Proceedings of the 1984
Winter Simulation Conference
S. Sheppard, U. Pooch, D. Pegden (eds.)

USE OF LATTICE STRUCTURES FOR REDUCTION OF SIMULATION RUN TIME

Norman S. Matloff
Department of Electrical and Computer Engineering
University of California at Davis

ABSTRACT

The use of quasi-random numbers has been shown to produce significant reductions in simulation run time, for a given target level of accuracy. Such reductions come from the uniformity of the sampling process generated by the quasi-random numbers. In this paper, a method is proposed under which quasi-random numbers can be used to achieve even greater reductions in run time. The method applies to a large class of simulation operations, specifically those in which some time-consuming but updatable operation is performed.

1. SIMULATION THROUGH QUASI-RANDOM NUMBERS

There is a substantial literature concerned with reduction of run time in simulation software. Most of the work has taken a probabilistic approach, aimed at reducing the output variance for a given amount of data. Examples of this are antithetic and control variates [7].

Another approach, less common, has been that of quasi-random numbers (e.g. [3], [5], [1] and [2]). Here the theme is qualitatively different from that of variance reduction. The motivation here is not probabilistic, even though the goals are still to estimate probabilities and expectations. Instead, there is a definite deterministic flavor to the method, more in the direction of numerical integration. The idea is to actively choose data points to represent the distributions being sampled, rather than simulate sequences of random variables directly. Good choices of points will cover the data space more uniformly than points generated in the usual random fashion, with the result being that the distributions will be approximated more accurately.

The concept will be made much easier to discuss by referring to the following very simple example: Suppose X and Y are independent random variables having exponential distributions with means 10.0 and 20.0, respectively, and that we wish to find $P(|X-Y| < 5.0)$. The usual approach would be to repeatedly (say, N times) generate pairs (U_1,U_2) of independent U(0,1) random variables, and then form (X,Y) as $(-0.10\ln(U_1), -0.05\ln(U_2))$. Let U_{1i}, U_{2i}, X_i and Y_i $(i=1,\ldots,N)$ denote the N sets of random numbers generated. The desired probability would then be estimated as the proportion of subscripts i such that $|X_i-Y_i| < 5.0$, among $i=1,\ldots,N$.

The method of quasi-random numbers takes a substantially different approach to this problem. To describe this approach, first note that in the classical simulation procedure described above, what is really occurring at the foundation is that we are using the N points $(U_{11},U_{21}),\ldots,(U_{1N},U_{2N})$ to approximate the uniform distribution on the unit square $[0,1]^2$. The idea of quasi-random numbers is then to improve on this approximation by choosing the pairs (U_{1i},U_{2i}) in such a way as to better approximate the uniform measure on $[0,1]^2$. In the form to be used here, the pairs can be taken to be the nodes on the lattice

$$\ell(M,2) = \{\tfrac{0}{M},\tfrac{1}{M},\ldots,\tfrac{M-1}{M}\}^2 = \{(\tfrac{i}{M},\tfrac{j}{M}):i,j=0,1,\ldots,M-1\}, \quad (1)$$

where $M=N^{1/2}$ and N is taken to be a perfect square (it is better to use $\frac{k+0.5}{M}$ instead of $\frac{k}{M}$, but the latter will be used to simplify notation).

The fact that the points in $\ell(M,2)$ are more uniformly spread out over the unit square implies that the resulting estimator of $P(|X-Y| < 5.0)$ is substantially more accurate than the estimator obtained by the usual probabilistic approach to simulation. This is discussed in detail in the references on quasi-random numbers cited above. However, in many applications the special lattice structure such as that in $\ell(M,2)$ can be exploited to yield even better accuracy. It is this point which is the subject of the methodology introduced here.

The approach proposed here is concerned with those applications in which one of the operations being simulated is a time-consuming but "updatable" procedure. For example, sorting is an updatable operation: Suppose we have a sorted list of d numbers, and that one of these is replaced by a new value. Then the new sorted list can be obtained from the old one in $O(\log_2 d)$ operations, using a binary search to find the position at which the new value should be inserted; this is in contrast to the $O(d \log_2 d)$ operations needed if the whole set of numbers were to be sorted "from scratch".

The purpose of this paper is to develop techniques with which this updatability quality can be exploited in the quasi-random number setting.

2. THE PROPOSED METHODOLOGY

We first need some notation. Define the lattice $\ell(M,d)$ to be the d-dimensional analog of $\ell(M,2)$. Formally, the lattice is defined as the d-fold Cartesian product

$$\{\tfrac{0}{M},\tfrac{1}{M},\ldots,\tfrac{M-1}{M}\}^d, \quad (2)$$

i.e. the set of all d-tuples whose entries are of the form $\frac{k}{M}$ $(k=0,1,\ldots,M-1)$. Note that nodes in the lattice may be considered as d-digit, radix-M numbers; this interpretation then defines an ordering on $\ell(M,d)$: If A and B are nodes on the lattice, then $A < B$ means that A is smaller than B when considered as a radix-M number.

Suppose that the simulation will consist of N replications of some setting in which there are d independent (but not necessarily identically distributed) random variables, $X_1,X_2,\dots,X_d$. It will be assumed that each random variable X_j can be generated using a single U(0,1) variate U_j. Note that although there are some distributions which are typically generated by more than one U(0,1) variate, such distributions can still be obtained using one such variate. For example, let Φ denote the cumulative distribution function for the standard Gaussian distribution N(0,1), which usually needs several U(0,1) variates for generation [4]. $\Phi^{-1}(t)$ can be found numerically for, say, t equal to i/10000 (i=1,...,10000), and stored on disk for use both in the current and in future simulations. Then the standard Gaussian random variable Z can be generated as $\Phi^{-1}(U')$, where U is a U(0,1) variate and U' is equal to the value of U rounded up to the nearest point of the form i/10000. Other Gaussian distributions can then easily be obtained from Z.

Thus each generation of the random variables $X_1,\dots,X_d$ will come from a corresponding set $U_1,\dots,U_d$. Let $U_{1i},U_{2i},\dots,U_{di}$ denote the uniform variates used in the i-th generation, and let V_i denote the vector $(U_{1i},\dots,U_{di})(i=1,\dots,N)$.

In the simplest setting, $V_1,\dots,V_N$ will cover the lattice $\ell(M,d)$, so that $N=M^d$. Thus V_i will be the point in the lattice which is the radix-M representation of i-1. As mentioned above, the kinds of applications being considered here are those in which the simulation study involves some updatable procedure Π. Let $\Pi(k)$ denote the result of determining the value of Π on Node k of the lattice (k=0,1,...,N-1). Then if Node k is an immediate neighbor of Node k-1 (i.e. differing only in the last coordinate), $\Pi(k)$ can be obtained from updating $\Pi(k-1)$. The proposed simulation algorithm is then the following:

```
initialize sums (e.g. counts for probabilities);
for i:=1 to N do
    begin
    if (i mod d = 1) then
       find Π(i-1) from scratch
    else
       find Π(i-1) by updating the value found
          previously for Π(i-2);
    add to sums
    end;
report results
```

Example

Suppose a system consists of 6 components, having independent, exponential lifetimes with means 1.0,2.0,...,6.0 hours, and that we need to know the probability that at least two failures occur in some 0.2 hour period of time. When the above algorithm was applied with M = 4, the simulation used 16.8 seconds of CPU time on a VAX 750. However, the standard Monte Carlo algorithm consumed 34.9 seconds of CPU time. Thus the algorithm introduced above achieved a 52% reduction in time. [With regard to accuracy, the proposed algorithm reported a probability of 0.6099, while the standard method reported 0.6150. The true value is 0.6127.]

3. USE IN HIGH DIMENSIONS

The method introduced in Section 2 exploits updatability by sampling every node in $\ell(M,d)$. This may not be feasible for larger values of d. For example, even $\ell(2,20)$, with its minimal value of M and moderate value of d, has more than a million nodes.

The reasonable solution would seem to be sampling of some regular subset of $\ell(M,d)$, say every k-th node in the radix-M ordering used here. However, this may mean that we would not be able to take advantage of updatability: The method in Section 2 implicitly utilized the fact that two nodes which were consecutive in the sampling process are usually immediate neighbors in the lattice.

To analyze this aspect, suppose we wish to sample N nodes in all, with the nodes spaced k nodes apart, as indicated above. Then

$$k \approx \frac{M^d}{N}. \tag{3}$$

This implies that the radix-M representation of K will have

$$r \approx d - \log_M N \tag{4}$$

digits. Thus, since two nodes which are consecutive in this new sampling process will usually differ only in their last r digits, the two nodes will typically have about $\log_M N$ digits in common. This commonality may serve as a basis for taking advantage of updatability.

For example, consider the case of sorting, with d = 20, M = 5, and N = 1000. Two consecutively sampled nodes will then typically have the first 4 of their 20 digits in common. Thus to obtain the second sorted list by updating the first, the last 16 digits of the new node can be sorted, and then a merge operation can be applied to these digits and the first 4 digits. The time ratio, compared to sorting the 20 elements from scratch, is about

$$\frac{20 \log_2 20}{16 \log_2 15}, \tag{5}$$

i.e. a time savings of about 35%.

Note that much larger reductions in CPU time will be obtained for updatable operations having much greater time complexity than that of sorting. For example, the Traveling Salesman Problem with n points requires $O(n2^n)$ comparisons [6]. In a simulation study of this problem with randomly generated points in the plane (d = 2n), quite large savings in time can be achieved.

REFERENCES

1. Fishman, G., "Antithetic Variates and Quasirandom Points as Variance Reduction Techniques," Proceedings of the 1983 Winter Simulation Conference, 557-558.

2. Fox, B., "Counterparts of Variance Reduction Techniques for Quasi-Monte Carlo Integration," Proceedings of the 1983 Winter Simulation Conference, 621.

3. Halton, J., "On the Efficiency of Certain Quasi-Random Sequences of Points in Evaluating Multidimensional Integrals," Numerische Mathematik, 2, 84-90, 1960.

4. Knuth, D., The Art of Computer Programming: Seminumerical Algorithms, Second Edition, Addison-Wesley, Massachusetts, 1981, Section 3.4.1.

5. Niederreiter, H., "Quasi-Monte Carlo Methods and Pseudo-Random Numbers," Bulletin of the American Mathematical Society, 84, 957-1041, 1978.

6. Reingold, E., Nievergelt, J., and Deo, N., Combinatorial Algorithms: Theory and Practice, Prentice-Hall, New Jersey, 1977, Section 4.1.

7. Rubinstein, R., Simulation and the Monte Carlo Method, Wiley, New York, 1981.