An Evolutionary Approach to the Design of Controllable Cellular Automata Structure for Random Number Generation

Sheng-Uei Guan and Shu Zhang

***Abstract*—Cellular automata (CA) has been used in pseudo-random number generation for over a decade. Recent studies show that two-dimensional (2-D) CA pseudorandom number generators (PRNGs) may generate better random sequences than conventional one-dimensional (1-D) CA PRNGs, but they are more complex to implement in hardware than 1-D CA PRNGs. In this paper, we propose a new class of 1-D CA—controllable cellular automata (CCA)—without much deviation from the structural simplicity of conventional 1-D CA. We first give a general definition of CCA and then introduce two types of CCA: CCA0 and CCA2. Our initial study shows that these two CCA PRNGs have better randomness quality than conventional 1-D CA PRNGs, but that their randomness is affected by their structures. To find good CCA0/CCA2 structures for pseudorandom number generation, we evolve them using evolutionary multiobjective optimization techniques. Three different algorithms are presented. One makes use of an aggregation function; the other two are based on the vector-evaluated genetic algorithm. Evolution results show that these three algorithms all perform well. Applying a set of randomness tests on the evolved CCA PRNGs, we demonstrate that their randomness is better than that of 1-D CA PRNGs and can be comparable to that of 2-D CA PRNGs.**

***Index Terms*—Controllable cellular automata, genetic algorithms (GAs), multiobjective optimization.**

NOMENCLATURE

CA	Cellular automata.
CCA	Controllable cellular automata.
EMOO	Evolutionary multiobjective optimization.
GA	Genetic algorithm.
PBCA	Periodic boundary cellular automata.
PCA	Programmable cellular automata.
PRNGs	Pseudorandom number generators.
SCC	Serial correlation coefficient.
VEGA	Vector-evaluated genetic algorithm.

I. INTRODUCTION

CELLULAR automata (CA) were initiated in the early 1950s to explore self-replicating structures. Later in 1986, Wolfram first applied them in pseudorandom number generation. Wolfram's work in [24] proved that the randomness of the patterns generated by maximum-length CA is significantly better than other widely used methods, such as linear feedback shift registers. The intensive interest in this field can be attributed to the phenomenal growth of the VLSI technology that permits cost-effective realization of the simple structure of local-neighborhood CA. CA has become one of the commonly used pseudorandom number generators (PRNGs) [20].

In the last decade, one-dimensional (1-D) CA PRNGs have been studied extensively [9], [11]–[14], [17]–[19], [21]–[23], [25]. Recent interest has been focused more on two-dimensional (2-D) CA PRNGs [3], [15] since their randomness appears better than that of 1-D CA PRNGs. But taking into account the design complexity and computation efficiency, it is quite difficult to conclude which one is better. Compared to 2-D CA PRNGs, 1-D PRNGs are easier to implement on a large scale. In this paper, we propose a novel CA PRNG—controllable cellular automata (CCA) PRNG, which obtains comparable randomness quality as that of 2-D CA PRNGs without losing the structural simplicity of 1-D CA PRNGs.

Based on the observation of the tested CCA PRNGs, we find that the randomness of CCA PRNGs is affected by their structures. To find some CCA structures for pseudorandom number generation, we use evolutionary multiobjective optimization (EMOO) techniques. Three different algorithms based on EMOO are presented. They generate compatible results on the CCA structures evolved, with slight difference in performance. Randomness test results on the evolved CCA PRNGs show that they can generate good randomness quality and the quality remains good for a wide range of initial seeds.

The paper is organized as follows. We first give an overview on CA background and related work in Section II. In Section III, two CCA PRNGs—CCA0 and CCA2—are introduced. Section IV presents three EMOO algorithms. Section V shows the evolution results on CCA structures and the comparison on these three techniques. Section VI compares the evolved CCA PRNGs with 1-D/2-D CA PRNGs. Section VII provides a Conclusion.

II. RELATED WORK

A. CA PNGs

A CA is an array of cells where each cell is in any one of its permissible states. At each discrete time step (clock cycle), the evolution of a cell depends on its transition rule, which is a function of the present states of its k neighbors for a k-neighborhood CA. The cellular array (grid) is n-dimensional, where n =1, 2,

Manuscript received December 4, 2001; revised June 3, 2002 and September 10, 2002. This work was supported by A*Star under Research Grant (ICT/00/013/014).

The authors are with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119260 (e-mail: eleguans@nus.edu.sg; elezs@nus.edu.sg).

Digital Object Identifier 10.1109/TEVC.2002.806856

3 is used in practice. We define the state of a CA at time t to be the n-tuple formed from the states of the individual cells, $x(t) = [x_1(t), \ldots, x_n(t)]$. The next-state function of a 3-neighborhood ($r = 1$) CA is computed as $x(t+1) = f(x(t)) = [f_1(0, x_1(t), x_2(t)), \ldots, f_i(x_{i-1}(t), x_i(t), x_{i+1}(t)), \ldots]$. When each f_i is a linear function, f is also a linear function, mapping n-tuples to n-tuples. The evolution of the ith cell in a 1-D, 3-neighborhood CA can be represented as a function of the present states of the $(i-1)$th, (i)th, and $(i+1)$th cells as: $x_i(t+1) = f_i(x_{i-1}(t), x_i(t), x_{i+1}(t))$, where f_i represents the transition rule for the (i)th cell.

Some definitions to characterize the properties of CA are noted below.

Definition 1: If the rules of a CA cell involve only XOR logic, then it is called a *linear rule*. Rules involving XNOR logic are referred to as *complemented rules*. In this paper, we use a combination of both linear and complemented rules. A CA having a combination of XOR and XNOR ($\sim$XOR) rules is called an *additive* CA [22].

Definition 2: If all the CA cells obey the same rule, then the CA is said to be a *uniform CA*; otherwise, it is a *nonuniform or hybrid CA* [9].

Definition 3: A CA is said to be a *periodic boundary CA* (PBCA) if the extreme cells (the first and last cells) are adjacent to each other. A CA is said to be a *null-boundary* CA if every extreme cell is only connected to its left (right) cell [22].

Generally, there are four aspects affecting the randomness of CA PRNGs: boundary condition, transition rule, length of CA, and initial seed. We use the periodic boundary condition because it is better than the null-boundary condition in random number generation [23]. The choice of transition rules is important for both uniform and nonuniform CAs. Since considerable effort is expended exploring the effect of different rules, we use only those rules that have proven to be good in random number generation. Here, we give the Boolean form of the rules used and their numbers are given in accordance with Wolfram's convention [24]. The following rules are either additive or linear except rule 30.

- Rule 30: $x_i(t+1) = x_{i-1}(t)$ XOR $[x_i(t)$ OR $x_{i+1}(t)]$.
- Rule 90: $x_i(t+1) = x_{i-1}(t)$ XOR $x_{i+1}(t)$.
- Rule 105: $x_i(t+1) = x_i(t)$ XNOR $[x_{i-1}(t)$ XOR $x_{i+1}(t)]$.
- Rule 150: $x_i(t+1) = x_{i-1}(t)$ XOR $x_i(t)$ XOR $x_{i+1}(t)$.
- Rule 165: $x_i(t+1) = x_{i-1}(t)$ XNOR $x_{i+1}(t)$.

The first work applying CA in randomness number generation was done by Wolfram on rule-30 uniform CA in 1986 [19]. His work demonstrated CA's ability to produce highly random, temporal bit sequences [23], [24]. Later, other rules were also applied in uniform CA PRNGs. Tomassini *et al.* concluded in [14] that according to the DIEHARD test results, rule 105 is the best, followed by rules 165, 90, and 150, with rule 30 coming in last.

Hortensius proposed the first nonuniform CA [or programmable CA (PCA)] PRNG using rules 90 and 150 in 1989 [17]. This CA PRNG is referred to hence as PCA 90–150. PCAs that allow different rules to be used on the same cell at different time steps have proven better than CA in random number generation. Fig. 1 shows a programmable cell structure. The rule control signals can be stored in a ROM or generated by a CA. In the latter case, we call it a two-stage PCA. Unlike uniform rule-30 CA, adjacent cells in nonuniform CA are not correlated in both time and space [17]. Hortensius [18] also proposed another PCA PRNG that uses a combination of rules 30 and 45. This generator can evolve to a random pattern of outputs, but its bit sequence correlation is much higher than that of the PCA 90–150 [18].

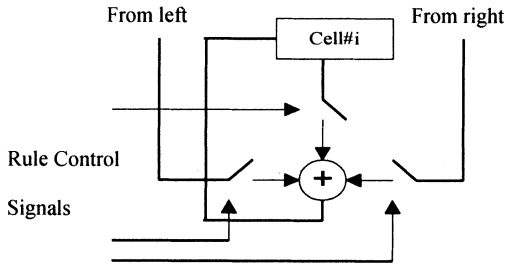


Fig. 1. Programmable cell structure.

Later in 1996, Sipper and Tomassini [14] evolved a 50-cell CA with a mélange of rules 90, 150, and 165. This CA is henceforth referred to as PCA 90–165. Based on their work, Tomassini *et al.* [15] evolved another 50-cell CA with the rule combination 90, 105, 150, and 165 in 1999. This CA is henceforth referred to as PCA 90–105. These two 2-bit PCA were evolved using a cellular programming evolutionary algorithm while those two CA proposed by Hortensius [18] were handcrafted. The DIEHARD test results showed that these two nonuniform CA PRNGs were better than those designed by Hortensius in [17], [18]; however, they still cannot pass some of the tests in DIEHARD and are inferior to the classical generators.

The first work on 2-D CA PRNGs was done by Chowdhury *et al.* [3] in 1994. Their results suggest that 2-D CA are superior to 1-D CA of the same size in pseudorandom number generation. Following their idea, Tomassini *et al.* [15] evolved several 8×8 2-D CA PRNGs with rules 15, 63, 31, and 47. Their DIEHARD test results show that some of the evolved CA PRNGs can pass all the tests. Based on the observation of these evolved 2-D CA PRNGs, they can handcraft even better PRNGs.

Although 2-D CA PRNGs are better than 1-D CA PRNGs in random number generation, they lose the structural simplicity and computation efficiency of 1-D CA PRNGs. Therefore, finding a set of CA PRNGs that can obtain good randomness quality without losing the merits of 1-D CA PRNGs becomes important. Following the idea of PCA, in which a rule control line is added into each cell to improve the randomness of CA PRNGs, we add more control lines on CA cells to control the neighborhood relation and updating of states to further improve the randomness of 1-D CA PRNGs. This results in a new type of CA—a CCA.

B. Introduction to Randomness Tests

Statistical (empirical) tests are used widely to evaluate the randomness of PRNGs. ENT [26] and DIEHARD [8] are the two commonly used test suites. The former is designed

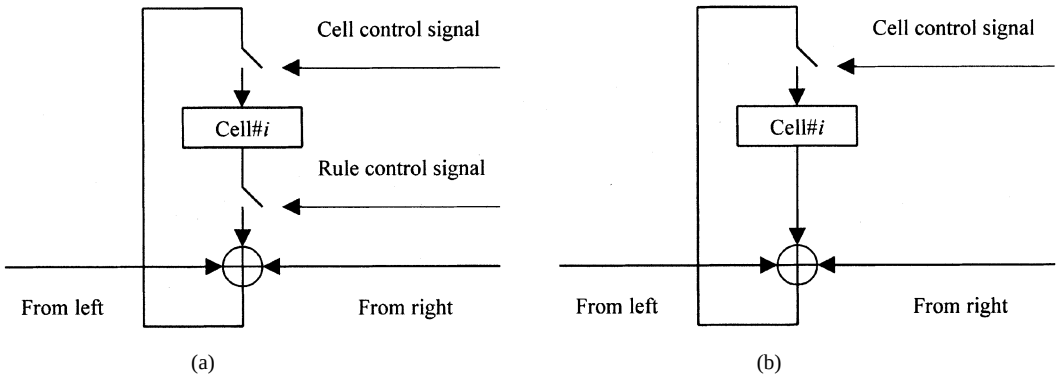


Fig. 2. Controllable cell structure. (a) Programmable controllable cell. (b) Non-programmable controllable cell.

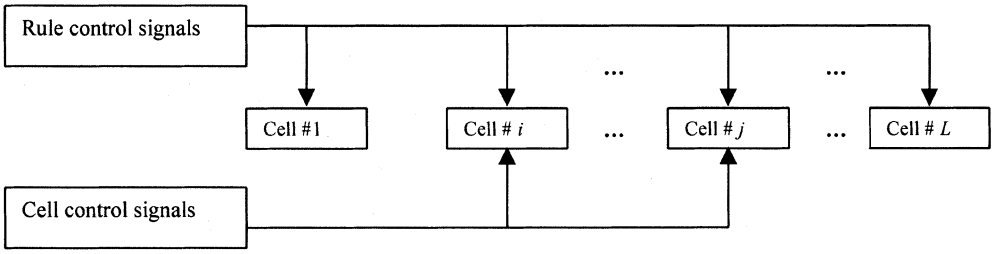


Fig. 3. Structure of a CCA.

according to the criteria set by Knuth [2]; the latter is devised by Marsaglia [8].

III. CCA PRNGs

A. CCA

In this section, the CCA is introduced. To explain the scheme explicitly, several new concepts are defined first to identify the CCA properties.

Definition 4: A CCA is a CA in which the action (how the state of a cell is updated in each cycle) of some cells can be controlled via cell control signals. Similar to rule control signals, cell control signals can be stored in ROM or generated by a CA.

Definition 5: If a cell is under the control of cell control signal, it is a *controllable cell*; otherwise, it is a *basic cell*. CCA is the combination of controllable cells and basic cells. Both controllable cells and basic cells could have rule control signals. Fig. 2 shows the nonprogrammable/programmable controllable cell structure. In this paper, we discuss programmable controllable cells only, henceforth referred to as controllable cells.

The action of a controllable cell is determined by its current cell control signal. A controllable cell can be normal (when the cell control signal is 0) or activated (when the cell control signal is 1). When the controllable cell is normal, the computation of the states of the controllable cell and its neighbors is as usual (according to the current rule control signals and the states of its neighbors). When the controllable cell is activated, the computation of the states of the controllable cell and its neighbors is specified by some predefined actions. The actions applied to the controllable cell and its neighbors could be different. It is observed that the predefined actions affect the state computation of controllable cells.

The structure of a CCA is shown in Fig. 3. It has L cells in total. M ($M <= L$) cells are controllable cells and the remaining $L-M$ cells are basic cells. Here, all the basic cells are programmable cells. Thus, in this CCA, there are L rule control bits and M cell control bits. Compared to an L-cell PCA, which has L rule control bits, the extra cost of CCA is the M cell control bits. During the CA transition, the rule control signals will decide which rule to be employed on both basic and controllable cells; the cell control signals will decide the status of controllable cells. In our work, the rule and cell control signals are generated by two uniform CA separately. All the CCA discussed in this paper are based on this structure. The only difference among them is that they could have different types of controllable cells. The number and location of controllable cells in CCA are called as "setting" of controllable cells in the following.

The usage of controllable cells in a CCA differentiates it from a PCA, in which only rule control signals exist. Once the actions of controllable cells and basic cells are specified, the setting of controllable cells will decide the performance of CCA. The common idea in PCA and CCA is that they both use some control lines on the CA cells to make the CA transition more unpredictable and flexible. The difference is that in PCA, all the cells have uniform structures, while obviously in CCA, the structure of controllable cells are not the same as that of basic cells. To achieve similar CA performance, we may use other methods; for example, increasing the radius (i.e., number of neighbors), using more states in each cell, or evolving the rule tables for each cell as some researchers do in [15], [16]. It is hard to say which method is better in performance or hardware design since their costs are not comparable.

Instead of evolving rule tables, we propose a scheme to control the status of CA cells so that different computation

approaches could be applied when a cell status changes. Based on this, we propose a new class of CA—CCA, capable of changing cell status on the fly. Our work then focuses on finding good configurations of CCA that can obtain good randomness quality. In the following, we introduce two controllable cell types and use them as examples of CCA to further study their performance.

B. Two Types of CCA: CCA0 and CCA2

As we have just introduced, when the cell control signal is zero, a controllable cell acts the same as a basic cell, while when the cell control signal is one, the controllable cell performs some predefined action, which can be different from the action it performs when it is normal. This means that the action a controllable cell performs when it is activated decides the property of controllable cell. The simplest action that an activated controllable cell can take is to keep its state during the CA computation process. In the meantime, the states of its neighbors are computed as usual. This type of controllable cell is called a Type 0 controllable cell. A CCA that is a combination of Type 0 controllable cells and basic cells is henceforth referred to as CCA0.

A Type 2 controllable cell is found when a controllable cell is activated, and it keeps its latest state, while its neighbors will bypass it. This means the activated controllable cell will not be involved in the sate computation of its neighbors. In this way, the neighborhood relationship is changed dynamically during the CA computation process. A CCA that is a combination of Type 2 controllable cells and basic cells is referred to as CCA2 or neighbor-changing CA (NCA). CCA2 cannot be simulated by any PCA due to its neighbor-changing behavior.

C. CCA PRNGs

As we have introduced in Section III-A, the rule control signals and cell control signals for CCA are generated by two CA separately. The CA generating rule (cell) control signals is called as rule (cell) control CA. In each cycle, the bit combination of rule (cell) control signals for CCA cells is termed the rule (cell) control word. The length of rule control word is the same as that of CCA, while the length of cell control word is decided by the number of controllable cells in CCA.

Fig. 4 shows the structure of a CCA PRNG. The running sequence of a CCA PRNG is described as the following. Initial seeds and transition rules are input to the rule/cell control CA and CCA to initialize them. The two control CA run synchronously with CCA to generate rule/cell control words for CCA cells. In each cycle, the previous states of CCA cells plus the rule/cell control words decide CCA cells' current states. The current states of some CCA cells are recorded in every cycle as the output bit sequence. We call these cells output cells. The bit sequence is then converted into an 8-bit random number sequence as the final output. Each CCA runs C cycles to generate the random number sequences. Generally, a long random number sequence is needed to evaluate CCA PRNG's randomness. Considering computation feasibility, we set C to 10 000 cycles.

Because considerable work has been done on the searching of good transition rules in PCA PRNGs [17], [18], we follow the recommended choice of rules here instead of evolving the rules ourselves. The four additive rules used in our work are rules 90, 150, 105, and 30. Rules 90 and 150 are used as the transition rules in CCA; this is to facilitate the comparison with 1-bit PCA 90–150. Rule 30 is used in the rule control CA and rule 105 is used in the cell control CA, since these two rules are said to be among the best ones in random number generation according to Hortensius [17].

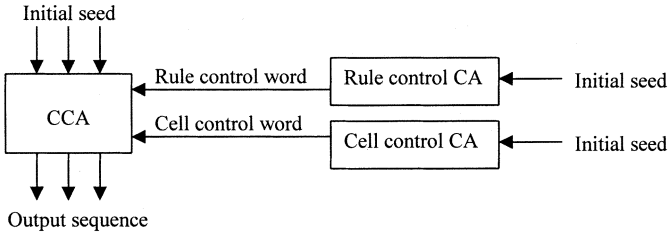


Fig. 4. CCA PRNG structure.

Traditionally, CA PRNGs are handcrafted. The design process is time consuming and troublesome. During the past ten years, researchers began to use evolutionary algorithm to evolve CA PRNGs. Recently, Tomassini *et al.* [14] successfully evolved the rule tables of 2-D CA PRNGs using their cellular programming algorithm. Because our objective is to evolve the setting of controllable cells but not rule tables, cellular programming is not suitable in our work. EMOO is employed here to evolve CCA PRNGs.

IV. EMOO Approaches

A. Objectives

With the introduction of controllable cells in CCA, the structure of CCA is not uniform, as that of conventional CA. The objective of our work is to find good settings of controllable cells in CCA0/CCA2 PRNGs. Moreover, because controllable cells may affect the choice of output cells, we also have to consider how to choose output cells (the CCA cells generating output bits per cycle) other than by using cell spacing, which is a conventional method used in uniform CA and PCA. Thus, we evolve the setting of controllable cells and output cells. By "setting" we mean the number and location of the controllable (output) cells in concern. The chromosome in our evolutionary algorithm is an $L*2$ bitstring. The first L bits identify the controllable cells' setting in which "1" stands for controllable cell, and "0" stands for none-controllable cell. The remaining L bits identify the output cells' setting in which 1 stands for output cell, and 0 stands for nonoutput cell.

In [15], entropy was used as a fitness measure. Randomness tests show that some generators obtain good entropy values but still cannot pass the chi-square test. To get a better evaluation on the randomness of CCA PRNGs, we use chi-square, entropy, and serial correlation coefficient (SCC) tests instead of entropy alone. Furthermore, realizing that the randomness of CCA PRNGs may differ under different initial seeds, we test the CCA PRNGs under a group of randomly generated initial seeds.

To ensure that the results obtained under one group of initial seeds are valid, we do the following test to decide the number of initial seeds to be included in one group. We set the number

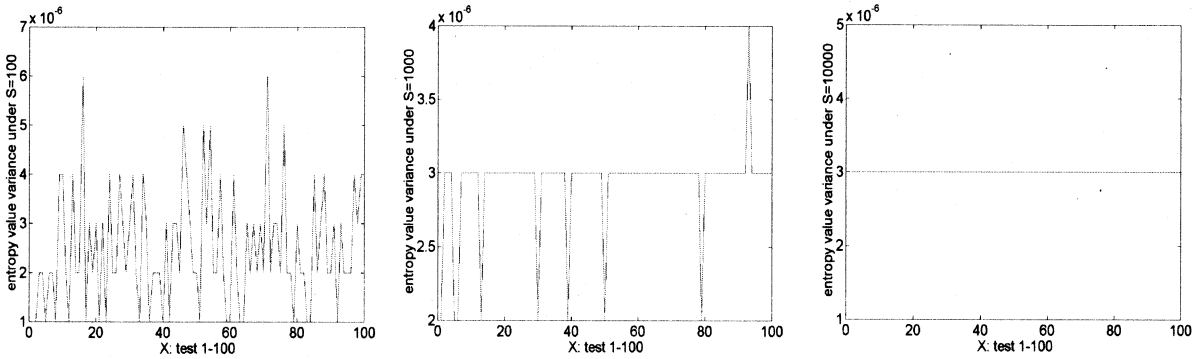
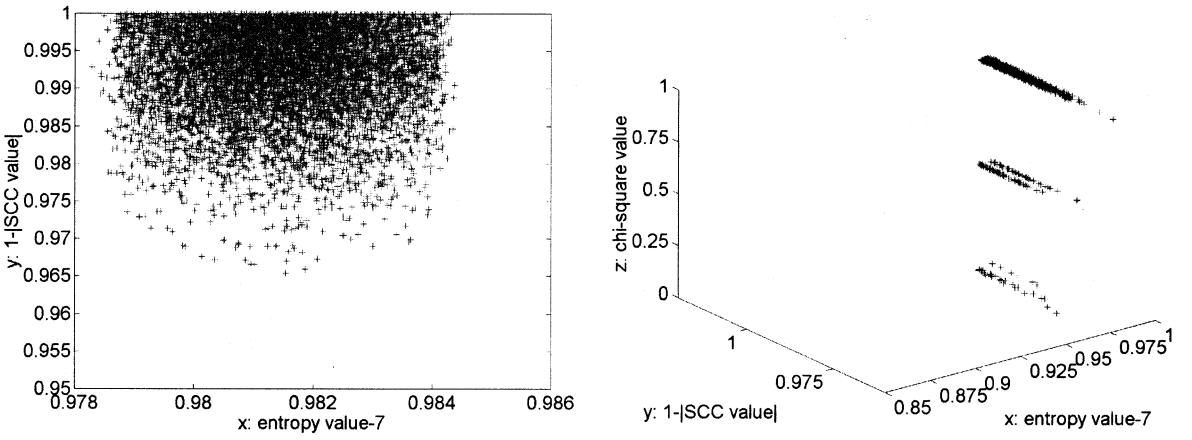


Fig. 5. Entropy variance under different number of initial seeds. Note: Tests 1–100 are independent. S is the number of initial seeds tested.



(a) (b)

Fig. 6. Relationship among chi-square, entropy, and SCC values. (a) Entropy and SCC values (chi-square=1). (b) Chi-square, entropy, and SCC values.

of initial seeds (S) in one group to 100, 1000, and 10 000. The variance of the entropy is used to decide whether or not the performance of CCA PRNGs becomes stable. If the variance gets to a stable value, we conclude that the performance of CCA PRNGs is also stable. Fig. 5 shows the test results. We can see that the entropy variances in all the 100 tests obtain a stable value of 0.000 003 when S is set to 10 000. Using the variance of chi-square or SCC as indication, we get results similar to those presented here.

The randomness of CCA PRNGs is represented by the average value of chi-square, entropy, and SCC tests on 10 000 initial seeds. We use the variance of these values to indicate the performance stability of CCA PRNGs. As a whole, the performance of CCA PRNGs is evaluated using the average value and variance of the chi-square, SCC, and entropy, respectively. Thus, the number of the objectives in our evolutionary approach is six, not just one. Since traditional genetic algorithms (GAs) cannot handle multiobjective optimization effectively, EMOO techniques are introduced.

Various EMOO techniques have been developed [1], [6]. Although surveys and comparative study have been conducted on them [1], [6], none can claim to be the best, since there are still some open questions in this field. Generally, most of the techniques known work in a convex space, while they may have difficulties in a concave space. Before we introduce the EMOO techniques, we first analyze the relationship of the six objectives which to some extent decides whether EMOO is suitable or not.

The chi-square test result is a percentage and any value between 10%–90% means the tested sequence cannot be declared to be nonrandom. We convert the chi-square test result as the following: the chi-square value is 1 if the percentage is between 10%–90%, the chi-square value is 0 if the percentage is greater than 99% or less than 1%; otherwise, the chi-square value is 0.5. The entropy value is between 0 and 8. A larger value means the randomness of the tested sequence is better. Generally, the entropy value is greater than 7. The SCC test result is a real number close to 0, which can be positive or negative. Only the absolute value is meaningful and the sign does not affect the randomness. Generally, absolute SCC values falls in [0, 1] and 0 is the optimal value. We convert the SCC value to $1 - |\text{SCC}|$. Thus, in the adjusted SCC value, 1 is the optimal value and a larger value is better.

Running CCA2 PRNGs under 10 000 randomly configured structures, we get the distribution of chi-square, entropy, and SCC values as presented in Fig. 6. We can see that most CCA2 PRNGs obtain a chi-square value at 1. Fig. 6(a) shows the relation of entropy and SCC values while the chi-square value is 1. Obviously, the search space shown in Fig. 6(a) is convex. And it is evident that the average value and variance are not related because the variance can be high or low no matter the average value is high or low. Thus, the search space is most likely

convex. Because most EMOO techniques work well in a convex search space, we have a wide range of choices here.

Taking computation efficiency into account, we choose the vector-evaluated genetic algorithm (VEGA) as our basic algorithm. VEGA was developed by Schaffer in 1985 [7]. It is the first multiobjective optimization algorithm proposed. The main strength of this technique is its simplicity but it has several problems, such as "middling" as described in Schaffer's paper [7]. To overcome the problems of VEGA, Cvetkovic *et al.* [5] proposed two approaches. One is to wait for a certain number of generations before shuffling the subpopulations together; the other is to avoid shuffling individuals, instead migrating or copying a certain number of individuals from one subpopulation to another. Reference [5] showed that these two methods obtain better results than Schaffer's and can be comparable to the other EMOO techniques. Following the ideas of Schaffer and Cvetkovic, we propose two EMOO approaches here. These two approaches are both based on VEGA, but the evolution algorithm on the whole population is different. In addition to these two approaches, we also use an aggregation function to evolve CCA PRNGs.

B. Algorithm 1 and 2—VEGA-Based Approaches

We develop two EMOO algorithms based on VEGA. One is to copy chromosomes among subpopulations; the other is to use a weighted fitness function to help select individual chromosomes from the whole population into subpopulations. The detail of these two algorithms is presented in Algorithms 1 and 2 individually. The common parameters in these two algorithms are set to the same value.

In both algorithms, the whole population is divided into K ($K = 6$, number of objectives) subpopulations. In each subpopulation, we have P ($P = 20$) chromosomes. P is set to 20 so that the computation time in one generation will not be too long. During the evolution process, the population size is fixed. Each CCA has L ($L = 64$) cells. L is set to 64 because it is a widely used number in both real applications and simulations. To describe the setting of controllable cells and output cells, each chromosome has $2 * L$ bits. Each CCA runs C ($C = 10\,000$) cycles to generate the random number sequence. For each CCA, S ($S = 10\,000$) initial seeds are tested. The subpopulation evolution algorithms are identical in both Algorithms 1 and 2. Each subpopulation evolves t_s ($t_s = 3$) steps before being mixed together or copied. t_s is set to 3 because a smaller value may weaken the effect of subpopulation evolution. And a large value may waste computation time on subpopulation evolution because we find that from the 4th step onward, the best chromosomes' fitness may become stable. The crossover rate is set to 1 and mutation rate r is set to 0.01. RATE is set to 0.5 when selecting parent and child chromosomes into the next generation. Because the population size is small, the selection rate—RATE1 and RATE2 is set to 0.1. Stopping criteria is set to the maximum stagnation steps. If the best chromosome in each subpopulation remains unchanged for T ($T = 200$) steps continuously, the evolution process will stop.

In Algorithm 1, the subpopulations will not be mixed together. To copy some good chromosomes from the current generation of the whole population to the next generation of subpopulation i (i =1–6), we rank the chromosomes in the current generation of each subpopulation j (j =1–6) according to objective i and copy the best $P*$ RATE1 (RATE1 = 0.1, $P*$ RATE1 = 2) chromosomes into the next generation of subpopulation i. It is likely that the chromosomes in subpopulation i (i =1–6) can still generate relatively good results in objective j, since they have been evolved for t_s steps in subpopulation j. The total number of the chromosomes being copied from all the subpopulations into subpopulation i is $K * 2$ ($K = 6$). The remaining ($P - K * 2$) chromosomes in the next generation of subpopulation i are generated by crossbreeding (crossover and mutation between two individuals that are from two different subpopulations). This is based on the assumption that utopian individuals are more likely to result from crossbreeding than inbreeding [7].

The main idea in Algorithm 1 is that we think the two chromosomes copied from the current generation of objective j to the next generation of objective i are likely to be good at both objective j and i after some generations. Thus, those chromosomes that obtain good values in more than one objective will have a greater chance of being maintained during the evolution.

Algorithm 1: VEGA With Elitist Copying

```
//initialization
randomly generate the initial population with
  a fixed size P * K;
randomly divide the whole population into
  K subpopulations (groups), each one has P
  chromosomes;
//evolution
while (stopping criteria is not true) do
   // evolution of subpopulations
   for (m = 1 to K) do in parallel
      for (t = 1 to t_s) do
        //fitness calculation of subpopulation
  m
        calculate each chromosome's fitness
  value according to objective m;
        //crossover & mutation
        • in each subpopulation, roulette-
  wheel select parents, do crossover to gen-
  erate child chromosomes;
        • mutate child chromosomes according
  to mutation ratio r;
           //selection
        • calculate child chromosomes' fit-
  ness;
        • copy the best P* RATE parents into
  the next generation;
        • copy the best P-P* RATE child chro-
  mosomes into the next generation;
      end for (t)
  end for (m)
  // whole population evolution: select and
  copy among subpopulations
  for (i = 1 to K) do
      for (j = 1 to K) do
        • rank the chromosomes in group j ac-
  cording to objective i;
```

```
        • copy the best P* RATE1 chromosomes
  into the next generation of subpopulation i;
      end for (j)
      for (n = P*K* RATE1 to P; n = n+2) do
        • randomly select two chromosomes
  among the whole population, make sure the two
  parents selected are from different subpopu-
  lation (crossbreeding);
        • do crossover and mutation to gen-
  erate 2 child chromosomes;
        • copy the two child chromosomes into
  the next generation of subpopulation i;
      end for (n)
    end for (i)
end while
```

In Algorithm 2, the subpopulations are mixed together after being evolved. Crossover and mutation are performed on the whole population to generate $P * K$ child chromosomes. Different from the selection procedure in the basic VEGA algorithm, we use a weighted fitness function to select the parent and child chromosomes in the next generation. The function F is $\sum$ (the value of objective i) $*$ 1/6 (i = 1 to 6).[1] This function aims at finding those chromosomes that can generate good results in all the objectives. Both the parent and child chromosomes are ranked according to the weighted fitness value and the best R ($R = 3$) chromosomes are copied into each subpopulation i (i =1–6). R is set empirically according to some initial test results. A smaller value of R may degrade the effect of weighted fitness function while a larger value may result in a quick convergence of chromosomes. Moreover, we rank all the parent and child chromosomes according to objective i (i =1–6) and copy the best $P*$ RATE2 (RATE2 = 0.1, $P*$ RATE2 = 2) chromosomes into subpopulation i. After these two steps, there are $P*$ RATE2+R chromosomes in each subpopulation. The remaining chromosomes are selected randomly from the whole population. During the selection, we avoid choosing those that have been selected. This is to avoid rapid convergence in the population. The main idea of Algorithm 2 is that we use a weighted fitness function to ensure that the chromosomes that obtain relatively good values in all objectives are maintained during evolution.

C. Algorithm 3—An Aggregation Function Approach

The simplest way to evaluate multiple objectives in evolutionary algorithm is to combine them into a single weighted-sum function using arithmetical operations. Here, we use a fitness function that is different from the one used in Algorithm 2.

If a sequence cannot pass the chi-square test, it is thought to be unsatisfactory in randomness. That is to say, the chi-square value is an important indication as to the randomness of the sequences tested. Thus, we think that the chi-square test is more important than the entropy and SCC tests in evaluating the randomness of CCA PRNGs. It is difficult to decide which between entropy and SCC is more important because they are testing different aspects of randomness. Taking into account these factors, we assign entropy and SCC the same ratio while giving the chi-square test a slightly higher ratio for emphasis. We use the function $F1$ as follows to evaluate the overall randomness of the CCA PRNGs

$$F1 = (\text{entropy value} - 7) * 0.3 + (1 - |\text{SCC}|) * 0.3 + (\text{chi-square value}) * 0.4. \quad (1)$$

We describe this $F1$ value as the randomness value henceforth. A higher randomness value represents better randomness and the optimal value is 1. For the entropy value, 7 is deducted from the original value and the adjusted value most likely falls within [0, 1]. It is based on our observation from the ENT test results of CCA PRNGs under 10 000 initial seeds. Generally, there is no sequence that earns an entropy value less than 7. To emphasize the difference of the randomness of tested CCA PRNGs, we deduct the common value (7) they obtained from the original test results. The optimal value of the adjusted entropy test result is 1. The larger the adjusted entropy value, the better the randomness. Generally, absolute SCC test values fall into [0, 1]. Contrary to the other two tests in which a better random sequence gets a larger adjusted result, a smaller absolute value gets a better randomness in the SCC test. To adjust an SCC value to the same direction as the other two tests, we deduct its absolute value from 1.

Except randomness, we also consider the performance stability of CCA PRNGs using the variance of chi-square, entropy, and SCC values. We set the same ratio on these three variances as indicated in the following function $F2$

$$F2 = (\text{chi-square value variance} + \text{entropy value variance} + \text{SCC value variance})/3. \quad (2)$$

Taking into account that both good and bad CCA PRNGs can yield small variance, the value of $F1$ is a more important indication than variance in evaluating the performance of CCA PRNGs. We emphasize the randomness in the overall fitness function $F3$

$$F3 = F1 \text{ value } * 0.7 + F2 \text{ value } * 0.3. \quad (3)$$

The ratios in function $F1$ and $F2$ are set empirically. Further study may be done to replace it using fuzzy functions or other techniques, while we think that it is acceptable in our algorithm. In Algorithm 3, the population size is $P*K$ (120). The crossover rate, mutation rate, selection RATE, and stopping criteria are the same as the settings in Algorithms 1 and 2. Algorithm 3 is a standard GA.

V. Results of the Evolutionary Approaches and Discussions

A. Preference

In addition to measurement and searching, decision making is also an important step for multiobjective evolution problems [4]. In Algorithm 3, the decision making process is not necessary (i.e., it is implicit in the search itself). In Algorithms 1 and 2, the evolution results are a group of nondominated chromosomes.

[1]The value of object 1 (entropy) is adjusted as $|\text{entropy} - 7|$.

To decide which chromosome obtains the best randomness, we set the preference as follows: the chi-square value is the most important, the entropy value and the SCC value come next, and the variance of these three statistics is less important.

We first compare the chi-square values of the nondominated chromosomes; the one with the highest chi-square value is regarded as obtaining the best randomness quality. If their chi-square values are identical, then we compare the entropy value; if the entropy values are identical, then we compare the SCC values, and so on. This preference is used to differentiate the performance of the three algorithms, too. The algorithm whose nondominated chromosome obtains the highest chi-square value is believed to achieve the best performance. The preference used is simple; we could use a more complicated method such as a fuzzy function to make the final decision. Looking at the experimental results, we feel the preference chosen is acceptable.

B. Evolved CCA2/CCA0 Structure

The evolution results of CCA2/CCA0 PRNGs are presented in this subsection. The evolution result of CCA2 from Algorithm 1 is the best among the three algorithms according to the preference set, with Algorithms 2 and 3 following. The evolved CCA2 structure (Algorithm 1) is as follows:

$$01\,101\,011\,110\,010\,110\,001\,110\,010\,101\,111\,011\,110\,111\,100\;110\,00100\,010\,000\,001\,111\;\mathbf{35}$$
$$0\,000\,111\,110\,000\,001\,101\,011\,111\,000\,100\,101\,100\,111\,100\,101\;011\,110\,001\,101\,111\,101\;\mathbf{35}\;(\mathbf{20}).$$

In the evolved CCA2 PRNG, there are 35 controllable cells and 35 output cells, in which 20 are controllable cells. The performance of the evolved CCA2 PRNG and the average performance of CCA2 PRNGs before evolution are compared to examine the evolution result. The evolved CCA2 PRNG is tested 1000 times. In each test, the randomness value and variance of chi-square, entropy, and SCC tests are recorded. At the same time, 1000 randomly configured CCA2 PRNGs are tested to compare with the evolved one. We present the comparison on the randomness value in Fig. 7.

We can see the average chi-square value improved (by the evolved one) from 0.75 to 0.945; the entropy value improved from 7.9795 to 7.9816; the SCC value decreased from 0.0096 to 0.0082. As shown in Fig. 7, we can see that the improvement on the entropy and SCC values is not as significant as that of the chi-square value in CCA2 PRNGs. Considering the output efficiency of evolved CCA PRNGs, we find that the evolved chromosomes have a range from 28 to 38 cells as output cells.

Differing with the evolution results of CCA2 PRNGs, the chromosome which obtains the best chi-square value in CCA0 PRNGs is generated by Algorithm 2. Algorithm 1 obtains the second best results and Algorithm 3 comes in last. The evolved CCA0 structure is as follows:

$$1\,101\,100\,100\,100\,011\,100\,101\,010\,111\,110\,101\,100\,000\,011\,011\;001\,111\,111\,100\,101\,100\;\mathbf{35}$$
$$10\,001\,000\,011\,011\,110\,010\,011\,010\,010\,011\,001\,001\,001\,001\;00\,100\,100\,010\,010\,010\,111\;\mathbf{27}\;(\mathbf{12}).$$

The evolved CCA0 has 35 controllable cells too, but it has only 27 output cells, which is less than 35 output cells in CCA2. And among the 27 output cells, only 12 cells are controllable cells. Similarly, we compare the performance of evolved CCA0 PRNGs with the average performance of CCA0 PRNGs before evolution and the results are similar to that of CCA2.

The major difference is that in CCA0, the chi-square value improved greatly from 0.1245 to 0.745 after evolution, and the entropy and SCC value became more stable than those before evolution. Referring to Fig. 7, we can see that the difference of chi-square values in CCA2 PRNGs before and after evolution is only around 0.195. But in CCA0 PRNGs, it is 0.6. It shows that a good setting of controllable cells and output cells is more critical in CCA0 PRNGs to generate good random number sequences. The performance of CCA2 PRNGs is more stable than that of CCA0 PRNGs. Moreover, we find that the evolved CCA2 PRNGs yield better performance than CCA0 PRNGs in general. The average performance of CCA2 PRNGs before evolution is even comparable to the performance of CCA0 PRNGs after evolution. Considering output efficiency, the evolved CCA0 PRNGs have from 19 to 35 cells as output cells. This range is larger than that of CCA2 PRNGs and the output efficiency of CCA0 is noticeably lower. It shows that CCA2 is generally better than CCA0 in random number generation.

C. Comparison on the Performance of the Three Algorithms

We have presented the three algorithms and their evolution results above. Here, we compare their performance and give some recommendation on the choice of algorithms in CCA PRNG evolution. We have pointed out that Algorithms 1 and 2 obtain the best results in CCA2 and CCA0 PNRG evolution individually, and their relative performance is comparable. Although the evolution results of Algorithm 3 in both CCA PRNGs are the worst, its results differ only slightly from the best results. Thus, we can say that Algorithm 3 is also effective. Fig. 8 shows the randomness value of the evolved CCA structure under each algorithm.

In addition to comparing the randomness of the evolved CCA PRNGs, we also need to consider their evolution speed when comparing their performance. The computation time of Algorithms 1 and 2 in one generation is around 4900 and 5300 s (calculated under 1.4-GHz, 256-MB PC), individually, while that of Algorithm 3 is only 1000 s. Our stopping criteria is the maximum stagnation steps. Algorithms 1 and 2 both stop evolution after 1700 evolution steps, while Algorithm 3 stops around 3800 evolution steps. The total computation time of Algorithm 3 is shorter than that of Algorithms 1 and 2.

Realizing that one generation in Algorithms 1 and 2 costs three generations of subpopulation evolution and one generation of the whole population evolution, we can see that the evolution steps of Algorithm 3 actually cost less than those of Algorithms 1 and 2. In conclusion, we can say that both Algorithms 1 and 2 obtain good performance, while Algorithm 3 gets comparable performance. While taking into account the evolution speed and effort, we may claim that Algorithm 3 is also a good choice.

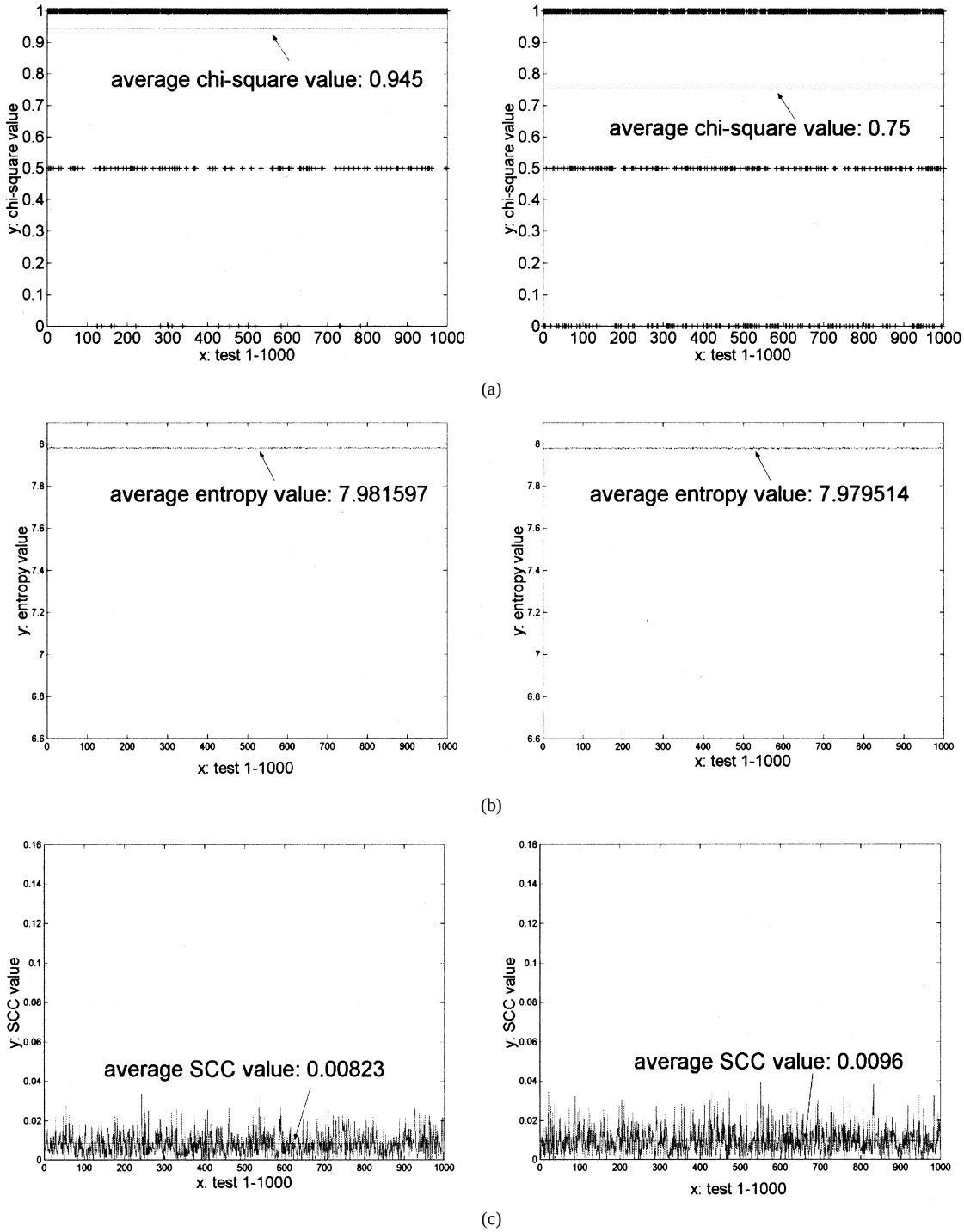


Fig. 7. Comparison of the randomness values between the evolved CCA2 PRNG and the average performance of 1000 randomly configured CCA2 PRNGs. Left column: the evolved CCA2. Right column: the average performance. (a) Chi-square value. (b) Entropy value. (c) SCC value. The straight line across (a) and (c) is the average value in 1000 tests.

D. Analysis of the Evolved CCA0 and CCA2 Structures Under Different Output Methods

In this section, we further study how the settings of controllable cells and output cells affect the performance of CCA PRNGs. At first, we explore the effect of output cells in both CCA2 and CCA0 PRNGs. We compare the performance of the evolved controllable cells' settings under three different output methods: evolved output, cell spacing (cell spacing is set to 1), and randomly configured output. The randomness under the evolved output and the cell spacing method is calculated based on S ($S = 10\,000$) randomly generated initial seeds. The randomness of the randomly configured output method is the average of S ($S = 10\,000$) randomly configured outputs under the evolved controllable cell settings. The tested initial seeds are identical in these three methods. Fig. 9 shows the

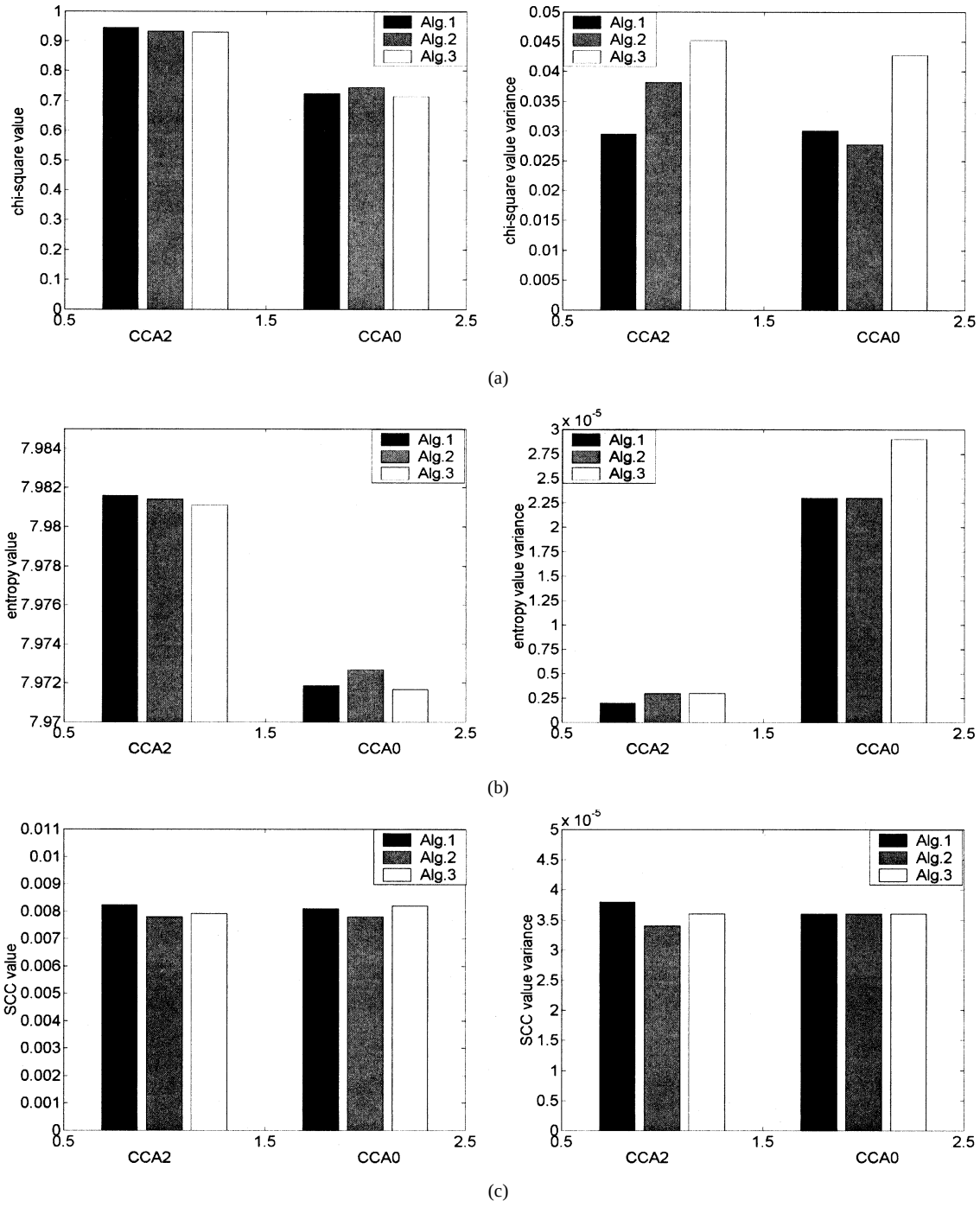


Fig. 8. Comparison on the evolution results of Algorithms 1–3. (a) Chi-square value and variance. (b) Entropy value and variance. (c) SCC value and variance.

randomness values of CCA PRNGs under the tested three output methods.

We can see in Fig. 9 that the evolved output method in CCA2 yields the highest chi-square value, while the other two methods obtain similar results. The results may mean that the setting of output cells is not as important as that of controllable cells in CCA2. That is to say, the performance of CCA2 PRNGs is decided mainly by the setting of controllable cells. Once the controllable cell setting has been evolved, the output cell setting can be flexible if the performance requirement is not strict. Note that the number of output cells in the evolved CCA2 structure is 35, which is more than the 32 used in the conventional cell spacing method. We can say that the generation of random number sequence in the evolved CCA2 structure is more efficient than that in the conventional cell spacing method.

Also shown in Fig. 9, we can see that in CCA0, only the evolved method can generate good randomness value while the other two methods yield peer results. It shows that in CCA0 PRNGs, the setting of output cells is as important as that of controllable cells. To find a good CCA0 PRNG in random number generation, we have to evolve them together. The evolved CCA0 PRNGs has 27 cells (results of Algorithm 2), which is fewer than 32 cells. This means that output efficiency may have to be sacrificed to generate good randomness in CCA0 PRNGs.

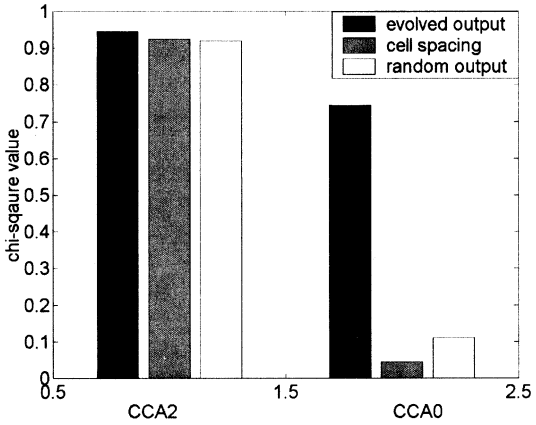


Fig. 9. Comparison on the chi-square values of the evolved controllable cell configuration under different output methods ($S = 10\,000$).

E. Discussions on the Evolved 16-Cell CCA0/CCA2 Structures

The discussion in Section V-B is only concerned with some individual evolved CCA2/CCA0 PRNGs. To find out more about the interrelations between the setting of controllable cells and output cells in a wide range, we conducted the following experiment. The output method is fixed using the conventional cell spacing method (cs = 1) in both CCA PRNGs and we evolve the setting of controllable cells alone. Due to the huge computation effort involved, to simplify the analysis we use Algorithm 3 and evolve 16- instead of 64-cell CA. The setting of controllable cells is encoded as chromsome1 in this evolution approach. The crossover, mutation, and selection rates in this approach are the same as those in Algorithm 3.

Setting the stopping criteria to 100, 200, 300, and 400 evolution steps, we observe the results as shown in Fig. 10. It shows the distribution of evolved chromosome1 in CCA0/CCA2 according to the number of controllable cells included in the 8 output cells. Referring to the results in Fig. 10(a), we can see that most CCA0 PRNGs have zero, one, or two controllable cells within eight output cells. No CCA0 has four or more controllable cells within its output cells. This means a good CCA0 PRNG generally has zero, one, or two controllable cells in its output cells. CCA0 PRNGs with one controllable cell their output cells are most common. It shows that the trend of evolution is to avoid choosing too many controllable cells in the output cells in CCA0.

Referring to Fig. 10(b), we can see that most CCA2 chromosome2s have three to five controllable cells within the eight output cells. This means that the number of basic cells and controllable cells in the output cells are nearly the same. Thus, we can say that the probability of a controllable cell to be chosen as an output cell is similar to that of a basic cell.

Fig. 11 presents the evolution results of CCA0 and CCA2 separately. Referring to Fig. 11(a), we can see that in CCA0 most evolved chromosome1s have 4–8 controllable cells. No chromosome1s have fewer than 2, or more than 10, controllable cells. Only a small number of chrosomosome1s have 9–10 controllable cells. That is to say, when we design CCA0 PRNGs, it is generally better to have 4–8 controllable cells if the total number of cells is 16. Note that all the chromosome1s have at least 2 controllable cells. This means that CCA0 is a better scheme than PCA, which corresponds to a CCA with no controllable cells. Fig. 11(a) also shows that there is a tradeoff for the number of controllable cells in a CCA0. Too many controllable cells may degrade the randomness of CCA0; too few controllable cells may not have the desired "randomizing" effect on the basic cells.

Fig. 11(b) presents the evolution results of CCA2. We can see that in CCA2, evolved chromosomes1s generally have 3–12 controllable cells, which has a wider range than CCA0. In CCA2, most chromosome1s have 5–10 controllable cells; while in CCA0, they generally have 4–7 controllable cells. This shows that in evolved CCA2, the ratio of controllable cells is higher than that in CCA0. Combining the results presented in Figs. 10 and 11, we can see that a suitable number and location of controllable cells is more critical for CCA0 than CCA2 to generate good random numbers. CCA2 PRNGs can get good results under a wider range. This conclusion is compatible with our previous evolution results obtained on 64 cells.

VI. Comparison on the Randomness of Evolved CCA PRNGs Versus 1-D/2-D CA PRNGs

In this section, the randomness of the evolved 64-cell CCA2/CCA0 PRNGs is compared to that of 1-D/2-D CA PRNGs [15], [17].

First, we present the average chi-square, entropy, and SCC values of the evolved CCA2/CCA0 PRNGs and 1-bit/2-bit PRNGs in Table I. The evolved CCA2 PRNG obtains the highest chi-square value with 2-bit PCA and the evolved CCA0 PRNGs following it. These three generators obtain similar entropy and SCC values. The 1-bit PCA PRNG gets the lowest chi-square, entropy, and SCC values. The randomness of the evolved CCA2/CCA0 PRNGs are highly improved compared to that of 1-bit PCA PRNG, and the evolved CCA2 PRNG outperforms the 2-bit PCA 90–105 PRNG.

Until now, CCA PRNGs were evaluated using the ENT test suite only. During the evolution process, the randomness of CCA PRNGs is evaluated using the ENT test suite. To validate our evolution results, the randomness of the evolved CCA PRNGs is examined from three aspects: DIEHARD, cycle length, and time–space diagram. Furthermore, their randomness is compared to those well-known 1-D/2-D CA PRNGs [15], [17].

A. DIEHARD Test Results

The DIEHARD test suite is said to currently be the most difficult test suite to pass. Table II presents the DIEHARD test results of the evolved CCA0/CCA2 PRNGs, 1-bit PCA 90–150 PRNG, 2-bit PCA 90–105 PRNG, and 2-D 8×8 CA PRNG. The tested sequence length is 10 Mbytes. The results show that except for 1-bit PCA, which can pass only 15 tests, the other three PRNGs can pass all the tests. Thus, the evolved CCA PRNG is comparable to the well-known 2-bit PCA and 2-D CA PRNGs. Furthermore, we apply DIEHARD to all the nondominated chromosomes obtained from the three algorithms. We find that each of them can pass all the tests in DIEHARD.

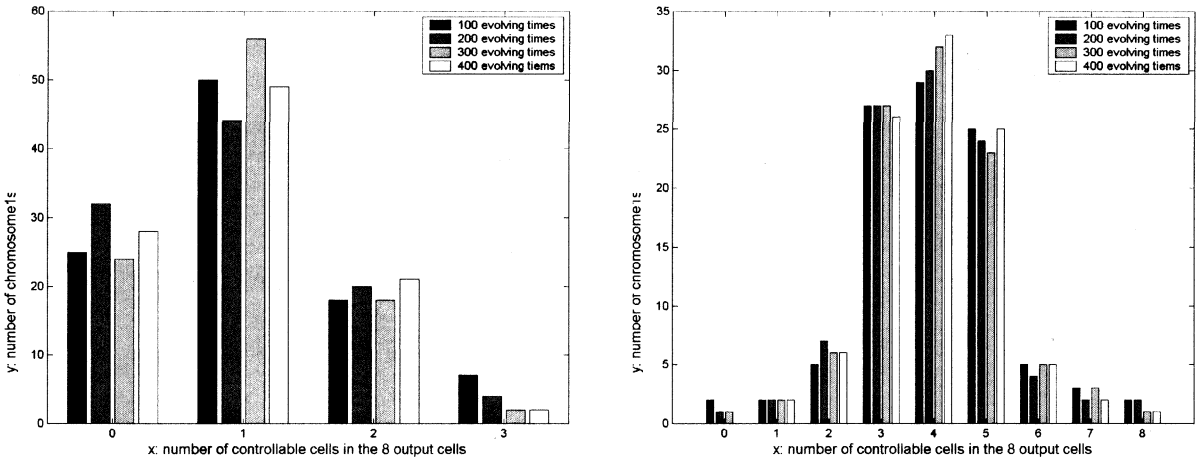


Fig. 10. Distribution of evolved chromosome1s with different number of controllable cells in the output cells. (a) Evolution results of CCA0. (b) Evolution results of CCA2.

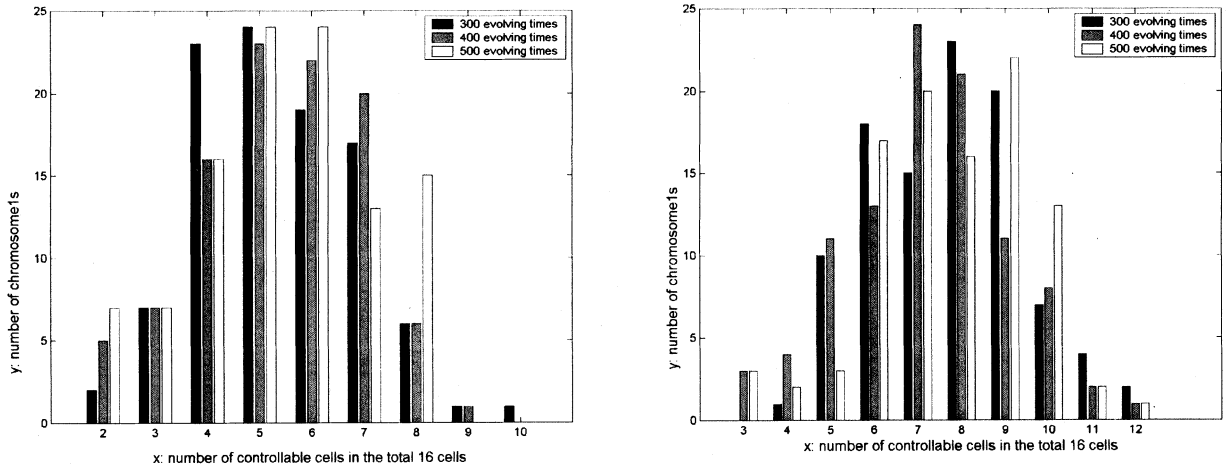


Fig. 11. Distribution of evolved chromosome1s with different number of controllable cells. (a) Results of CCA0. (b) Results of CCA2.

TABLE I
AVERAGE CHI-SQUARE, ENTROPY, AND SCC VALUES OF PCA/CCA PRNGS ($C = 10\,000$, $S = 10\,000$)

	chi-square	entropy	SCC
1-bit PCA 90-150 PRNG	0.450	7.301210	0.0091479
2-bit PCA 90-105 PRNG	0.785	7.981210	0.007932
evolved CCA2 PRNG (Alg.2)	0.945	7.981707	0.007965
evolved CCA0 PRNG (Alg.1)	0.745	7.972676	0.007807

B. Cycle Length

In addition to statistical tests, cycle length (the length of a CA's state cycle) is also important to determine whether or not a CA is suitable for random number generation. We do not consider it one of the objectives when evolving CCA PRNGs because the calculation of cycle length is too time consuming and the value may not be stable under different initial seeds. To verify our evolution results, we calculate the cycle lengths of evolved CCA2/CCA0 PRNGs. Fig. 12 shows the cycle lengths of 1-bit PCA 90–150, 2-bit PCA 90–105, evolved CCA0/CCA2 and 2-D CA PRNGs. The cycle lengths are calculated as average values over 20 random initial seeds.

The results show that the cycle length of PCA 90–150 is the smallest. The cycle length of CCA0 is greater than 2-bit PCA, but less than CCA2. This matches with the conclusion we have derived from the ENT tests that CCA2 is better than CCA0 in random number generation. The average cycle length of 2-D CA is greater than CCA0 but less than CCA2. It means that CCA PRNGs can be better than or comparable to 2-D CA PRNGs. The evolved three CCA0 PRNGs get closer cycle lengths and it is the same case in CCA2. We can say that the randomness

TABLE II
DIEHARD TEST RESULTS OF CCA/CA PRNGS

Test name	CCA0 L=64			CCA2 L=64			PCA 90-150 L=64	8×8 2-d CA	2-bit PCA L=64
	Alg1	Alg2	Alg3	Alg1	Alg2	Alg3			
1. Overlapping sum	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
2. Runs up 1	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
Runs Down 1	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
Runs up 2	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
Runs Down 2	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
3. 3D sphere	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
4. A parking lot	Pass	Pass	Pass	Pass	Pass	Pass	Fail	Pass	Pass
5. Birthday Spacing	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
6. Count the ones 1	Pass	Pass	Pass	Pass	Pass	Pass	Fail	Pass	Pass
7. Binary Rank 6*8	Pass	Pass	Pass	Pass	Pass	Pass	Fail	Pass	Pass
8. Binary Rank 31*31	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
9. Binary Rank 32*32	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
10. Count the ones 2	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
11. Bitstream test	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
12. Craps wins	Pass	Pass	Pass	Pass	Pass	Pass	Fail	Pass	Pass
throws	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
13. Minimum distance	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
14. Overlapping Permu	Pass	Pass	Pass	Pass	Pass	Pass	Fail	Pass	Pass
15. Squeeze	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
16. OPSO test	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
17. OQSO test	Pass	Pass	Pass	Pass	Pass	Pass	Fail	Pass	Pass
18. DNA test	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
Number of tests passed	***18***	***18***	***18***	***18***	***18***	***18***	***12***	***18***	***18***

[3]In each algorithm, the chromosome that obtains the best chi-square value is tested here.

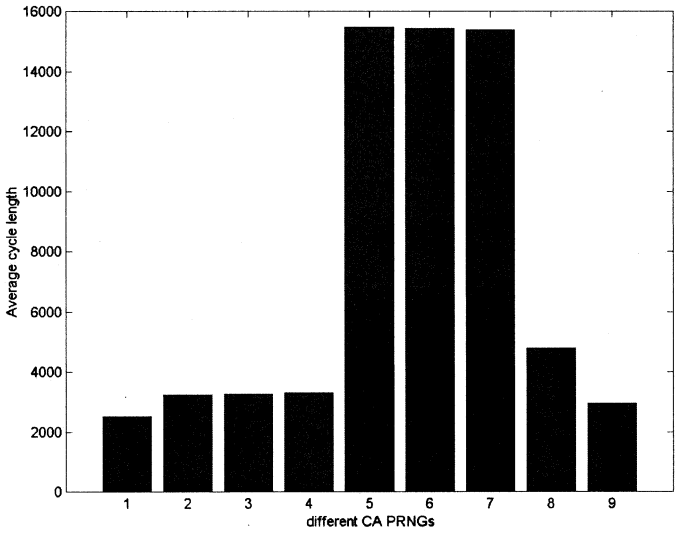
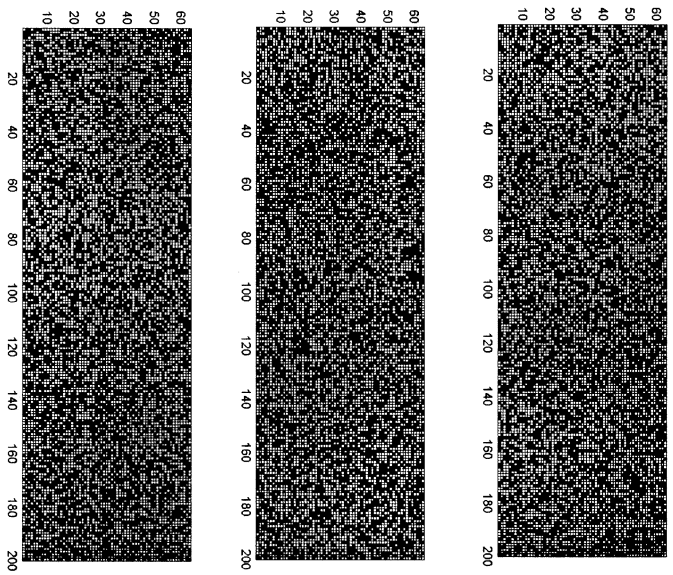


Fig. 12. Average cycle lengths of CA PRNSs. Note: All the tested 1-D CA PRNGs have 16 cells. 1: PCA90–150 PRNG. 2–4: evolved CCA0 PRNGs. 5–7: evolved CCA2 PRNGs. 8: 2-D 4 × 4 CA PRNG. 9: 2-bit PCA90–105 PRNG. Results are based on 20 initial seeds.



(a) (b) (c)

Fig. 13. Time–space diagram of evolved CCA2/CCA0 and 2-bit PCA. (a) CCA2. (b) CCA0. (c) 2-bit PCA 90–105.

of evolved CCA PRNGs is stable. It will not vary greatly under different evolved structures.

C. Time–Space Diagram

We present the 2-D time–space diagram of the evolved CCA2/CCA0 PRNGs and 2-bit PCA PRNG in Fig. 13. Each generator has 64 cells and runs for 200 time steps. The x-axis stands for cells from cell 1 to 64, while the y-axis traces each cell from time step 0 to 200 (from top to bottom). We can see that none of the generators have obvious patterns in their diagrams. Tomassini *et al.* presented the time–space diagram of their 2-D CA PRNGs in [15]. The diagram shows that there is no obvious pattern in 2-D CA PRNGs. From this point of view, we can say

that the evolved CCA2/CCA0 PRNGs are as good as 2-D CA PRNGs or 2-bit PCA PRNGs.

VII. Conclusions

Controllable CA is proposed in this paper and two types of CCA—CCA0 and CCA2 are introduced. CCA0/CCA2 PRNGs are evolved using the EMOO techniques. Two EMOO algorithms based on VEGA and an evolutionary algorithm using an aggregation function are described. Each of them can produce good CCA PRNGs while the performance of Algorithms 1 and 2 is slightly better than that of Algorithm 3. The evolution results show that EMOO helps improve the randomness of CCA PRNGs by evolving the setting of controllable cells and output cells in them.

The evolved CCA2 PRNGs not only obtain good randomness quality, but also generate a better output efficiency than the conventional cell spacing method. Comparison of the evolved CCA2/CCA0 PRNGs shows that CCA2 is better than CCA0 in random number generation. The setting of controllable cells and output cells is more critical for CCA0 than CCA2 in generating good random number sequences. Randomness test results on the evolved CCA0/CCA2 PRNGs show they can be comparable to the well-known 2-bit PCA and 2-D CA PRNGs. Moreover, the cycle length and time–space diagram of these PRNGs are presented. In our current work, we only consider average randomness value and variance as objectives when evolving CCA structures. Further, we may take into account other parameters together as evolution objectives. For example, output efficiency and cycle length are also important to evaluate the performance of CA PRNGs and they can be considered as two objectives in addition to the current six objectives used.

References

[1] C. A. Coello Coello, "An updated survey of evolutionary multiobjective optimization techniques: State of the art and future trends," in *Proceedings of the 1999 Congress on Evolutionary Computation*. Piscataway, NJ: IEEE Press, 1999, vol. 1, pp. 3–13.

[2] D. E. Knuth, *The Art of Computer Programming*, 3rd ed. Reading, MA: Addison-Wesley, 1998, vol. 2, Seminumerical Algorithms.

[3] D. R. Chowdhury, I. S. Gupta, and P. P. Chaudhuri, "A class of two-dimensional cellular automata and applications in random pattern testing," *J. Elect. Testing: Theory and Applic.*, vol. 5, pp. 65–80, 1994.

[4] D. Cvetkovic and I. Parmee, "Preference and application in evolutionary multiobjective optimization," *IEEE Trans. Evol. Comput.*, vol. 6, pp. 42–57, Feb. 2002.

[5] D. Cvetkovic, I. Parmee, and E. Webb, "Multiobjective optimization and preliminary airframe design," in *The Integration of Evolutionary and Adaptive Computing Technologies With Product/System Design and Realization*, I. Parmee, Ed. Berlin, Germany: Springer-Verlag, Apr. 1998, pp. 255–267.

[6] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms—A comparative case study," in *Parallel Problem Solving From Nature V*, A. E. Eiben, Ed. Amsterdam, The Netherlands: Springer-Verlag, Sept. 1998, pp. 292–301.

[7] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in *Genetic Algorithms and Their Applications: Proceedings of the First International Conference on Genetic Algorithms*. Hillsdale, NJ, 1985, pp. 93–100.

[8] G. Marsaglia. (1998) Diehard. [Online]. Available: http://stat.fsu.edu/~geo/diehard.html

[9] I. Kokolakis, I. Andreadis, and Ph. Tsalids, "Comparison between cellular automata and linear feedback shift registers based pseudo-random number generators," *Microprocessors and Microsyst.*, vol. 20, pp. 643–658, 1997.

[10] J. E. Baker, "Adaptive selection methods for genetic algorithms," in *Proc. Int. Conf. Genetic Algorithms and Their Applications*, Hillsdale, NJ, 1985, pp. 101–111.

[11] M. Matsumoto, "Simple cellular automata as pseudorandom m-sequence generators for built-in self-test," *ACM Trans. Modeling and Comput. Simul.*, vol. 8, no. 1, pp. 31–42, 1998.

[12] M. Mihaljevic, "Security examination of a cellular automata based pseudorandom bit generator using an algebraic replica approach," in *Proc. Applied Algebra, Algorithms and Error Correcting Codes*, 1997, vol. 1255, Lecture notes in Computer Science, pp. 250–262.

[13] M. Mihaljevic and H. Imai, "A family of fast keystream generators based on programmable linear cellular automata over GF(q) and time-variant table," *IEICE Trans. Fundamentals*, vol. E82-A, no. 1, pp. 32–39, 1999.

[14] M. Tomassini, M. Sipper, M. Zolla, and M. Perrenoud, "Generating high-quality random numbers in parallel by cellular automata," *Future Gen. Comput. Syst.*, vol. 16, pp. 291–305, 1999.

[15] M. Tomassini, M. Sipper, and M. Perrenoud, "On the generation of high-quality random numbers by two-dimensional cellular automata," *IEEE Trans. Comput.*, vol. 49, pp. 1146–1151, 2000.

[16] M. Sipper and M. Tomassini, "Generating parallel random number generators by cellular programming," *Int. J. Modern Phys.*, vol. 7, no. 2, pp. 181–190, 1996.

[17] P. D. Hortensius, R. D. Mcleod, and H. C. Card, "Parallel random number generation for VLSI system using cellular automata," *IEEE Trans. Comput.*, vol. 38, pp. 1466–1473, 1989.

[18] 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, pp. 842–859, 1989.

[19] P. P. Chaudhuri, D. R. Chowdhury, S. Nandi, and S. Chattopadhyay, *Additive Cellular Automata: Theory and Applications*. Los Alamitos, CA: IEEE CS Press, 1997, vol. 1.

[20] P. Sarkar, "A brief history of cellular automata," *ACM Comput. Surveys*, vol. 32, no. 1, pp. 80–107, 2000.

[21] P. H. Bardell, "Analysis of cellular automata used as pseudorandom pattern generators," in *Proc. Int. Test Conf.*, 1990, pp. 762–768.

[22] S. Nandi, B. K. Kar, and P. P. Chaudhuri, "Theory and applications of cellular automata in cryptography," *IEEE Trans. Comput.*, vol. 43, pp. 1346–1357, 1994.

[23] S. Wolfram, "Cryptography with cellular automata," in *Proc. CRTPTO 85—Advances in Cryptography*, vol. 218, Lecture Notes in Computer Science, 1985, pp. 429–432.

[24] ——, *Theory and Applications of Cellular Automata: Including Selected Papers 1983–1986*. River Edge, NJ: World Scientific, 1986.

[25] W. Pries, A. Thanailakis, and H. C. Card, "Group properties of cellular automata and VLSI applications," *IEEE Trans. Comput.*, vol. C-35, pp. 1013–1024, 1986.

[26] (1998) ENT test suite. [Online]. Available: http://www.fourmilab.ch/random.

Sheng-Uei Guan received the M.Sc. and Ph.D. degrees from the University of North Carolina at Chapel Hill.

He is currently with the Electrical and Computer Engineering Department at National University of Singapore. Previously, he was with Computer Communications Research Labs, ITRI, Taiwan, R.O.C., an R&D organization, serving as a Design Engineer, Project Leader, and Manager. He has also served as a member on the R.O.C. Information and Communication National Standard Draft Committee. After leaving the industry, he joined Yuan-Ze University in Taiwan for three and half years. He served as deputy director for the Computing Center, and also as the chairman for the Department of Information and Communication Technology. Later he joined La Trobe University with the Department of Computer Science and Computer Engineering where he helped to create a new multimedia systems stream.

Shu Zhang is a research engineer in the Department of Electrical and Computer Engineering at National University of Singapore. She received B.Sc. in computer science from Huazhong University of Science and Technology, and M.Eng. from National University of Singapore. Her current research interests include cellular automata, evolutionary computation, and artificial intelligence.