Knuth, D. E. 1973. The Art of Computer Programming, Vol. 3 : Sorting and Searching. Addison-Wesley: Reading, Mass.

Kronmal, R. A. and A. V. Peterson. 1979. Programs for generating discrete random integers using Walker's alias method. Department of Biostatistics, University of Washington.

Kronmal, R. A. and A. V. Peterson. 1979. On the alias method for generating random variables from a discrete distribution. *The American Statistician* 33: 214–218.

Mahmoud, H. and B. Pittel. 1984. On the most probable shape of a search tree grown from a random permutation. *SIAM Journal on Algebraic and Discrete Methods* 5: 69–81.

Mahmoud, H. M. 1992. Evolution of Random Search Trees. John Wiley: New York.

Neuts, M. F. and M. E. Pagano. 1981. Generating random variates from a distribution of phase type. Technical Report 73B, Applied Mathematics Institute, University of Delaware, Newark, Delaware.

Peterson, A. V. and R. A. Kronmal. 1983. Analytic comparison of three general-purpose methods for the computer generation of discrete random variables. *Applied Statistics* 32: 276–286.

Pittel, B. 1984. On growing random binary trees. *Journal of Mathematical Analysis and Applications* 103: 461–480.

Popescu, I. and I. Vaduva. 1991. A survey of computer generation of some classes of stochastic processes. *Mathematics and Computers in Simulation* 33: 223–241.

Preparata, F. P. and M. I. Shamos. 1985. *Computational Geometry. An Introduction.* New York: Springer-Verlag.

Rényi, A. and R. Sulanke. 1963. Über die konvexe Hülle von n zufällig gewählten Punkten. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* 2: 75–84.

Rényi, A. and R. Sulanke. 1964. Über die konvexe Hülle von n zufällig gewählten Punkten, *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* 3: 138–1247.

Robson, J. M. 1979. The height of binary search trees. *The Australian Computer Journal* 11: 151–153.

Robson, J. M. 1982. The asymptotic behaviour of the height of binary search trees. *Australian Computer Science Communications* 88.

Seneta, E. 1981. *Non-Negative Matrices and Markov Chains.* New York: Springer-Verlag.

Stadlober, E. 1989. Ratio of uniforms as a convenient method for sampling from classical discrete distributions. In *Proceedings of the 1989 Winter Simulation Conference*, ed. E. A. MacNair, K .J. Musselman and P. Heidelberger, 484–489. IEEE.

Stadlober, E. 1989. Binomial random variate generation: a method based on ratio of uniforms. *American Journal of Mathematical and Management Sciences* 9: 1–20.

Walker, A. J. 1977. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters* 10: 127–128.

Walker, A. J. 1977. An efficient method for generating discrete random variables with general distribution. *ACM Transactions on Mathematical Software* 3: 253–256.

AUTHOR BIOGRAPHY

LUC DEVROYE is a Professor in the School of Computer Science of McGill University in Montreal. His research interests include random variate generation, probabilistic analysis of algorithms, random search, data structures, and nonparametric estimation.