How to sample from a probability distribution

Warren D. Smith*
wds@research.NJ.NEC.COM

April 18, 2002

Abstract —
This brief note explains how to sample a probability N-vector in $O(1)$ time per sample, with the aid of a $O(N)$-word data structure buildable in $O(N)$ steps. The algorithm is highly practical; we give pseudocode based on actual working code.

Keywords — Sampling. Linear time algorithm. Data structure.

A fundamental programming task is sampling from an explicitly given probability distribution on N events, which we shall imagine is represented by an N-vector[1] $\vec{X}$. Surprisingly, the best method for doing managed to escape previous notice. What makes this even more surprising, is that the key idea was invented in 1977 by A.Walker [2], and then was presented in Knuth's textbook [1][2], but both Walker and Knuth (although Knuth improved upon Walker's original conception) failed to notice the optimal result, our theorem 1.

function WalkerSample(real $\vec{Y}$, positive integer N, $\vec{A}$)
positive integer i; real r;
$i \leftarrow 1 + \lfloor N \cdot \mathrm{rand}_{[0,1)} \rfloor$; $r \leftarrow \mathrm{rand}_{[0,1)}$;
if $r > Y_i$ **then**
  $i \leftarrow A_i$;
end if
return i.

Figure 1. Walker's sampling algorithm.

Theorem 1. (Complexity of sampling) *Let $\vec{X}$ be a probability N-vector. Then on a RAM with a generator of random variates uniform in $[0,1)$ there is an algorithm to produce samples $i \in \{1,...,N\}$ with probability X_i, in $O(1)$ steps (worst-case) per sampling. The sampling algorithm relies on a data structure consisting of one integer array $\vec{A}$ and one real array $\vec{Y}$, each having N elements. This data structure may be built (starting from $\vec{X}$) in $O(N)$ steps with the temporary use of one additional N-element integer array for bookkeeping.*

Proof. The $O(1)$-time sampling algorithm `WalkerSample` is due to A.Walker [2]. Its intuitive idea is: we sample from a uniform distribution (all $X_i = N^{-1}$), which may be thought of as a histogram with N bars all of equal height. We then wish to correct this to get the actually-desired distribution $\vec{X}$. That is accomplished by dividing the ith histogram bar into two pieces, one of which is labeled "stay here at i" and the other of which is labeled "go to $j = A_i$ instead." By selecting the heights of the two sub-bars (and the destinations j) correctly, we may always get $\vec{X}$ exactly correct after only 1 such correction step.

procedure BuildSampler(real $\vec{X}$, natural N, $\vec{A}$, $\vec{B}$)
assert $X_i \geq 0$ for all $i = 1, \ldots, N$, and $N \geq 1$;
natural j, k;
for $j = 1, 2, \ldots, N$ **do**
  $A_j \leftarrow j$; $B_j \leftarrow j$; ▷ initial destinations (stay there)
  $X_j \leftarrow N \cdot X_j$; ▷ initial (scaled) bar heights
end for
$B_0 \leftarrow 0$; $X_0 \leftarrow 0.0$; ▷ sentinels
$B_{N+1} \leftarrow N+1$; $X_{N+1} \leftarrow 2.0$;
$k \leftarrow 0$; $j \leftarrow N+1$;
loop
  repeat ▷ find k so B_k's budget too small
    $k \leftarrow k+1$;
  until $X[B_k] \geq 1.0$
  repeat ▷ find j so B_j's budget too large
    $j \leftarrow j-1$;
  until $X[B_j] < 1.0$
  exitwhen $k \geq j$;
  swap $B_k \leftrightarrow B_j$;
end loop
assert $\vec{B}$ is now (and will remain) ordered so overfunded histogram bars first, underfunded ones last;
$k \leftarrow j+1$;
while $j > 0$ **do** ▷ B_j initially overfunded
  while $X[B_k] \leq 1.0$ **do** ▷ get next underfunded bar B_k
    $k \leftarrow k+1$;
  end while
  exitwhen $k > N$; ▷ done with all adjustments
  $X[B_k] \leftarrow X[B_k] + X[B_j] - 1.0$; ▷ adjust bar heights
  $A[B_j] \leftarrow B_k$; ▷ and destinations
  if $X[B_k] < 1.0$ **then**
    swap $B_j \leftrightarrow B_k$; $k \leftarrow k+1$;
  else
    $j \leftarrow j-1$;
  end if
end while.

Figure 2. $O(N)$-time algorithm to build Walker's data structure (based on actual working C code). Probability N-vector $\vec{X}$ is overwritten by $\vec{Y}$ on output and $\vec{A}$ is created. X_0, X_{N+1} and $B_{0,\ldots,N+1}$ are used for temporary storage.

This existence claim is made clear by the $O(N)$-step construction algorithm `BuildSampler`. (Walker's original build-algorithm had required order N^2 steps.) Its operation is largely explained by the comments in the code. The idea is that there are two subsets of histogram bars: those which are below average and those which are above average. Unless the distribution is exactly uniform (in which case sampling is a triviality) both of these subsets have cardinality≥ 1. We find two histogram bars, one from each set, and make the bar whose desired probability is below average donate an appropriate amount of its probability budget (initially N^{-1}) to the other, so that its probability budget now agrees with its

*NECI, 4 Independence Way Princeton NJ 08540 USA

[1] Here we assume every coordinate of the vector is nonzero, or equivalently redefine N so that it is the number of nonzero coordinates in the vector.

[2] The *second* edition of Knuth [1], section 3.4.1 (pages 120-121) and exercise 7.

desired probability; this bar may now be eliminated from all further consideration. The bar that accepted the donation now has its budget readjusted (and may switch sets). We continue this process until all bars are eliminated. The 2 sets (and the third set, of eliminated bars) are conveniently kept in parts of a partitioned array.

Because it is simpler – and because ultimately this is needed anyway to make the output routine interface with a random number generator on $[0, 1)$ – we work with all bar heights scaled up by a factor of N. It is easiest to think about the algorithm as manipulating *two* real N-vectors, one for the desired probability distribution $\vec{X}$, and the other for the current "budget," or approximation. But if such an algorithm is written down, you will soon see that it is possible to simplify it to make it have only *one* real N-vector, which on input is the desired probability distribution $\vec{X}$ and which is converted in place to the output $\vec{Y}$. The fact that, each such step, the counter k increments or j decrements (and once $k > N$ or $j \leq 0$ the procedure terminates), makes the proof that the runtime is $O(N)$ trivial. □

Working C code for all this is available at `http://www.neci.nj.nec.com/homepages/wds/WDSsampler.c`.

References

[1] D.E.Knuth: Seminumerical algorithms, *second* edition, Addison-Wesley-Longman 1999.

[2] Alastair J. Walker: An Efficient Method for Generating Discrete Random Variables with General Distributions, ACM Trans. Mathematical Software 3,3 (Sept 1977) 253-256.