TECHNICAL OPINION

George Marsaglia

Seeds for Random Number Generators

Techniques for choosing seeds for social and scientific applications of random number generators.

The relatively short periods and meager sets of possible m-tuples $(x_{i+1},\ldots,x_{i+m})$ produced by single-seed random number generators (RNGs) has led to development of RNGs that use many seeds, have extremely long periods, and the ability to produce all or almost all of the possible m-tuples, for m's ranging from a few to thousands. While such attributes may be desirable for scientific applications, for certain applications in law, or under the supervision of regulatory agencies, use of multiple-seed RNGs may be mandatory. This column discusses this situation and suggests some good multiple-seed RNGs and ways to choose their seeds when many may be required and the process may have to be documented.

Random Number Generators and Seeds

A RNG in the context of this column is a computer program that, given a certain set of random values, called the **seeds**, produces a random output whose values are a fixed function of the random input. Thus the output from a RNG is completely determined by the random input. If there are N ways to choose the random seeds, then the number of possible outcomes cannot exceed N.

The choice and number of seeds may be vital to proper use of a RNG for certain applications. The following examples serve to illustrate the difference between what might be called scientific applications and social applications of RNGs. Suppose the RNG returns a selection of six different numbers from the set 1,2,3,…,49, that is, it chooses a particular kind of lottery ticket. Suppose that process depends on a single random integer seed value. Suppose further, as is often the case, that the seed value is determined by the 16 bits in the CPU's clock register.

There will be only 65,536 possible random seed values, and thus the RNG can only choose that many of the $\binom{49}{6}$ = 13,983,816 possible lottery tickets. That might be suitable for a scientific application—for example when a statistician, to refute claims that lottery draws containing a pair of adjoining numbers are rare, has the RNG produce 10,000 tickets from that restricted set, and finds that 4,996 of them had adjoining pairs, (example: 11,17,23,24,37,41), a result quite consistent with the probability for the entire set, 0.5048. It often happens that for elements with a particular property, the distribution in the restricted set is similar to that in the entire set, and thus simulations based on the RNG's sampling from the restricted set provide accurate estimates of the true situation.

Now consider a social application: each week you buy 10 lottery tickets at your local convenience store, and you let the store's device choose your 10 tickets randomly. That device uses its 16-bit clock register to initiate its RNG. You think "That's OK, any particular ticket has the same chance as any other." But a similar device is used by hundreds of other ticket sellers; all of the tickets produced are in a small (65,536) subset of the 13,983,816 tickets. If you happen to hold a winning ticket, chances are you will have to share the prize with several others whose tickets came from that restricted set.

As will be discussed later, such considerations have led to reappraisal of gaming devices that use RNGs. But first, here is a social application of RNGs that has raised serious problems in law: The jury in most trials, civil or criminal, is chosen from a **venire**—a panel of prospective jurors chosen from lists of citizens. Most state codes permit venire selection to be made by use of computers, provided that selection is *by lot and at random*. The most common procedure has been to use a random number generator with a single seed to make specific selections from the list of eligibles, and the seed is usually a 10-digit number, often chosen by the computer clock in a manner hidden in proprietary software.

Thus the venire selection process can only provide some 10^{10} possible venires, and it is difficult to argue that such a selection was, as the law requires, by lot and at random. While it can be considered a random selection, it cannot be considered a "selection by lot," as an overwhelming majority of the possible venires can never be selected.

For example, consider a simple case: a small county must select a venire of 80 from a list of 200 eligibles. There are $\binom{200}{80}$ ways to choose such a venire, since, by law, all venires must be given equal weight. Now $\binom{200}{80}$ is a number of 58 digits. The RNG can only select an exceedingly small fraction of the full set—and that is for a simple case. Consider a real-life selection of 1,200 from a list of 500,000 eligibles in Palm Beach County. The number of equal-weight possibilities is an integer with 3,662 digits.

The legal system's requirement that selection be by lot and at random implies a litigant is entitled to the chance his jury panel will have, if not a majority, then at least a few jurors who might tend to favor his case. Questions raised over the common practice of using a single-seed RNG to select jury venires (see [1]), have led to reappraisal of the selection procedure. New procedures require the number of possible random seed selections exceed the number of possible venires, but another concern remains: the possibility that convicted felons will demand new trials because the venire selection process for their trial did not satisfy the requirement that selection was "by lot and at random."

Questions have also arisen over the use of RNGs in gaming machines. For example, players of a machine that plays simultaneous hands of poker should be entitled to the chance that all hands will be straight flushes. Since an ordinary single-seed RNG can provide only a relatively tiny subset of the possible shuffles of a single deck ($52! > 10^{67}$), such a social application of RNG's calls for RNG's with many seeds. This is evidenced, for example, by the Michigan Game Control Board's requirement that certain gaming machines have multiple-seed RNGs before being licensed.

Multiple-Seed Random Number Generators

The preceding examples indicate that social applications may require the RNG is able to select from every possible outcome, a requirement that can be satisfied with RNGs having many random seed values. For scientific applications, the usual single-seed RNG may serve for problems where the features in question are expected to have distributions in the restricted set that mirror those in the entire set, but only for sample sizes less than the number of possible seeds. Single-seed RNGs typically have periods around 2^{32}, through which a modern computer can run in a few minutes. Scientific applications may require RNGs with multiple seeds for a more important reason: every possible m-tuple of integers should be available for reasonably large m. Many potential problems with RNGs arise from their performance in higher dimensions—for example, most single-seed RNGs can produce only $1/2^{32}$ of the possible pairs of integers (x_i, x_{i+1}), only $1/2^{64}$ of the possible 3-tuples (x_i, x_{i+1}, x_{i+2}), and so forth.

Thus, multiple-seed RNGs seem desirable for some applications and mandatory for others. Fortunately, several good multiple-seed RNGs are available. The widely used "Universal" RNG [3] uses 97 seed values and returns uniform (0,1) floating point numbers directly, using only subtraction. The Mersenne Twister [4] is a 32-bit RNG that uses 624 seeds and has passed extensive tests of randomness. But it requires a relatively complicated program. Versions of recently developed complimentary-multiply-with-carry RNGs use from a few to several thousand seeds. They have passed all tests of randomness put to them, particularly the Diehard battery of tests [2],

Social applications may require the RNG is able to select from every possible outcome, a requirement that can be satisfied with RNGs having many random seed values.

and can be implemented with brief programs in C, Java, or other languages.

The following example, in C, takes 1,024 seeds:

```
static unsigned long Q[1024];

unsigned long CMWC(void){
unsigned long long t, a=123471786LL;
static unsigned long c=362436,i=1023;
unsigned long x,r=0xfffffffe;
     i=(i+1)&1023;
     t=a*Q[i]+c;
     c=(t>>32); x=t+c; if(x<c){x++;c++;}
     return(Q[i]=r-x);  }
```

This complimentary-multiply-with-carry RNG is based on the recursions

$x_n = (b\text{-}1) - (ax_{n-1024} + c_{n-1} \bmod b)$,

$c_n = \lfloor (ax_{n-1024} + c_{n-1}/b) \rfloor$

with $b=2^{32} - 1$ and $x_0, x_1,\ldots,x_{1023}$ the initial 32-bit seeds in the array `Q[1024]` and with initial c any choice in $0 \leq c<a$. It is good practice to provide the `Q[1024]` array with a set of default seeds—for example with

```
j=123456789;   for(i=0;i<1024;i++){j^=j<<13;
j^=j>>17; j^=j<<5; Q[i]=j}
```

(Perhaps using a seed of choice rather than 123456789 for the shift register RNG.) Then any number of up to 1,024 seeds can be set among `Q[0],Q[1]`,... to supplement the default set. For scientific applications, a few seed values might do, but for applications that must be able to return any one of a huge number of possible outcomes, enough seeds must be chosen to meet that requirement.

For versions with fewer than the maximum of 1,024 seeds in the preceding listing,

Use `Q[512]` and `a=123554632LL` and replace 1,023 by 511.
Use `Q[256]` and `a=8001634LL` and replace 1,023 by 255.
Use `Q[128]` and `a=8007626LL` and replace 1,023 by 127.
Use `Q[64]` and `a=647535442LL` and replace 1,023 by 63.
Use `Q[32]` and `a=547416522LL` and replace 1,023 by 31.
Use `Q[16]` and `a=487198574LL` and replace 1,023 by 15.
Use `Q[8]` and `a=716514398LL` and replace 1,023 by 7.

Any one of the choices for seed table size and multiplier will provide a RNG that has passed extensive tests of randomness, particularly those in [3], yet is simple and fast—approximately 30 million random 32-bit integers per second on a 850MHz PC. The period is ab^n, where a is the multiplier, n the size of the seed table and $b=2^{32}-1$. (a is chosen so that b is a primitive root of the prime ab^n+1.)

Getting the Seeds

For applications where use of many random seeds is necessary, the problem of getting them is not trivial, if the process is, as in law, a formal one that may be subject to review and verification. Recently available devices that use thermal noise can provide suitably random bytes to a computer, and if the process does not have to be recorded or documented, can serve well. Of course a set of witnesses and affidavits on the results of reading the output from a random noise generator could serve, but there is a better way that does not depend on such devices or formal monitoring of output.

First, suppose we need several hundred random seeds, which can be taken as 10-digit integers. Those digits can be randomly chosen one at a time as follows: On a specified day in the future, use—one after another—the digits in the daily sales of stocks for some 3,000 listed on the New York Stock Exchange. Such data is virtually unpredictable, readily available and a matter of public record after each closing. The individual digits may not be uniform in {0,1,2,3,4,5,6,7,8,9}, but they can lead to those that are: use a good single-seed RNG to generate a sequence of uniform random digits from 0 to 9.

Then add each of those digits to the daily-sales digits, taking the result mod 10. It is easy to see that if *I* is a uniform random digit, then so is *J*+*I* mod 10, where *J* is any digit, random or fixed, independent of *I*.

The result of combining, mod 10, the not-necessarily-uniform daily-sales digits with the uniform RNG digits will, taken 10 at a time, provide the necessary seeds. Exact specification of the days of collection and the exact order and number of sales whose digits are to be used, together with the RNG and its seed, can provide formal documentation and means of verification.

For example, a `nyclose` file for a business day in July reports data for 3,526 stocks traded on the New York Stock Exchange. The first column gives the stock symbol, starting with (columns changed to rows to save space):

A,AA,AAE,AAGPRT,AAM,AAR,AAS,AAT,ABB, ABF,ABG,ABI,ABJ,…

The last column of that NYSE file provides the day's 'Total Volume' for those stocks:

2943000,2277300,202200,1300,11200,4600,8636 00,2600,23300,244000,…,

Taking these digits one at a time provides the first of three rows of digits. The second is a row of random digits produced by a RNG, then the final row is obtained by adding, mod 10, the stock market digits and the random digits:

2,9,4,3,0,0,0,2,2,7,7,3,0,0,2,0,2,2,0,0,1,3,0,0,1,1, 2,0,0,4,6,0,0,8,6,3,6,0,0,2,6,…
2,6,4,9,0,4,7,2,1,9,0,4,3,2,7,6,9,3,7,7,0,3,2,5,9,3, 8,7,9,9,5,7,3,7,2,8,0,6,3,2,8,…
4,5,8,2,0,4,7,4,3,6,7,7,3,2,9,6,1,5,7,7,1,6,2,5,0,4, 0,7,9, 3,1,7,3,5,8,1,6,6,3,4,4,…

The final sequence of digits can be taken 10 at a time to form the necessary sequence of random 10-digit seeds:

4582047436, 7732961577, 1625040793, 1735816634, …

It is a routine task to write a computer program that will extract total volume values from a specified list of stocks, take the digits one after another and then combine them, mod 10, with random digits from a single-seed RNG. After that, group the resulting digits to form enough random 10-digit seeds for a multiple-seed RNG to provide as many possible outcomes as the application at hand requires.

The preceding procedure is recommended for applications such as in law, where the procedure for choosing the necessary number of uniformly random seeds may have to be documented, specified in advance, "unfixable," and available for review afterward. In many applications such precautions may not be necessary or desirable; one may merely want to get the number of uniformly random seeds called for in the application, by any convenient means. The important point is to keep in mind the randomness of the results from a RNG is a direct consequence of the randomness of its seeds, the number of which should be governed by the fact that totality of results cannot exceed totality of seed choices. C

References

1. Marsaglia, G. Problems with the use of computers for selecting jury panels. *Jurimetrics 41* (Summer 2001), 425–427.
2. Marsaglia, G. *The Marsaglia Random Number CD-ROM, with The Diehard Battery of Tests of Randomness.* Produced at Florida State University under a grant from The National Science Foundation, 1995; www.stat.fsu.edu/pub/diehard. A new version of Diehard is available at www.csis.hku.hk/~diehard.
3. Marsaglia, G., Zaman, A., and Tsang, W. Toward a universal random number generator. *Statistics and Probability Letters 8* (1990), 35–39.
4. Matsumoto M. and Nishimura, T. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation 8*, 1 (1998), 3–30.

George Marsaglia (geo@stat.fsu.edu) is Professor Emeritus at Florida State University in Tallahassee.