The Wheel of Fortune

Brian Hayes

Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin." These wry words were written 40 years ago by John von Neumann, the Hungarian-American mathematician and pioneer of computing. He went on to explain: "As has been pointed out several times, there is no such thing as a random number—there are only methods to produce random numbers, and a strict arithmetic procedure of course is not such a method."

Von Neumann's argument would seem to rule out any possibility of having a computer generate random numbers. A properly functioning digital computer must at all moments follow an algorithm—von Neumann's "strict arithmetic procedure"—which means the computer's actions are entirely deterministic. The path taken by the algorithm may be tangled and tortuous, but in principle every step could be predicted in advance; the outcome is never a matter of chance or caprice. If you were to execute the algorithm again and again from the same initial state, the result would always be the same.

The computer's incomprehension of randomness is troublesome in certain areas of scientific computing, which demand vast quantities of chaos. One of the most voracious consumers of random numbers is the technique called the Monte Carlo method, named in honor of the well-known generator of random integers (between 0 and 36) in the Mediterranean principality. Securing an adequate supply of high-grade disorder for Monte Carlo calculations has been a challenge for some decades. Recent observations indicate that the problems are not yet fully solved.

Divagations of a Neutron

The idea behind the Monte Carlo method goes back two centuries or more, but the modern form of the technique was invented at Los Alamos shortly after World War II. The inventor was Stanislaw Ulam, who was working on a problem in the diffusion of neutrons (Metropolis and Ulam 1949). Consider a neutron passing through a lump of uranium. The neutron collides with many atomic nuclei, and in each collision it can either bounce off the nucleus or else be absorbed by it; in the latter case there is a chance the nucleus will undergo fission and thereby liberate more neutrons. Ulam was trying to estimate how many neutrons would eventually escape from the lump and how many would remain behind to sustain a fission reaction. This is a question of some practical interest.

When Ulam took up the problem of neutron diffusion, the individual collision events were already well understood. For any single collision, experimental data gave the probability of scattering or absorption or fission. Nevertheless, working out the ultimate fate of the average neutron was beyond the capacity of conventional mathematical methods. The equations defining the sums and products of all those probabilities were too large to be solved directly.

Ulam's answer was to play the part of a neutron. He would imagine moving through the crystal lattice, occasionally colliding with atomic nuclei. At each collision he would decide randomly what should happen next, based on the known probabilities. One way to make these random decisions is by spinning a roulette wheel. For example, if scattering is twice as likely as absorption, and fission is rather unlikely, then the following scheme might work. Whenever the wheel produces a number between 1 and 24, the neutron is scattered; a number from 25 through 36 indicates absorption; and a 0 predicts fission. (The real probabilities are more complicated, and they depend on factors such as the neutron energy and whether the nucleus is uranium 235 or uranium 238.) By following a neutron for hundreds of collisions, and then repeating the calculation for thousands of neutrons, one can estimate important statistical properties of the neutron trajectories.

Ulam refined and developed his method in collaboration with von Neumann, Nicholas Metropolis and other colleagues at Los Alamos. Within a few years it had been applied to a variety of problems in physics, statistical mechanics and chemistry. Today it is an indispensable tool in fields ranging from solid-state physics to economics.

Manufacturing Randomness

In principle, Monte Carlo calculations can be done by hand, but problems of practical size can be undertaken only with high-speed computing machinery. (It is no accident that the technique evolved within one of the first groups of people to have regular access to such machinery.) But if a computer is to carry out a Monte Carlo simulation, the computer must have the equivalent of a roulette wheel—that is, some source of random numbers.

Random-number generators based on unpredictable physical phenomena such as noise in electronic circuits and the decay of radioactive nuclei have occasionally been employed in Monte Carlo studies, but they have certain shortcomings. There is no way to repeat a sequence of numbers, which complicates the debugging of programs. What is

Guest columnist Brian Hayes is a former editor of American Scientist *who has written on computing for several publications. Address: 102A Roundhill Road, Chapel Hill, NC 27514. Internet: hayes@concert.net. Peter Denning will return in the next issue.*

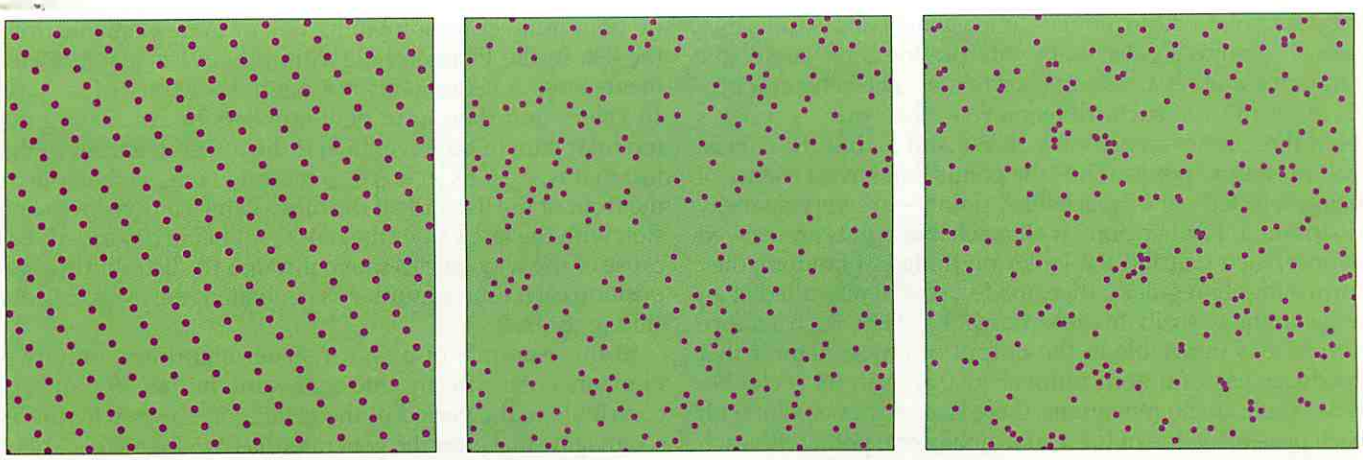

Figure 1. Pseudo-random numbers can exhibit a surprising amount of regularity. In each of these panels 256 pairs of pseudo-random numbers are interpreted as the x and y coordinates of points in a two-dimensional space. At left is the output of a linear-congruential generator whose period is just 256, so that the entire production of the algorithm is included in the diagram. Because each integer from 0 through 255 appears exactly once in the sequence of numbers, the points form a crystal-like lattice. In the middle panel 256 numbers have been selected from a linear-congruential generator whose period is 1,024; the regularity of the pattern has been lessened somewhat but is still apparent. At right the pattern is suppressed altogether in the output of a shift-register generator, which has a period of more than 10^{31}. Nevertheless, recent experiments have exposed apparent flaws in some shift-register algorithms.

more important, generators based on physical effects are too slow; they cannot satisfy the random-number appetite of modern Monte Carlo simulations.

The alternative to a hardware random-number generator is von Neumann's "strict arithmetic procedure," and his state of sin. The idea is to find an algorithm whose output is a sequence of numbers that "looks" random, and which can pass statistical tests of randomness, even though the procedure is entirely deterministic. Von Neumann himself invented one of the earliest generators of such "pseudo-random" numbers. It is called the middle-square method: Begin with an arbitrary n-digit number, or "seed"; square the seed to yield a number having $2n$ digits; then select the middle n digits to be the next pseudo-random value and the seed for further iterations of the procedure. For example, an initial seed of 6744 is squared to yield 45481536, with the middle digits 4815 selected as the next seed value. Continuing in the same way, the next few values in the series are 1842, 3929, 4370, 0969 and 9389. This sequence certainly looks patternless, but the appearance does not last. After just three more iterations the generator stumbles onto the value 2500, whose eight-digit square is 06250000; the middle digits are again 2500, and so the series becomes eternally stuck on this one value. Middle-square sequences begun with many other seeds degenerate into similar fixed points or short cycles.

Better algorithms for generating pseudo-random numbers were soon devised, and along with them a body of theory and practice for evaluating the quality of the results (Knuth 1981). For some years the most popular generators have been those described as linear-congruential generators, invented by D. H. Lehmer at about the same time as von Neumann's middle-square method. Linear congruential generators are based on a three-step algorithm: Multiply a seed value by a constant, r; add another constant, c; and finally reduce the result by a modulus, m, or in other words divide by m and retain only the remainder. Much attention has been given to the selection of optimum values for the constants r, c and m. What is probably the most widely used generator today has the values $r = 16{,}807$, $m = 2^{31} - 1$ and $c = 0$ (Park and Miller 1988). The output of this generator is a stream of numbers whose appearance is very convincingly random.

Running Out of Numbers

If a sequence of numbers is truly random, then knowing any one number in the sequence gives you no information that would help you guess any other number. Judged by this standard, the sequences generated by both the middle-square method and the linear-congruential algorithm are not random at all, since knowing any one value allows you to predict all subsequent values. Nevertheless, the numbers have statistical properties indistinguishable from those of genuine random sequences. For example, in a uniformly distributed random stream of decimal digits, all 10 digits from 0 to 9 should be equally represented on the average, as should all 100 pairs of digits from 00 to 99 and all 1,000 triplets from 000 to 999. Good linear-congruential generators pass this test, and many other empirical tests as well. The numbers they produce appear patternless. If you were to look at a true random sequence and at a pseudo-random sequence, you could not tell which is which.

But the numbers produced by common implementations of Lehmer's method do have one serious failing: There just aren't enough of them. Every number produced by a linear-congruential generator is an integer less than m, and so the generator can yield no more than m distinct values. If it is asked for $m + 1$ numbers, it must repeat itself at least once. Furthermore, because each number in the sequence is determined entirely by the preceding number, repeating even one value leads into a cycle that the generator can never escape. Suppose 57 is followed by 43 and then by 92 when the generator is first started; when 57 appears again, it must necessarily be followed by 43 and 92 again. From that moment on, the generator is stuck in its own rut.

The best linear-congruential generators have the longest possible period before repetition, namely m (or $m - 1$ for generators with $c = 0$). The popular m value $2^{31} - 1$ is equal to a little more than 2 billion, which seems like a great many random numbers, but there are computers on desktops today that could run through that stock of numbers in 10 minutes. Large Monte Carlo studies demand an ampler supply.

The shortage of random numbers becomes particularly acute when the numbers are used in pairs or triplets or larger groups. Suppose a Monte Carlo study calls for choosing a

random point within a three-dimensional cube m units on a side. The point is selected by interpreting three successive random values as x, y and z coordinates. The generator produces at most m such triplets (even when each number is used three times—once each as x, y and z), but the cubical volume has m^3 points. Thus the points that have a chance of being selected—the "reachable" points—are very sparsely distributed. Furthermore, if all reachable points are viewed at once, their distribution looks anything but random; they form a highly regular lattice, made up of planes tilted at an angle to the coordinate axes (Marsaglia 1968). Such a lattice structure is inevitable in the output of any generator that produces each random number just once in its cycle. Because there are no repetitions, there is exactly one point with each possible value of the x coordinate, one point with each possible y value and one point with each z value.

Shift-Register Methods

To avoid these failings, some workers have turned away from linear-congruential generators to algorithms of another class, known collectively as shift-register algorithms. In a shift-register algorithm, each newly generated number depends on many previous values, not just on the immediately preceding one. Hence the reappearance of a single number does not cause the algorithm to enter a cycle; that happens only with the repetition of the entire set of numbers that enter into the calculation of the next element of the sequence. Therefore, a shift-register algorithm can have a very long period.

The simplest shift-register algorithm is based on the Fibonacci series, in which each number is the sum of the previous two numbers—in other words, $X_n = X_{n-1} + X_{n-2}$. This formula can be made to yield pseudo-random numbers between 1 and a modulus m by carrying out the addition modulo m. Unfortunately, the sequences created by this specific generator do not perform very well on statistical tests of randomness. More elaborate generators based on similar principles, however, yield very good results.

Shift-register algorithms were already known in the 1950s (Green, Smith and Klem 1959). The basic idea is to retain a list of prior values and then to generate each new pseudo-random number by combining two or more elements from the list. In the Fibonacci algorithm the list is just two elements long. A better generator might keep track of the past 15 values and then form new numbers by calculating the modular sum of the seventh and the fifteenth entries in the list; that is, $X_n = (X_{n-6} + X_{n-14})$ modulo m. X_n is then put at the front of the list and all the other values move down one slot, with the last value (the old X_{n-14}) being discarded. Because of the way entries move through the list, shifting one position each time a number is generated, the list is called a shift register.

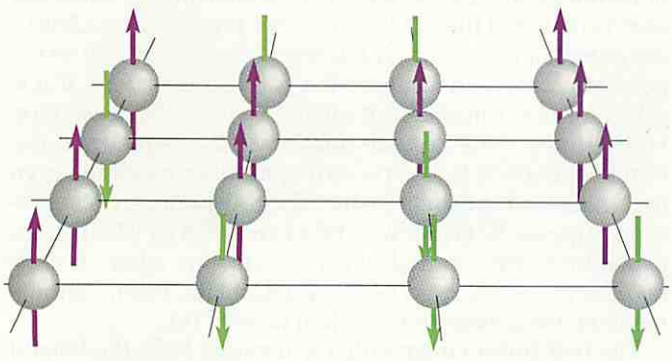

Figure 2. Ising model is a conceptual device of solid-state physics that is generally studied by Monte Carlo methods, which rely heavily on pseudo-random numbers. The model represents the microscopic structure of a ferromagnet in terms of atomic spins that can be either up or down. In traditional Monte Carlo studies a spin is selected at random and then either flipped or not, with a probability depending on the temperature and on the effects of neighboring spins. A new algorithm creates clusters of spins, which are then flipped as a group, speeding the calculation.

In this example of a shift-register algorithm, only two numbers enter directly into each sum, but all 15 numbers contribute to the period of the generator. To see why this is so, imagine that after the generator has run for a while, 14 of the 15 numbers have all returned to their original values, but one number—call it the oddball—remains different from its starting value. After a few more iterations, the oddball will have shifted down the list to position $n-6$ or $n-14$, at which point the new X_n must take on a value different from what it had during the first part of the run. Hence the generator has not entered a cycle; only when all 15 elements of the list have returned to their initial values does the output of the generator begin to repeat.

The maximum possible period of a shift-register generator is 2^{qp}, where q is the number of binary digits in each pseudo-random value and p is the number of values retained in the list. For a list of 15 numbers of 32 bits each, the maximum period is 2^{480}, which is equal to about 10^{144}. It is hard to imagine exhausting such a copious stockpile of numbers. But the shift-register algorithms have a drawback of their own. Whereas the linear-congruential method requires the user to supply only a single initial value as a seed, a shift-register algorithm demands p initial values in order to fill up the shift register. Moreover, the simplest such algorithms, like the one described above, turn out to be very sensitive to the choice of initial values. With a poor choice, the period may be much shorter than the maximum period.

To strengthen the shift-register algorithms, they have been made somewhat more complicated. One strategy is to maintain a longer shift register, or in other words to choose larger values of x and p in the formula $X_n = X_{n-x} + X_{n-p}$. (Guidance in selecting the best values of x and p comes from the mathematical theory of primitive polynomials.) Another approach is to replace simple modular summation with some other operation. Add-with-carry generators maintain an auxiliary variable, the carry bit, which is set to 1 whenever the sum $X_{n-x} + X_{n-p}$ exceeds some fixed limit; including this extra bit in the sum increases the generator's period considerably. Subtract-with-borrow generators employ an analogous principle. Still another family of shift-register algorithms replace modular addition with the exclusive-or (*xor*) operation, which is essentially binary addition modulo 2. In most computers *xor* is a single hardware instruction, and so *xor* generators can be very efficient (Kirkpatrick and Stoll 1981).

Spin the Atom

After 40 years of development, one might think that the making of random numbers would be a mature and trouble-free technology, but it seems the creation of unpredictability is ever unpredictable. Last year Alan M. Ferrenberg and David P. Landau of the University of Georgia and Y. Joanna Wong of IBM were preparing for a series of large-scale Monte Carlo

simulations. As a test of their programs, they made some preliminary studies of a problem for which exact results were already known. They got the wrong answers. Further checking eventually implicated the random-number generator. Ironically, the problem arose not with comparatively weak algorithms such as a linear-congruential generator but with some of the highly regarded shift-register methods.

The Georgia-IBM study is focused on the Ising model, a simplified representation of certain systems in solid-state physics, such as a ferromagnet, devised in the 1920s by the German physicists Wilhelm Lenz and Ernst Ising. In a ferromagnet, nearby atomic spins can reduce their energy by lining up in the same direction, but thermal perturbations disrupt the alignment; the equilibrium between these two tendencies determines the magnetization of the material. In the Ising model all the physical details of atomic structure are ignored. The atoms are represented by abstract sites in a lattice, and each site has just two discrete states, corresponding to spin-up and spin-down. Moreover, only interactions between immediate neighbors are taken into account; all longer-range influences are ignored.

The classical algorithm for studying the Ising model works as follows. First, select a site at random (in d dimensions this requires d random numbers). Examine the neighboring sites and calculate ΔE, the change in the energy of the system if the selected spin were reversed. The energy is reduced (that is, ΔE is negative) if flipping the selected spin brings it into alignment with a majority of its nearest neighbors. Next, based on ΔE and the temperature, calculate the probability of a spin flip. The probability is 1 (or in other words a spin flip is certain) when the transition is energetically favored. The probability declines toward 0 as ΔE becomes more positive and as the temperature is lowered. The final step in the algorithm is to calculate a random number X between 0 and 1. If X is less than the calculated probability, flip the spin; otherwise leave the spin unchanged. Now repeat the entire procedure a few thousand or million or billion times until the system reaches equilibrium.

In spite of its simplifications, the Ising model reflects some important properties of real ferromagnets. In particular, as the temperature is lowered through a critical value called the Curie temperature, the system undergoes a phase transition: It becomes magnetized. At high temperature the spins fluctuate rapidly, and there is no long-range order in their arrangement. As the temperature falls, the spins become organized into magnetic domains, which grow larger and more stable as the system cools further. This spontaneous emergence of order out of chaos near the critical point is of much interest; unfortunately, at just this point the algorithm begins to fail. As the spin fluctuations become slower, the number of Monte Carlo steps needed to reach equilibrium rises sharply, making accurate simulations impractical.

This issue of "critical slowing down" is addressed in a new

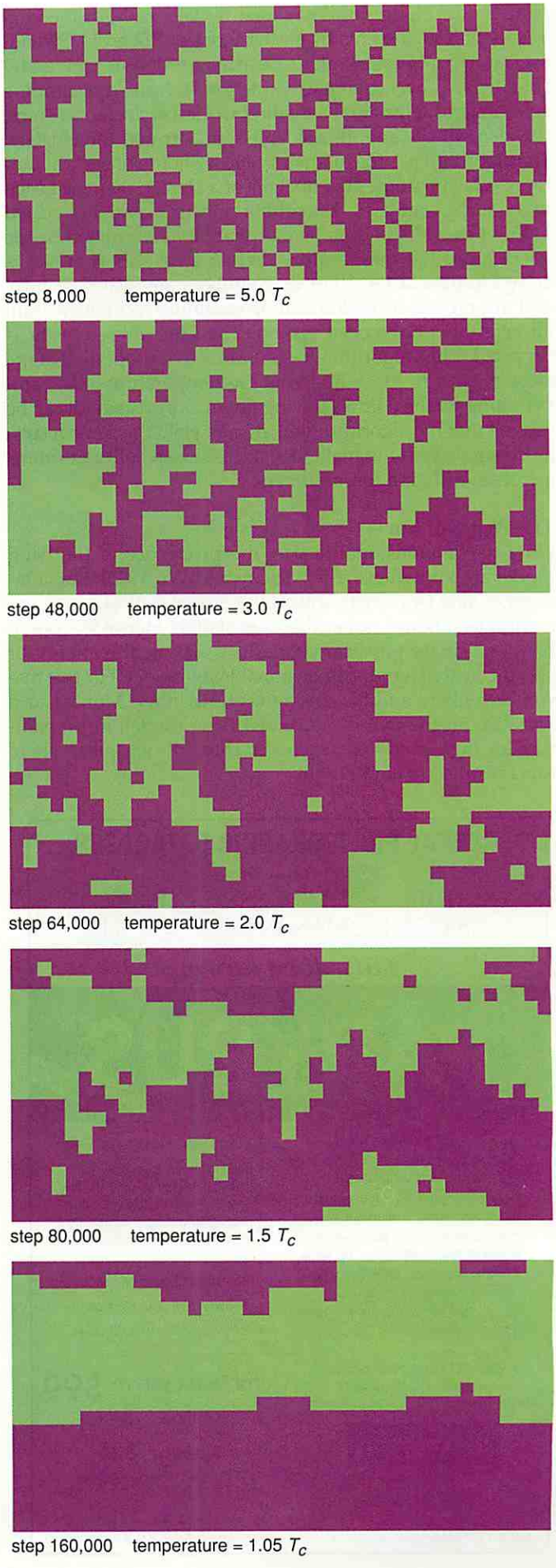

Figure 3. Evolution of the Ising model slows as the temperature falls toward a critical point, T_c. In these snapshots of a small two-dimensional Ising model, a purple square represents an up spin and a green square indicates a down spin. At high temperature there are small-scale variations in spin direction, and there are also rapid fluctuations in spin direction at any given site. As the temperature is reduced, the domains of aligned spin grow larger, and the fluctuations also become slower. Near the critical point, this "critical slowing down" makes the one-spin-at-a-time Monte Carlo algorithm impractical; cluster methods are faster, but they may be incompatible with certain random-number generators.

Ising-model algorithm developed by Ulli Wolff of the University of Kiel, inspired by earlier work of Robert H. Swendsen and Jian-Sheng Wang of Carnegie-Mellon University (Wolff 1989, Swendsen and Wang 1987). Instead of flipping spins individually, Wolff's method works on clusters of spins; the element of randomness enters in forming the clusters. The algorithm has the following steps. Choose one site at random, which becomes the nucleus of a cluster, then visit each of the site's neighbors. A neighbor may or may not be added to the cluster, depending on a probability determined by the energy and the temperature. When a neighboring site *is* added to the cluster, all of its neighbors are visited in turn, and this process of agglomeration continues recursively until all neighbors have been visited. Then the entire cluster is flipped. Wolff's algorithm has the attractive property that the clusters tend to grow larger as the temperature falls, compensating for the critical slowing down. Surprisingly, it can be proved that the algorithm will always yield the same results as the single-spin-flip method—at least when both algorithms are given truly random numbers.

The Wages of Sin

On a two-dimensional lattice, the properties of the Ising model are known exactly (McCoy and Wu 1973). Lars Onsager of Yale University solved the model in 1944, using analytical techniques rather than simulation. Hence it is possible to calculate physical properties such as the energy or specific heat of a two-dimensional Ising model. For this reason Ferrenberg and his colleagues tested their Monte Carlo software on the two-dimensional Ising model, in preparation for high-precision studies in three dimensions, where exact results are not known.

The disturbing outcome of this test was the discovery that the Wolff algorithm gives incorrect results when it is used with certain random-number generators. Specifically, it fails with two shift-register algorithms—an *xor* variant and a subtract-with-borrow algorithm. The discrepancies in calculated quantities such as specific heat are subtle—they generally show up in the fifth decimal place—but because the simulations attained very high precision, the errors are statistically significant. In one case the standard deviation is 42, and in another it is 107.

Ferrenberg and his colleagues have given an account of their findings in *Physical Review Letters* (Ferrenberg, Landau and Wong 1992). They have established that the source of the problem lies in some interaction between the Wolff algorithm and the shift-register generators. Running the Wolff algorithm with a linear-congruential generator (the ever-popular $r = 16{,}807$, $m = 2^{31} - 1$) gives results accurate to within one standard deviation. Conversely, the shift-register generators give acceptable results with the single-spin-flip algorithm for the Ising model, or with a different cluster algorithm invented by Swendsen and Wang. The findings have been confirmed by other workers, using independently written versions of the programs.

In the weeks since the report was published, Ferrenberg has continued to search for the cause of the failures. He speculates that subtle correlations in the most significant bits of the random numbers may create a slight bias in the size of the clusters formed in the Wolff algorithm. Such correlations were not detected in earlier statistical tests of the random-number generators, but the Wolff algorithm itself may constitute a more sensitive test than any of the standard ones. Ferrenberg is now trying to learn more about the aberration by running the simulations with random-number generators known to have specific biases.

The Georgia-IBM program of Monte Carlo studies remains stalled until the defect is understood. "This problem is *consuming* me," Ferrenberg says, with frustration and discouragement in his voice. "You know what John von Neumann said about random numbers and sin? I think I'll put that up over my door." One is reminded of another famous entryway warning to sinners: "Abandon all hope, you who enter here."

References

Ferrenberg, Alan M., D. P. Landau and Y. Joanna Wong. 1992. Monte Carlo simulations: Hidden errors from "good" random number generators. *Physical Review Letters* 69:3382–3384.

Green, Bert F., Jr., J. E. Keith Smith and Laura Klem. 1959. Empirical tests of an additive random number generator. *Journal of the Association for Computing Machinery* 6:527–537.

Kirkpatrick, Scott, and Erich P. Stoll. 1981. A very fast shift-register sequence random number generator. *Journal of Computational Physics* 40:517–526.

Knuth, Donald. 1981. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms.* Second edition, pp. 1–177. Reading, Mass.: Addison-Wesley.

Marsaglia, George. 1968. Random numbers fall mainly in the planes. *Proceedings of the National Academy of Sciences of the U.S.A.* 61:25–28.

McCoy, Barry M., and Tai Tsun Wu. 1973. *The Two-Dimensional Ising Model.* Cambridge, Mass.: Harvard University Press.

Metropolis, Nicholas, and S. Ulam. 1949. The Monte Carlo method. *Journal of the American Statistical Association* 247:335–341.

Park, Stephen K., and Keith W. Miller. 1988. Random number generators: Good ones are hard to find. *Communications of the ACM* 31:1192–1201.

Swendsen, Robert H., and Jian-Sheng Wang. 1987. Nonuniversal critical dynamics in Monte Carlo simulations. *Physical Review Letters* 58:86–88.

Von Neumann, John. 1951. Various techniques used in connection with random digits. *Journal of Research of the National Bureau of Standards: Applied Mathematics Series* 12:36–38.