

On the Generation of High-Quality Random Numbers by Two-Dimensional Cellular Automata

Marco Tomassini,
Moshe Sipper, *Senior Member, IEEE*, and
Mathieu Perrenoud

Abstract—Finding good random number generators (RNGs) is a hard problem that is of crucial import in several fields, ranging from large-scale statistical physics simulations to hardware self-test. In this paper, we employ the cellular programming evolutionary algorithm to automatically generate two-dimensional cellular automata (CA) RNGs. Applying an extensive suite of randomness tests to the evolved CAs, we demonstrate that they rapidly produce high-quality random-number sequences. Moreover, based on observations of the evolved CAs, we are able to handcraft even better RNGs, which not only outperform previously demonstrated high-quality RNGs, but can be potentially tailored to satisfy given hardware constraints.

Index Terms—Cellular automata, random number generators, evolutionary algorithms.

1 INTRODUCTION AND PREVIOUS WORK

PSEUDORANDOM number generation by cellular automata (CA) has been an active field of research in the last decade [1], one of the underlying motivations stemming from the advantages offered by CAs when considered from a VLSI viewpoint: CAs are simple, regular, locally interconnected, and modular. These characteristics make them easier to implement in hardware than other models, thus making CAs an attractive choice for onboard applications. CAs have traditionally been used to implement RNGs in cryptographic devices [2] and in Built-In Self-Test (BIST) circuits [3]. Random number generators play an import role in several computational fields, including Monte Carlo techniques, Brownian dynamics, and stochastic optimization methods (such as simulated annealing and genetic algorithms). With the advent of massively parallel scientific computation, the parallel generation of pseudorandom numbers has become essential.

The above domains depend critically on the quality of the random numbers, as measured by appropriate statistical tests. Moreover, when very long sequences of random numbers are needed, computational efficiency is often of prime import, i.e., the sequence must be produced as rapidly as possible. CAs provide a good solution to this problem, able to rapidly produce high-quality random-number streams.

One-dimensional CA random number generators have been extensively studied in the past [1], [3], [4], [5]. These studies have shown convincingly the suitability of CA-generated pseudorandom numbers and their superiority with respect to other widely used methods, such as linear feedback shift registers (LFSRs), especially in the case of delay type faults which require pairs of patterns in a specified order [6]. In these works, CA RNGs were essentially handcrafted by studying the structure of the bit patterns generated over time, with theoretical results serving as a baseline offering guidance.

- M. Tomassini and M. Perrenoud are with the Institute of Computer Science, University of Lausanne, CH-1015 Lausanne, Switzerland. E-mail: Marco.Tomassini@iismail.unil.ch.
- M. Sipper is with the Logic Systems Laboratory, Swiss Federal Institute of Technology, IN-Ecublens, CH-1015 Lausanne, Switzerland.

Manuscript received 28 June 1999; accepted 8 June 2000.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 110139.

Another possibility is to let the CA random number generator be evolved automatically by a genetic algorithm [7], [8]. A first step in the evolution of one-dimensional CA RNGs was taken by us in [9]. In a later work, we confirmed and extended these results by generating much longer pseudorandom sequences and by subjecting them to a more stringent and elaborate suite of randomness tests [10]. We concluded that: 1) Very good CA RNGs can indeed be evolved by a genetic algorithm and 2) with minimal (human) design the evolved CAs can be yet further improved.

Chowdhury et al. [11] described a methodology for producing pseudorandom numbers by two-dimensional cellular automata. Their results suggest that two-dimensional CAs are superior to one-dimensional ones of the same size (i.e., with an equal number of grid cells) in terms of the quality of the resulting pseudorandom numbers.

In the present study, we extend upon these previous works, using a genetic algorithm to evolve two-dimensional CA RNGs. After a short description of CAs in the next section, we describe the evolutionary algorithm in Section 3. Section 4 delineates our results, showing that evolved two-dimensional CAs produce high-quality random-number sequences. In Section 5, we study the cycle lengths of our CAs. Finally, in Section 6, we offer some concluding remarks.

2 CELLULAR AUTOMATA

Cellular automata (CA) are dynamical systems in which space and time are discrete. A cellular automaton consists of an array of cells, each of which can be in one of a finite number of possible states, updated synchronously in discrete time steps, according to a local, identical interaction rule. Here, we will only consider Boolean automata in which the cellular state, $s_i \in \{0, 1\}$. The state of a cell at the next time step is determined by the current states of a surrounding neighborhood of cells. The cellular array (grid) is d -dimensional, where $d = 1, 2, 3$ is used in practice; in this paper, we shall concentrate on $d = 2$, i.e., on two-dimensional grids. The identical rule contained in each cell is essentially a finite state machine, usually specified in the form of a rule table (also known as the transition function), with an entry for every possible neighborhood configuration of states. The cellular neighborhood of a cell consists of itself and of the surrounding (adjacent) cells. For one-dimensional CAs, a cell is connected to r local neighbors (cells) on either side, where r is referred to as the radius (thus, each cell has $2r + 1$ neighbors). For two-dimensional CAs, two types of cellular neighborhoods are usually considered: five cells, consisting of the cell along with its four immediate nondiagonal neighbors (also known as the von Neumann neighborhood) and nine cells, consisting of the cell along with its eight surrounding neighbors (also known as the Moore neighborhood). In this work, we only consider 5-neighbor grids, thus limiting the already large search-space size; moreover, results exist only for this neighborhood type, which is also more amenable to hardware implementation. When considering a finite-size grid, *cyclic* boundary conditions are frequently applied, resulting in a circular grid for the one-dimensional case and in a toroidal one for the two-dimensional case. *Fixed*, or *null*, boundary conditions can also be used, in which the grid is surrounded by an outer layer of cells in a fixed state of zero. This latter case is usually easier to implement in hardware.

Nonuniform, or inhomogeneous, cellular automata function in the same way as uniform ones, the only difference being in the cellular rules that need not be identical for all cells [12]. Nonuniform CAs share the basic "attractive" properties of uniform ones: simplicity, parallelism, and locality [13].

3 EVOLUTION OF TWO-DIMENSIONAL, NONUNIFORM CAs

Though one-dimensional CA RNGs have been thoroughly studied (see Chaudhuri et al. [1] for a good review), the use of two-dimensional CAs for pseudorandom number generation has received little attention in the literature; indeed, the only detailed study we are aware of is that of Chowdhury et al. [11]. They found that two-dimensional CAs can generate better pseudorandom numbers than one-dimensional ones, while retaining their (easy) amenability to VLSI implementation. More recently, Cattel et al. gave a complete characterization of a particular type of 2D CA, with two connected lines of n cells each [6]. In this work, a number of properties of these special 2D CAs are derived, such as characteristic polynomials, minimal cost, and maximal length generators. The work described herein confirms the conclusions of Chowdhury et al. with respect to two-dimensional CAs, while surpassing their CAs in terms of quality. In this section, we describe the evolutionary method used to automatically generate our CA RNGs; this method is based on so-called genetic algorithms [7], [8].

A genetic algorithm is an iterative procedure that involves a constant-size population of individuals, each one represented by a finite string of symbols, known as the *genome*, encoding a possible solution in a given problem space. This space, referred to as the *search space*, comprises all possible solutions to the problem at hand. The algorithm sets out with an initial population of individuals that is generated at random or heuristically. At every evolutionary step, known as a *generation*, the individuals in the current population are *decoded* and *evaluated* according to some predefined quality criterion, referred to as the *fitness*, or *fitness function*. To form a new population (the next generation), individuals are *selected* according to their fitness and then transformed via genetically inspired operators, the most oft-used being *crossover* (combining two or more genomes to form novel offspring) and *mutation* (randomly flipping bits in the genomes). Iterating this procedure, the genetic algorithm may eventually find an acceptable solution, i.e., one with high fitness.

As with our previous one-dimensional CAs, the two-dimensional, nonuniform CA RNGs described herein were also obtained via a genetic algorithm known as *cellular programming* [9], [10], [12]. The goal of the evolutionary algorithm is to evolve “good” rule tables for a nonuniform CA, i.e., rules that give rise to high-quality sequences of random numbers. Cellular programming involves a single nonuniform CA, where each cell’s rule table is encoded as a bit string—the *genome*—and rule evolution in the grid is driven by a local fitness measure; the genetic operations of selection and crossover [7] are also performed locally between neighboring cells.

As in our previous studies, each and every cell of the evolving CA is assigned a fitness score according to the *entropy* E_h of its bit sequence in time. Let k be the number of possible values per sequence position (in our case CA states) and h a subsequence length. E_h (measured in bits) for the set of k^h probabilities of the k^h possible subsequences of length h is given by:

$$E_h = - \sum_{j=1}^{k^h} p_{h_j} \log_2 p_{h_j},$$

where h_1, h_2, \dots, h_{k^h} are all the possible subsequences of length h (by convention, $\log_2 0 = 0$ when computing entropy). The entropy attains its maximal value when the probabilities of all k^h possible subsequences of length h are equal to $1/k^h$; in our case, $k = 2$ and the maximal entropy is $E_h = h$. Higher entropy implies better fitness.

High entropy is a necessary, but by no means sufficient, condition for obtaining high-quality RNGs; it is simple enough to act as a fitness measure driving the evolutionary process, however,

a battery of tests must be applied after evolution to ascertain whether the evolved RNGs are indeed of high quality. The test results, described in the next section, show that our method does indeed produce high-quality RNGs.

Most of our evolutionary runs were performed with 8×8 CAs, with each of the 64 rule tables randomly initialized at the outset. The evolutionary algorithm is then run for a total of 200 generations. A generation consists of presenting the evolving CA with a random initial configuration and letting it run (in accordance with the rule tables) for 4,096 time steps. Fitness of each cell—i.e., entropy—is computed on the basis of the sequence of bits generated during these 4,096 steps, using a subsequence length $h = 4$. Each cell is thus assigned a fitness value, after which genetic operators (crossover and mutation) are applied [7]. These operators are local, i.e., crossover is applied only between the rule tables of adjacent cells (see Sipper [12] for details).

During our initial experiments, a cell’s rule table was unconstrained, i.e., the evolved rule (genome) could be any of the possible 2^{32} rules (a 5-neighbor rule is completely specified by 32 bits). Though the resulting two-dimensional CAs proved better than the previously evolved one-dimensional ones [10], they were not exceptional in that they did not surpass the performance of other high-quality CA RNGs (e.g., those of Chowdhury et al. [11]). This is probably due to the huge size of the search space— $(2^{32})^{64}$ for an 8×8 nonuniform grid—coupled with the relative sparseness of very high-quality solutions. To find better CAs, and based on our observation of the resulting rules in the one-dimensional case, we restricted the genomic representation of the rule tables to allow for any of the 64 additive rules—those involving only XOR and XNOR logic (again, since the CA is nonuniform, the search space is still quite large: $64^{8 \times 8}$, for an 8×8 grid).

In order to delineate our results, we introduce the following rule-numbering scheme (after [11]): Since there are 64 possible (additive) rules, we need 6 bits to describe a rule. Let $s_{i,j}(t)$ be the state of the cell at row i and column j , at time t . Its state at the next time step, $s_{i,j}(t+1)$ is then computed as follows:

$$s_{i,j}(t+1) = X \oplus (C \cdot s_{i,j}(t)) \oplus (N \cdot s_{i-1,j}(t)) \oplus (W \cdot s_{i,j-1}(t)) \\ \oplus (S \cdot s_{i+1,j}(t)) \oplus (E \cdot s_{i,j+1}(t)),$$

where \oplus and \cdot are the operations XOR and AND, respectively, and X, C (center), N (north), S (south), W (west), and E (east) are binary variables. C, N, S, W , and E denote whether the respective neighboring cell state is taken into account (a value of 1) or not (a value of 0). The binary variable X demarcates linear ($X = 0$) from nonlinear ($X = 1$) additive rules. The genome of a cell is then given by the 6-bit string $XCNWSE$. For example, rule 14 (001110) represents the following function:

$$s_{i,j}(t+1) = s_{i-1,j}(t) \oplus s_{i,j-1}(t) \oplus s_{i+1,j}(t).$$

4 RESULTS

As with previous experiments in cellular programming [12], we observed that the grid, being completely nonuniform (because random) at the outset, converges to a small number of distinct rules; this was dubbed a *quasi-uniform* grid by Sipper [12]. For example, an 8×8 grid may potentially contain 64 distinct rules, though after evolution takes place we find that there are usually four to nine distinct rules. Moreover, as described by Sipper [12], there is a tendency for some rules to “dominate” over others, i.e., the distribution of rules is not homogeneous; this is demonstrated in Fig. 1, which delineates one of our best evolved CA random number generators.

We observed that, when evolution converges to a grid with but one rule, the average fitness drops; fitness may possibly rise again

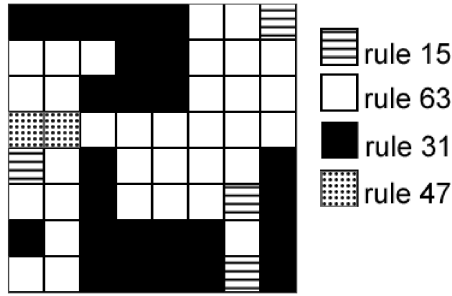


Fig. 1. An evolved 8×8 cellular automaton. The one shown above is among the best evolved, i.e., it produces among the highest-quality sequences of random bits. Shown above is a so-called *rules map* [12], delineating the rule present in each cell when evolution ends. The four rules—15, 31, 47, 63—correspond, respectively, to the following Boolean expressions:

$$\begin{aligned}
 s_{i,j}(t+1) &= s_{i-1,j}(t) \oplus s_{i,j-1}(t) \oplus s_{i+1,j}(t) \oplus s_{i,j+1}(t), \\
 s_{i,j}(t+1) &= s_{i-1,j}(t) \oplus s_{i,j-1}(t) \oplus s_{i+1,j}(t) \oplus s_{i,j+1}(t) \oplus s_{i,j}(t), \\
 s_{i,j}(t+1) &= 1 \oplus s_{i-1,j}(t) \oplus s_{i,j-1}(t) \oplus s_{i+1,j}(t) \oplus s_{i,j+1}(t), \\
 s_{i,j}(t+1) &= 1 \oplus s_{i-1,j}(t) \oplus s_{i,j-1}(t) \oplus s_{i+1,j}(t) \oplus s_{i,j+1}(t) \oplus s_{i,j}(t).
 \end{aligned}$$

The rule-numbering scheme is explained in Section 3.

if the genetic operator of mutation is “successful” in replenishing the genetically depleted population (indeed, our experiments revealed that this latter is usually the case, i.e., evolution manages to bootstrap itself from this low-fitness phase).

We performed a total of 30 experiments, observing that, of the 64 possible rules, there is a subset that tends to emerge via evolution, comprising rules 31, 47, and 63. We also noted that a majority of rules were “1-dominated,” i.e., their genomes (see Section 3) contained a majority of 1s: Considering all experiments performed, 63.3 percent of the rules comprised five or six 1s and 32 percent comprised four 1s. Based on the 1-dominance phenomenon, we then constructed by hand custom CAs containing only rules with five or six 1s (rules 31, 47, 55, 59, 61, 62, and 63) by randomly placing these rules in the grid. The idea of applying the observations emerging from the evolutionary process in order to further improve the results was previously used by us in [10]. Fig. 2 demonstrates the operation of such a handcrafted, 15×15 CA.

All evolved and constructed two-dimensional CA random number generators were subjected to an extensive battery of statistical randomness tests. Randomness can never be proven, only the lack of it can be shown. Although no amount of testing can guarantee that a given sequence is truly random, testing is the

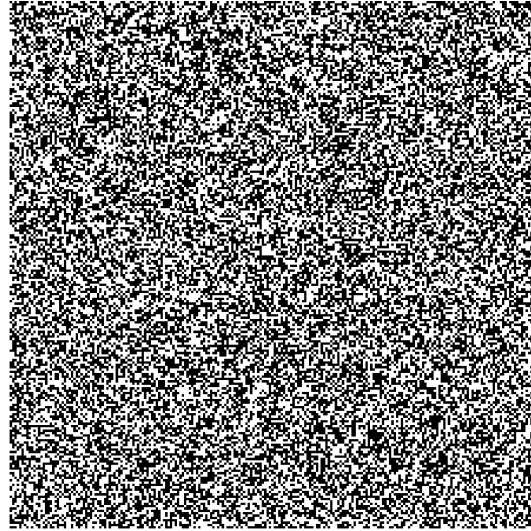


Fig. 2. The operation of a handcrafted CA. In the above image, the 15 lines of the 15×15 grid have been juxtaposed, thus forming one 225-cell line. Each line in the image depicts the configuration of (225) states at a given time, with time increasing down the page. This image allows us to visually observe the random nature of the CA bit sequences (a fact confirmed by the statistical randomness tests).

most acceptable pragmatic way of assessing the pseudorandomness of a given pattern. For testing, we used what is probably the most stringent suite of randomness tests presented to date: Marsaglia’s *Diehard* suite [14]. A detailed description of the tests is beyond the scope of this paper and the interested reader is referred to [10], [14], [15].

How does one extract random numbers from a two-dimensional cellular automaton? There are several methods of doing this, with our own choice based on simplicity: The CA is run for four time steps, each cell thus producing a sequence of four bits (in time)—which are treated collectively as a hexadecimal digit. These hexadecimal digits are then juxtaposed—cell after cell and line after line in a left-right, top-down manner—to create a sequence of $x \times y$ random numbers, where x, y are the grid’s dimensions. The process is then iterated. For example, an 8×8 grid will produce 64 hexadecimal random digits every four time steps.

Table 1 delineates the test results of five evolved CAs. Our main conclusion is that excellent CA random number generators can be

TABLE 1
Test Results of Five Evolved 8×8 CA Random Number Generators

Test name	CA 1	CA 2	CA 3	CA 4	CA 5
Birthday spacing	pass	pass	pass	pass	pass
Overlapping permutation 1	pass	pass	pass	pass	fail
Binary rank 31*31	pass	pass	pass	pass	pass
Binary rank 32*32	pass	pass	pass	pass	pass
Binary rank 6*8	pass	pass	pass	pass	fail
Count the ones	pass	pass	pass	pass	fail
Parking lot	pass	pass	pass	pass	pass
Minimum distance	pass	pass	pass	pass	pass
3D sphere	pass	pass	pass	pass	pass
Squeeze	pass	fail	pass	pass	pass
Overlapping sum	pass	pass	pass	pass	pass
Run up	pass	pass	pass	pass	pass
Run down	pass	pass	pass	pass	pass
Craps	pass	pass	pass	pass	pass

CAs 1, 3, and 4 are among the best ones evolved, passing all the tests. For comparison, we also show two CAs of slightly lesser quality. Fixed boundary conditions are used (Section 2). For a full description of the random-number tests, see [10], [14].

TABLE 2
Test Results of a Handcrafted 8×8 CA Random Number Generator

boundary conditions	cyclic	fixed (full)	fixed (reduced)
Birthday spacing	pass	pass	pass
Overlapping permutation 1	pass	fail	pass
Binary rank 31×31	pass	pass	pass
Binary rank 32×32	pass	pass	pass
Binary rank 6×8	pass	pass	pass
Count the ones	pass	fail	pass
Parking lot	pass	fail	pass
Minimum distance	pass	fail	pass
3D sphere	pass	pass	pass
Squeeze	pass	fail	pass
Overlapping sum	pass	pass	pass
Run up	pass	pass	pass
Run down	pass	pass	pass
Craps	pass	pass	pass

For the CA with fixed boundary conditions, we applied two distinct bit-sampling modes: *full*—in which the cells impinging upon the fixed boundary cells are taken into account and *reduced*—in which these next-to-boundary cells are not taken into account. For a full description of the random-number tests, see [10], [14].

evolved which pass (with flying colors) one of the hardest batteries of randomness tests used today [14].

Table 2 shows the test results of a hand-designed CA, using both cyclic and fixed boundary conditions (see Section 2). For the CA with fixed boundary conditions, we applied two distinct bit-sampling modes: *full*—in which the cells impinging upon the fixed boundary cells are taken into account and *reduced*—in which these next-to-boundary cells are not taken into account. We note that, with cyclic boundary conditions, as well as with fixed boundary conditions in reduced mode, the CA passes all the tests, while,

with fixed boundary conditions in full mode, the CA fails a number of tests; this latter is probably due to the fixity of the outer-layer cells, which—when in full mode—reduces the randomness of the resulting sequence (since the impinging cells have fixed neighbors).

Comparing Tables 1 and 2, we note that both evolved CAs, as well as hand-designed ones (though based on evolutionary results), produce high-quality random sequences. The advantage of hand-designing CAs has to do mainly with hardware implementation: One can choose rules that are more easily implemented in the target medium (e.g. VLSI).

Table 3 shows that our CA passes all the tests, thus surpassing that of Chowdhury et al. [11]. In particular, we performed four additional multivalued tests described by Marsaglia [14], with our CA passing all of them, while that of Chowdhury et al. fails most of them. Our CAs also outperform two of the best known classic RNGs: CGL (an RNG based on the linear congruential method) and RAN3 (an RNG of the lagged Fibonacci type); these two RNGs were studied by us in detail in [10]. Our CAs are thus among the best known RNGs to date (perhaps even the best), with the added advantage of easier amenability to hardware implementation (because of their “CA-ish” nature).

Finally, we also performed experiments with smaller (down to 5×5) and larger (up to 15×15) grids. The smaller CAs are good RNGs for some applications, though our conclusion is that a grid of size at least 7×7 is necessary in order to obtain excellent results.

5 STUDYING CYCLE LENGTHS

A state of the global discrete dynamical system comprised by a CA is the pattern of 0s and 1s at a given time step. One can trace out the behavior of a one-dimensional CA in time by completely describing its trajectory through state space from any given initial configuration. This is portrayed as rows of successive global states

TABLE 3
Comparison of One of Our CAs with the CA Described by Chowdhury et al. [11]

Test name	Our CA	Chowdhury <i>et al.</i> [11]
Birthday spacing	pass	pass
Overlapping permutation 1	pass	fail
Overlapping permutation 2	pass	fail
Binary rank 31×31	pass	pass
Binary rank 32×32	pass	pass
Binary rank 6×8	pass	fail
Count the ones	pass	fail
Parking lot	pass	pass
Minimum distance	pass	pass
3D sphere	pass	pass
Squeeze	pass	fail
Overlapping sum	pass	pass
Run up 1	pass	pass
Run up 2	pass	pass
Run down 1	pass	pass
Run down 2	pass	pass
Craps number of throws	pass	pass
Craps number of wins	pass	pass
Bitstream	20/20	20/20
OPSO	23/23	0/23
OQSO	28/28	0/28
DNA	31/31	0/31

Our CA was created by randomly placing in each cell of an 8×8 grid one of the rules with at least five 1s. Note that, for the evaluation, we performed four additional multivalued tests described in [14] (the results of these additional tests are given in the last four lines; in these tests, the notation k/l means that the CA passes k of the l tests).

TABLE 4
Average and Maximum Cycle Lengths for Various Small Two-Dimensional CAs

No. cells	Avg cycle length	Max cycle length	Max/Avg	$\log_2(\text{Avg cycle})$
9 (3 × 3)	28	512	18.29	4.81
12 (3 × 4)	1135	4096	3.61	10.15
15 (3 × 5)	8123	32768	4.03	12.99
16 (4 × 4)	4778	65536	13.72	12.22
18 (3 × 6)	17072	262144	15.36	14.06
20 (4 × 5)	148173	1048576	7.08	17.18
21 (3 × 7)	393212	2097152	5.33	18.58
24 (4 × 5)	1293922	16777216	12.97	20.30
25 (5 × 5)	2464153	33554432	13.62	21.23

The ratio of maximum to average cycle lengths and the logarithm of the average value are also shown. For each CA size, the average length was computed over 20 randomly generated CAs, each run over 20 random initial configurations.

of the entire array—the so-called space-time pattern. Space-time patterns, which are well-known in the domain of continuous dynamical systems, represent a deterministic sequence of global states advancing along one particular path within a basin of attraction. In a finite array, the path inevitably leads to a state cycle. Other sequences of global states typically exist that lead to the same state cycle. The set of all possible paths make up the basin of attraction. CA basins of attraction are thus composed of global states linked according to their trajectory relationship and will typically have a topology of branching trees rooted at attractor cycles [16]. Other separate basins of attraction typically exist within the set of all possible array configurations. A CA will, in a sense, crystallize state space into a set of basins of attraction, known as the basin of attraction field. The basin of attraction field is a mathematical object which, if represented as a graph, is an explicit global portrait of a CA's entire repertoire of behavior. It includes all possible space-time patterns.

The length of a CA's state cycle is very important in determining the suitability of the CA as a generator of random numbers. In general, and other things being equal, the longer the cycle—the better the CA acts as an RNG. For instance, typical Monte Carlo applications may require on the order of 10^8 pseudorandom numbers. Ideally, an arbitrary n -cell CA RNG should have a maximum cycle length, i.e., it should start repeating itself only after 2^n time steps since this will result in the longest

possible pseudorandom sequence. It is indeed possible to construct a CA with maximum cycle length for a particular class of CAs called *group CAs*. In this case, the cycle length is $2^n - 1$ and the construction is based on the characteristic polynomial of the CA, which should be primitive, as described in [1]. However, according to our results, CAs in this class are not the best in terms of the quality of the random numbers generated (note that totally nonrandom sequences can exhibit maximal cycle length). In a recent work, Cattel et al. [6] showed that maximal length, nonuniform, $2 \times n$ CAs (i.e., two lines of n cells each) can be constructed up to $n = 250$. The bit patterns generated by these CAs are useful in self-test applications.

Since the rules of our nonuniform CA RNGs have been either artificially evolved or drawn from evolved rules (rather than devised on the basis of the characteristic polynomial), it follows that they will not in general exhibit maximum cycle length. Hence, we wanted to study empirically the behavior of the cycle lengths in our automata.

Finding cycle lengths is difficult because the number of possible global CA states increases exponentially with the number of cells; this limits the size of CAs for which cycle lengths can be calculated. We tested a number of randomly generated, small-size, good-quality, two-dimensional CAs over 20 random initial configurations, with the goal being to compare the average observed cycle length with the theoretical maximal length. The results, summar-

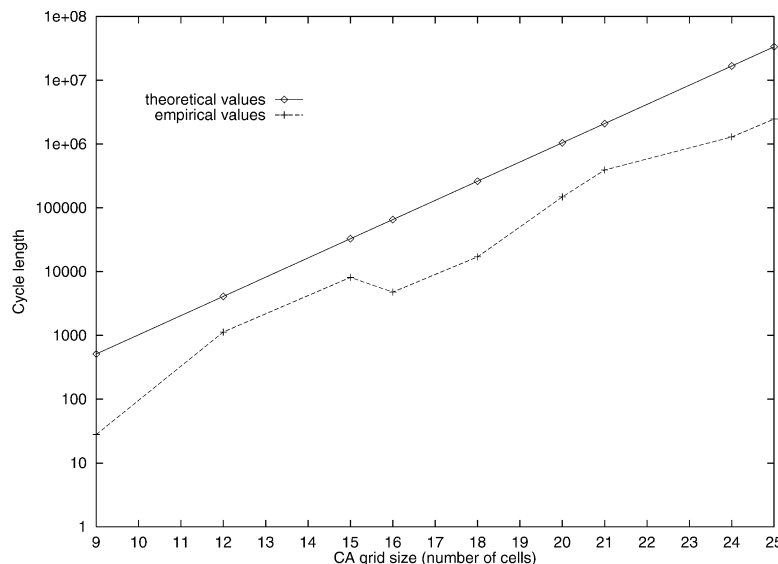


Fig. 3. Average CA cycle lengths (dotted line) compared with the theoretical maximum values (solid line) for various CA grid sizes.

ized in Table 4, suggest that the average cycle length for an n -cell CA increases exponentially and is on the order of 2^{n-3} .

The observed relationship between maximum and average cycle lengths is depicted in Fig. 3. Though we could not test large CAs, our experiments with small CAs never resulted in a CA of cycle length less than 2^{n-4} on average, except for a small deviation (4.81 instead of 5 or higher) in the 9-cell case (3×3). Extrapolating from this observation, we postulate that most cycles of an 8×8 two-dimensional CA will not be shorter than 2^{60} . If extremely long cycles are of crucial import (which might be the case with large-scale statistical physics applications), then one could reseed the generator before the end of the probable average cycle (i.e., initialize the CA with a new initial configuration) or simply use a slightly larger CA.

6 CONCLUDING REMARKS

In this paper, we applied the cellular programming evolutionary algorithm to the hard problem of automatically producing random number generators. Our results show that excellent two-dimensional RNGs can be evolved that outperform previous RNGs. Moreover, based on the rules discovered by evolution, one can handcraft high-quality CAs, possibly tailoring them to meet hardware constraints. Since evolutionary algorithms are stochastic heuristic search techniques, there is no guarantee that their results are optimal. In other words, there is no theory that can assure us that better 2D CAs cannot be constructed. However, judging by the independent comprehensive statistical tests applied, the automata found are of very good quality for pseudorandom number generation.

Comparing the two-dimensional CAs studied herein with previously studied one-dimensional CAs brings to light an important point which has to do with so-called *time spacing*. Time spacing means that not all the bits generated are considered as part of the random sequence. For instance, one might keep only one bit out of two, referred to as a time spacing value of 1, which means that sequences will be generated at half the maximal rate. With one-dimensional CAs, it was found that time spacing is needed in order to produce high-quality random numbers [3], [4], [10]. A major result of this work is that excellent random-number sequences can be produced by our two-dimensional CAs without recourse to time spacing. This is important in that the random numbers can be generated at a much higher rate—without loss of quality. Lack of time spacing may also facilitate hardware implementation.

In conclusion, by applying an extensive suite of randomness tests, we have demonstrated that two-dimensional CAs can be automatically designed to rapidly generate high-quality random-number sequences.

REFERENCES

- [1] P.P. Chaudhuri, D.R. Chowdhury, S. Nandi, and S. Chattopadhyay, *Additive Cellular Automata: Theory and Applications*, vol. 1. Los Alamitos, Calif.: IEEE CS Press, 1997.
- [2] S. Nandi, B.K. Kar, and P.P. Chaudhuri, "Theory and Application of Cellular Automata in Cryptography," *IEEE Trans. Computers*, vol. 43, pp. 1,346-1,357, 1994.
- [3] P.D. Hortensius, R.D. McLeod, W. Pries, D.M. Miller, and H.C. Card, "Cellular Automata-Based Pseudorandom Number Generators for Built-In Self-Test," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 8, pp. 842-859, Aug. 1989.
- [4] P.D. Hortensius, R.D. McLeod, and H.C. Card, "Parallel Random Number Generation for VLSI Systems Using Cellular Automata," *IEEE Trans. Computers*, vol. 38, no. 10, pp. 1,466-1,473 Oct. 1989.
- [5] P. Tsalides, T.A. York, and A. Thanailakis, "Pseudorandom Number Generators for VLSI Systems Based on Linear Cellular Automata," *IEE Proc. E. Computers and Digital Technology*, vol. 138, pp. 241-249, 1991.
- [6] K. Cattel, S. Zhang, M. Serra, and J.C. Muzio, "2-by- n Hybrid Cellular Automata with Regular Configuration: Theory and Application," *IEEE Trans. Computers*, vol. 48, no. 3, pp. 285-295, Mar. 1999.
- [7] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, third ed. Heiderberg: Springer-Verlag, 1996.
- [8] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, Mass.: MIT Press, 1996.
- [9] M. Sipper and M. Tomassini, "Generating Parallel Random Number Generators by Cellular Programming," *Int'l J. Modern Physics C*, vol. 7, no. 2, pp. 181-190, 1996.
- [10] M. Tomassini, M. Sipper, M. Zolla, and M. Perrenoud, "Generating High-Quality Random Numbers in Parallel by Cellular Automata," *Future Generation Computer Systems*, vol. 16, pp. 291-305, 1999.
- [11] D.R. Chowdhury, I.S. Gupta, and P.P. Chaudhuri, "A Class of Two-Dimensional Cellular Automata and Applications in Random Pattern Testing," *J. Electronic Testing: Theory and Applications*, vol. 5, pp. 65-80, 1994.
- [12] M. Sipper, *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*. Heidelberg: Springer-Verlag, 1997.
- [13] M. Sipper, "The Emergence of Cellular Computing," *Computer*, vol. 32, no. 7, pp. 18-26, July 1999.
- [14] G. Marsaglia, "Diehard," <http://stat.fsu.edu/~geo/diehard.html>, 1998.
- [15] D.E. Knuth, *The Art of Computer Programming: Volume 2, Seminumerical Algorithms*, third ed. Reading, Mass.: Addison-Wesley, 1998.
- [16] A. Wuensche and M. Lesser, *The Global Dynamics of Cellular Automata: An Atlas of Basin of Attraction Fields of One-dimensional Cellular Automata*. Santa Fe Inst. for Studies in the Sciences of Complexity, Reading, Mass.: Addison-Wesley, 1992.