Incremental Evolution of Cellular Automata for Random Number Generation

Sheng-Uei Guan and Shu Zhang

Department of Electrical & Computer Engineering
National University of Singapore
10 Kent Ridge Crescents, Singapore 119260
{eleguans, engp9594}@nus.edu.sg

Abstract Cellular automata (CA) have been used in pseudorandom number generation for over a decade. Recent studies show that controllable CA (CCA) can generate better random sequences than conventional one-dimensional (1-d) CA and compete with two-dimensional (2-d) CA. Yet the structural complexity of CCA is higher than that of 1-d PCA. It would be good if CCA can attain good randomness quality with the least structural complexity. In this paper, we evolve PCA/CCA to their lowest complexity level using genetic algorithms (GAs). Meanwhile, the randomness quality and output efficiency of PCA/CCA are also evolved. The evolution process involves two algorithms — a multi-objective genetic algorithm (MOGA) and an algorithm for incremental evolution. A set of PCA/CCA are evolved and compared in randomness, complexity, and efficiency. The results show that without any spacing, CCA could generate good random number sequences that could pass DIEHARD. And, to obtain the same randomness quality, the structural complexity of CCA is not higher than that of 1-d CA. Furthermore, the methodology developed could be used to evolve other CA or serve as a yardstick to compare different types of CA.

Key words: controllable cellular automata, pseudorandom number generation, multi-objective optimization, incremental evolution

1. Introduction

The first work to apply cellular automata (CA) in pseudorandom number generations was done by Wolfram in 1986 when he studied the randomness of a uniform rule-30 CA. Since then, CA pseudorandom number generators (PRNGs) have been an active research field. Wolfram's work [20] has shown that the randomness of patterns generated by maximum-length CA was significantly better than other widely used methods, such as linear feedback shift registers (LFSR).

In the past 10 years, one-dimensional (1-d) CA PRNGs were studied extensively [1, 11-14, 16-19, 21, 22]. However, the randomness quality of 1-d PCA is unsatisfactory since they still fail some randomness tests. To further improve the randomness of CA, some researchers began to employ two-dimensional (2-d) CA in pseudorandom number generation. Tomassini et al. [15] evolved a 2-d CA that could pass the DIEHARD test [8], which is said to be the most difficult test suite to pass currently. Based on their work, 2-d CA appears to be superior to 1-d PCA in pseudorandom number generation.

Another possibility to improve the randomness of CA is to enhance 1-d PCA by adding cell control signals on some cells. Some work was done in [21]. In that work, the idea of controlling the status of CA cells has been proposed. In a later work [22], this idea was further refined to the concept of controllable CA (CCA). The randomness test results on the proposed CCA showed that CCA are better than traditional 1-d CA and are comparable to 2-d CA in randomness. In [22], CCA are handcrafted by studying the properties of controllable cells. Compared to 1-bit PCA with the same length, the structural complexity of handcrafted CCA is higher because of the usage of cell control signals in controllable cells. On the other hand, the output efficiency of CCA is lower than that of 2-d CA that needs no spacing at all.

In the present work, we extend upon our previous work to minimize the structural complexity and maximize the output efficiency of CCA while maintaining their randomness quality. "Structural complexity" will be elaborated in section 3.1. Our objective is to find whether or not CCA can outperform other 1-d CA in randomness while maintaining simplicity and efficiency. The evolution process involved two algorithms. Fixed-length CCA are evolved using a multi-objective genetic algorithm (MOGA). An incremental evolutionary algorithm is also applied to search for the minimal length of CCA to attain a pre-specified target. Here, the target is to pass all the tests in DIEHARD.

Section 2 gives an overview on 1-d/2-d CA PRNGs in the literature and introduces CCA PRNGs. Section 3 describes the evolutionary algorithms — MOGA and an algorithm for incremental evolution. Section 4 delineates the evolution results, showing that CCA could generate good random number sequences without any spacing while using only a few controllable cells. Section 5 provides a conclusion.

2. Cellular Automata PRNGs

2.1 Previous Work on CA PRNGs

Cellular automata (CA) were originally proposed by von Neumann in the early 1950s to explore self-replicating structures [23]. The increasing interests in CA PRNGs may be due to their simple and cascade structures. CA are regular, locally interconnected, and modular. These characteristics make CA easier to implement in hardware than other models. CA can generate random sequences either sequentially or in parallel. In practice, most CA produce sequences in parallel to obtain higher output efficiency.

A cellular automaton is an array of cells where each cell can be in any one of its permissible states. At each discrete time step (clock cycle) the change of a cell's state depends on its transition rule, which is a function of the present states of its *k* neighbors for a *k*-neighborhood CA. A cellular array (grid) is *n*-dimensional, where *n*=1,2,3 is used in practice. A CA having a combination of XOR and XNOR (~XOR) rules is called an *additive* CA [20]. If all the CA cells obey the same rule, then the CA is said to be *uniform*; otherwise, it is *nonuniform or hybrid* [17]. 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 the extreme cells are connected only to its left (right) cell [20].

The first CA used in pseudorandom number generation is a uniform rule-30 CA. The random sequences were generated by the central cell of CA-30 in consecutive steps. Wolfram has shown in [20] that the random sequences generated by rule-30 CA could obtain fairly good randomness. Later, rule-45 CA has been investigated and compared to rule-30 CA. Conclusively, rule-30 CA has better randomness properties than rule-45 CA but rule-45 CA generally has much larger cycle length for arbitrary starting states [18]. In both CA, cell spacing (cs) and time spacing (ts) were used to form better random sequences. Here, time spacing (e.g. ts=1) means that not all the bits generated are considered as part of the random sequence. For example, a time spacing value of 1 means that sequences will be generated at half the maximum rate. Cell spacing (e.g. cs=3) means only certain cells in a row are considered in generating output, where an integer number (3) indicates how many cells are to be ignored between two successive cells. On the whole, uniform CA without time spacing could comprise fairly good generators, but they could not compare well with standard classical PRNGs like congruential and lagged-fibonacci random number generators [14].

Following the idea of uniform CA, Hortensius [18] studied rule 90-150 programmable CA (PCA) and rule 30-45 PCA. Their study showed that rule 90-150 PCA has better potential than rule 30-45 PCA in pseudorandom number generation. These two PCA are 1-bit PCA where the rule control signal for each programmable cell is 1-bit. Later in 1996, Sipper and Tomassini [16] evolved a 2-bit 50-cell PCA with a mélange of rule 90, 150, and 165, where the rule control signal for each programmable cell is 2-bit. Also, Tomassini et al. [14] evolved another 2-bit 50-cell PCA with the rule combination 90, 105, 150, and 165 in 1999. These two 2-bit PCA were evolved using a cellular programming evolutionary algorithm while the two 1-bit PCA proposed by Hortensius were handcrafted. The DIEHARD test results showed that although 2-bit PCA are better than 1-bit PCA in randomness, they still fail to pass all the tests in DIEHARD with a time spacing of 1. The randomness of 2-bit PCA with the rule combination 90, 105,150, and 165 will be compared with CCA/1-bit PCA in section 4.

Tomassini et al. [15] studied extensively the setting of time spacing parameters in CA. They tested both uniform CA and nonuniform CA under a time spacing parameter of 5, 2, 1, and 0. The results showed that time spacing is critical to generate high-quality random number sequences. Only those PCA with a time spacing parameter greater than 1 could pass all the tests in DIEHARD. Considering output efficiency, they recommended a time spacing of 2 for practical use.

The first work on 2-d CA was done by Chowdhury et al. [6] in 1994. Their results suggested that 2-d CA are superior to 1-d ones with the same size in pseudorandom number generation. Following their idea, Tomassini et al. [15] evolved several 8×8 2-d CA with rule 15, 63, 31, and 47. Their DIEHARD test results showed that some of the evolved 2-d CA could pass all the tests in DIEHARD. Different from 1-d CA, spacing was not used in 2-d CA. Obviously, it resulted in higher output efficiency.

2.2 Controllable Cellular Automata PRNGs

To further improve the randomness quality of nonuniform CA, we proposed controllable cellular automata (CCA) in [22]. A *controllable CA* 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. If a cell is under control via some 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. Here, we discuss programmable controllable cells only. Thus, programmable controllable cell is referred to henceforth as controllable cell.

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

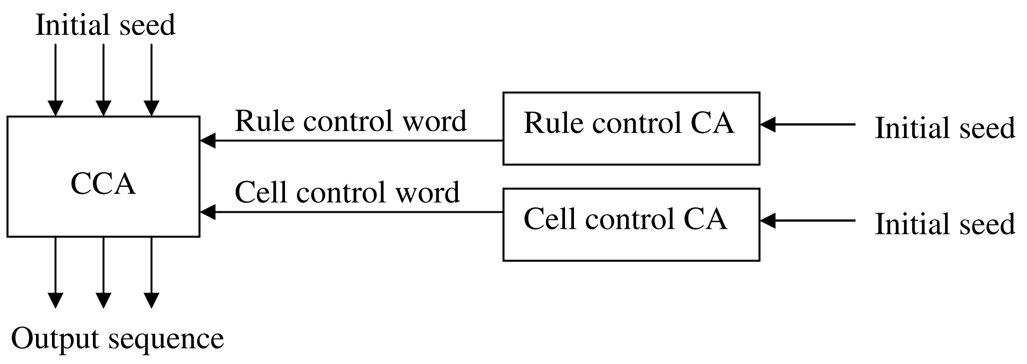

Fig. 1 A CCA PRNG structure

Fig. 1 shows the structure of a CCA PRNG. The running sequence of a CCA PRNG is described as follows. 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 signals for CCA cells. Output sequences are generated by CCA cells. Here, rules 90 and 150 are used as the transition rules in CCA. Rule 30 is used in the rule control CA and rule 105 is used in the cell control CA.

We have presented eight types of CCA in [22]. The following is a brief introduction to them. If an active controllable cell keeps its state and the states of its neighbors are computed as usual, it is a type 0 controllable cell. If the state of an active controllable cell is complemented and the computation of its neighbors' states is as usual, it is a type 1 controllable cell. CCA containing type 0 or type 1 controllable cells are referred to as CCA0 or CCA1. If an active controllable cell keeps its latest state while its neighbors bypass it, it is a type 2 controllable cell. CCA with this type of controllable cells are referred to as CCA2 or neighbor-changing CA (NCA).

If an active controllable cell keeps its latest state while its neighbors treat it as a mirror, it is a type 3 controllable cell. CCA with this type of controllable cells are referred to as CCA3 or boundary-changing CA (BCA). If the right neighbors of an active controllable cell bypass it while the left neighbors still use it in the CA computation and the active controllable cell itself keeps its state, it is a type 4 controllable cell. CCA with this type of controllable cells are referred to as CCA4.

Type 2 controllable cells can also be modified as follows: an active controllable cell will make transitions according to some transition rules while its neighbors will perform the action as defined in type 2. Setting the rule to 30, 105, and 165 respectively, we get type 5, type 6, and type 7 controllable cells. The corresponding CCA are referred to as CCA5, CCA6, and CCA7 individually. Obviously, different transition rule choices will affect the randomness of these types of CCA. We have shown in [22] that the properties of these three types are similar. Therefore, we will study only CCA5 among these three generators.

3. Evolution of CCA/PCA PRNGs

3.1 Objectives

In our earlier work [22], CCA have been shown to outperform 1-bit PCA and uniform CA and be comparable to 2-d CA in terms of randomness/cycle length. These CCA are handcrafted. Their structural complexity is higher than that of a 1-bit PCA with the same length because the structure of a controllable cell is more complex than that of 1-bit programmable cell. Compared to 2-d CA in which no spacing is used to generate output bits, the CCA we tested use cell spacing at 2 and time spacing at 1. It would be interesting to know whether the structural complexity and output efficiency of CCA can be optimized while their randomness quality is maintained in the meantime.

Basically, the structural complexity of CA is decided by three factors. They are the length of CA (*L*: total number of cells in CA), number of controllable/programmable cells in CA (CN), and controllable/programmable cell type. For CCA with a specified controllable cell type, the length of CCA and number of controllable cells decide the complexity. For PCA, the length of PCA and the programmable cell type (1-bit or 2-bit) decide the complexity. Specially, for uniform CA, the structural complexity depends on the length of CA only. When cell spacing and time spacing are used, the output efficiency of CA is decided by the amount of spacing. In practice, cell spacing and time spacing should be made as low as possible to avoid low bit rate. Here, we set the upper bound of spacing parameter to 7.

For a certain type of CCA, we aim at both maximizing their output efficiency and minimizing their structural complexity while maintaining good randomness quality. To evaluate the randomness of individual CCA, we apply the ENT test suite [24] and DIEHARD test suite to the sequences generated. Three tests are chosen from ENT to indicate randomness. They are entropy, serial correlation coefficient (SCC), and chi-square. The average entropy, SCC, and chi-square values are calculated to evaluate the randomness of each CCA tested.

The evolution of CCA involves two evolutionary algorithms — MOGA and incremental evolution. MOGA is used to minimize the number of controllable cells, cell spacing, and time spacing parameter for a fixed-length CCA. Incremental evolution is applied to find the minimal length of a CCA to pass DIEHARD. In the evolution process, these two algorithms are interleaved with each other. Incremental evolution decides the length of the CCA to be evolved in MOGA. MOGA evolves the CCA at specified length and finds whether or not the evolved CCA could pass DIEHARD. The DIEHARD test results will be used as feedback to incremental evolution, which decides whether to adjust the length of CCA and continue the evolution process or to end the evolution process.

3.2 Evolution of Fixed-length CCA PRNGs

MOGA are widely used to solve engineering problems where simultaneous optimization of multiple, often competing, objectives is required. Various schemes have been developed in recent years [3, 9]. These techniques could be divided into two categories: population-based approach and Pareto-based approach. On the whole, the population-based approach has a common deficiency that it tends to generate solutions such that one of the objectives is extremely good but the other objectives are not so [9]. Hence, we use the Pareto-based approach.

Different from the population-based approaches, the Pareto-based approach performs selection/reproduction by referring not only to the objective values themselves but also to the dominance property of them. Among several proposed schemes, we choose Fonseca and Fleming's [2] as our basic algorithm. In their scheme, the rank of each individual is defined by one plus the number of individuals in the current population that dominates it. The encoding of chromosome and setting of algorithm parameters are discussed below.

For each fixed-type CCA, we consider the value of six objectives: number of controllable cells, cell spacing parameter, time spacing parameter, entropy value, SCC value, and chi-square value. The first three objectives are to be minimized and the last three are to be maximized (SCC value is revised to 1-SCC during the evolution process to ensure that a larger SCC value means better randomness). Each chromosome has L+3+3 (L is the length of CCA) bits. The first L bits stand for CCA configuration that decides the number and position of controllable cells in CCA. The next three bits represent the cell spacing parameter as a binary number. It is 3 because we set the upper bound of spacing to 7. The last three bits represents the time spacing parameter.

Algorithm 1: Evolution of fixed-length CCA PRNGs

```
Input: P chromosomes that are received from incremental evolution
//evolution
While (stopping criteria is not true) do
        Calculate the objective values of each chromosome;
        Calculate the Pareto-rank of each chromosome;
        Perform crossover and mutation to generate P child chromosomes;
        Calculate the objective values of each child chromosome;
        Calculate the Pareto-rank of each child chromosome;
        Copy the first half of parent chromosomes and first half of child
        chromosomes to the next generation;
End while
Output: P chromosomes in the last generation
```

The detail of the algorithm is presented in Algorithm 1. The input of MOGA consists of *P* chromosomes that are passed from incremental evolution. The output is *P* chromosomes in the last generation. The total population size *P* is 80. The stopping criterion is the maximum stagnation steps, which is 200. If the best chromosome keeps unchanged for 200 continuous evolution steps, the process will be stopped. The 1-point crossover rate is set at 0.95. The bit mutation rate is set at 0.05. During reproduction, half of both the better-performing parent and child chromosomes will be copied into the next generation.

3.3 Incremental Evolution of CCA/PCA PRNGs

Incremental evolution could be used to find the minimal length of CA to pass DIEHARD where CA could be PCA or CCA. The proposed incremental evolutionary algorithm is able to decide at each stage whether to add or delete cells from CA and how cells should be added or deleted. A general description of the evolutionary algorithm on CA PRNGs is presented in Algorithm 2, where MOGA and the incremental evolution algorithm collaborate to determine the minimal length with which CA could pass DIEHARD.

Algorithm 2: Evolution of CA PRNGs

```
Input: none
Set the initial length of CA as $L1$ ($L1$=50);
Randomly initialize $P$ ($P$=80) CA;
Evolve CA using Algorithm 1;
While (the evolved CA fail to pass DIEHARD)
    • Increase the length of CA according to the incremental evolution
      algorithm described in Algorithm 3, record the current length of CA
      $Lc$;
    • Evolve $P$ CA with the length adjusted using Algorithm 1;
End while;
While (the evolved CA pass DIEHARD)
    • Decrease the length of CA according to the incremental evolution
      algorithm, record the current length of CA $Lc$;
    • Evolve $P$ CA with the length adjusted using Algorithm 1;
End while;
While (the evolved CA fail to pass DIEHARD)
    • Increase the length of CA by 1;
    • Evolve $P$ CA with the length adjusted using Algorithm 1;
End while;
Output: The non-dominated chromosomes in the last population to form the
candidate group for preference selection.
```

In the beginning of the evolution, CA will start with a small number of cells (50 cells). MOGA is then used to optimize the 50-cell CA. We already know that 50-cell CCA cannot pass DIEHARD and CA with more cells will have better randomness quality.

Hence, an incremental evolution algorithm is applied to increase the length of CA at a higher rate (*Si*= 10) until a length with which CA could pass DIEHARD is found. Apparently, this length may not be the minimal length we are looking for. Thus, the incremental evolution algorithm is applied again to decrease the length of CA at a lower rate (*Sd*=5) until a point where the length of CA is close to and yet less than the minimal length to pass DIEHARD. From this point, the length of CA is increased by 1 until we find the minimal length of CA to pass DIEHARD.

Algorithm 3: Incremental evolution algorithm

```
Input: current length Lc, decrease/increase flag
Si=10; Sd=5;
Switch (flag)
{
        case '+':
                Increase the length of CA from Lc to Lc+Si;
                Update Lc with Lc+Si;
                Break;
        case '–':
                Decrease the length of CA from Lc to Lc-Sd;
                Update Lc with Lc-Sd;
                Break;
};
Update Lc with Lc+Si or Lc-Sd;
Output: current length Lc
```

The incremental evolution algorithm is presented in Algorithm 3. Initially, the growth of cells will be at a faster rate as the tested CA may be quite far from the required number of cells at the beginning. If the evolved CA manage to pass DIEHARD, the evolution process will return to the previous step. At this time, the growth of cells will slow down until the evolution process manages to find the point whereby CA can pass DIEHARD.

The increasing/decreasing size (*Si/Sd*) in CA length could be adjusted dynamically during the evolution. This size depends on the performance of tested CA. The initial size could be set to a large number. Here, we set *Si* at 10 and *Sd* at 5 in the beginning. And then *Si/Sd* could be decreased while the length of CA is closer to the minimal length to pass DIEHARD.

The process to decide whether or not the evolved CA can pass DIEHARD includes three steps. First, MOGA evolution generates a group of non-dominated chromosomes. Second, one chromosome is selected according to the predefined preference set, which will be discussed in section 4.1. Third, the selected chromosome is tested under DIEHARD. If it fails to pass DIEHARD, we think that the evolved CA most likely cannot pass DIEHARD. Once the minimal length of CA to pass DIEHARD is found (assume the length is *L*), the evolution process will check the randomness of (*L*-1)-cell CA again. 10 chromosomes that obtain the best randomness quality among the evolved (*L*-1)-cell CA are tested under DIEHARD. If none of the chromosomes tested could pass DIEHARD, we think that the minimal length found is valid. Because exhaustive searching is impossible, we could not exclude the possibility that some (*L*-1)-cell CA could pass DIEHARD. Given the resource constraints, we feel that the minimal length (*L*) found is acceptable under the above-mentioned validation scheme.

The process to increase or decrease the length of CA involves both algorithms. When the length of CA is changed, the chromosomes evolved using MOGA should be adjusted too. One method is to generate initial population randomly in MOGA evolution no matter what the length of CA is. But this method somehow wastes the evolution effort from previous generations. It is natural to keep the evolved chromosomes from the previous generations as the initial population for the current generation. The problem is that the length of chromosomes has changed during the transition from the last generation to the current one. When the length is decreased, we could just truncate the superfluous bits from the chromosomes. Which bits to be deleted is decided randomly. Note that modification is only applied to the first L bits in the chromosome. The six bits encoding cell spacing and time spacing will not be changed. Similarly, bits are randomly generated and added to the chromosomes in the last generation when the length of CA is increased.

In the next section, we first present the evolution results on fixed-length CCA and then present the incremental evolution results on CCA/PCA. Moreover, the evolved CCA are compared to 1-d/2-d CA not only in randomness but also in complexity and efficiency.

4. Evolution Results

4.1 Evolution Results on 50-cell CCA PRNGs

The evolution results on fixed-length CCA PRNGs are a group of non-dominated chromosomes from which we can extract the number and location of controllable cells in CCA, cell spacing parameter, and time spacing parameter. One chromosome is selected from the non-dominated group according to the predefined preference. The preferences are set as follows. The selection process includes two steps.

Step 1: choose chromosomes from a group of non-dominated chromosomes in the last generation using the chi-square value as preference. Maintaining the randomness quality of CCA is a pre-requisite requirement. Hence, we must ensure that the selected chromosomes could generate good random number sequences. In this step, we use the chi-square value alone to evaluate randomness. Only those non-dominated chromosomes that obtain a chi-square value at 1 are chosen as the final candidates. Further check on randomness quality will be done in the next step using the entropy and SCC values. In the next step, one chromosome will be chosen from the final candidates as the best chromosome evolved.

Step 2: choose the best chromosome. The following rules are applied in descending priority order to choose one chromosome from the final candidates:

Rule 1: compare cell spacing & time spacing parameter: a smaller value is preferred because it means better output efficiency. The sum of cell spacing and time spacing parameter is first compared and then if the sum is equal, the one with the smallest time spacing parameter will be chosen.

Rule 2: compare the number of controllable cells: a smaller value is preferred because it means less structural complexity.

Rule 3: compare 1-SCC and entropy values: the higher the sum of (1-SCC) and entropy value is, the better randomness quality the chromosome obtains. Generally, the final candidates obtain similar entropy and SCC values. The reason that entropy and SCC values are set to the lowest priority is because we think obtaining high output efficiency and low structural complexity are more important than obtaining slightly better entropy and SCC values for CCA.

The chromosomes in the final candidate group will first be compared according to rule 1. If only one chromosome is selected, it will be the best solution. If more than one chromosome is selected, these chromosomes will be compared under rule 2 and so on. Normally, the sum of entropy and (1-SCC) values will not be identical for two chromosomes. Hence, we could ensure that only one chromosome is selected out of the non-dominated chromosomes.

The objective values of the best solutions for CCA0-CCA5 are shown in Table 1. The entropy and SCC values are average values. The chi-square value encodes the pass rate: 100% means all the sequences tested can pass the chi-square test. Each CCA/PCA has 50 cells. Referring to Table 1, we can see that all the tested CCA can pass chi-square at 100% while the 1-bit and 2-bit PCA pass at lower rates (83% and 92%). The entropy and SCC values obtained by the evolved CCA and 2-bit PCA are in the same range while those of 1-bit PCA are worse. Note that except 1-bit PCA that uses time spacing at 1, the others require no spacing.

Table 1. Objective values of evolved 50-cell CCA/PCA PRNGs

	Number of control bits Per PCA/CCA	cs	ts	entropy	SCC	chi-square
CCA0	58	0	0	7.872323	0.012145	1.0 (100%)
CCA1	58	0	0	7.876290	0.017982	1.0 (100%)
CCA2	58	0	0	7.892309	0.012624	1.0 (100%)
CCA3	56	0	0	7.883857	0.009930	1.0 (100%)
CCA4	58	0	0	7.886308	0.003207	1.0 (100%)
CCA5	59	0	0	7.889176	0.005018	1.0 (100%)
1-bit PCA	50	0	1	7.758143	0.041355	0.83 (83%)
2-bit PCA	100	0	0	7.872308	0.020187	0.92 (92%)

Legends: cs: cell spacing parameter; ts: time spacing parameter. Number of control bits: total number of rule control bits and cell control bits per CCA/PCA.

Compared to PCA with the same length, the structural complexity of CCA is higher due to the usage of several controllable cells as indicated in Table 1. Yet the output efficiency of CCA is higher. The output efficiency of the evolved CCA2 is two times that of the 1-bit PCA PRNG with the same length. That is to say, to compete with CCA in randomness and output efficiency, the length of 1-bit PCA should be increased.

But this will in turn increase the cost of 1-bit PCA. The randomness quality of 2-bit PCA is closer to CCA0/CCA1/CCA5. The complexity of a 2-bit programmable cell is at same level as a controllable cell because both of them use 2-bit control signals. Therefore, we think that the structural complexity of 2-bit PCA is similar to that of CCA.

Table 2 shows the DIEHRAD test results of the evolved 50-cell CCA and PCA PRNGs. We can see that CCA2 fail only one test while 1-bit PCA fails 7 tests. The other CCA/PCA obtain some performance between CCA2 and 1-bit PCA.

Table 2. DIEHARD test results of CCA/PCA PRNGs

Test name	L=50							
	CCA0 cs=0 ts=0	CCA1 cs=0 ts=0	CCA2 cs=0 ts=0	CCA3 cs=0 ts=0	CCA4 cs=0 ts=0	CCA5 cs=0 ts=0	1-bit PCA cs=0 ts=1	2-bit PCA cs=0 ts=0
1. Overlapping sum	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
2. Runs up 1	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
Runs Down 1	Pass	Pass	Pass	Pass	Pass	Pass	Fail	Pass
Runs up 2	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
Runs Down 2	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
3. 3D sphere	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
4. A parking lot	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
5. Birthday Spacing	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
6. Count the ones 1	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail
7. Binary Rank 6*8	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
8. Binary Rank 31*31	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
9. Binary Rank 32*32	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
10. Count the ones 2	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
11. Bitstream test	Pass	Pass	Pass	Pass	Pass	Pass	Fail	Pass
12. Craps wins	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
games	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
13. Minimum distance	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
14. Overlapping Permu	Pass	Pass	Pass	Pass	Pass	Pass	Fail	Pass
15. Squeeze	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
16. OPSO test	Fail	Fail	Pass	Fail	Pass	Fail	Fail	Fail
17. OQSO test	Fail	Fail	Pass	Fail	Fail	Fail	Fail	Fail
18. DNA test	Pass	Fail	Pass	Pass	Pass	Fail	Fail	Pass
number of tests passed	***15***	***14***	***17***	***15***	***16***	***14***	***11***	***15***

Legends: cs: cell spacing parameter; ts: time spacing parameter;
L: total number of cells in CCA.

4.2 Evolution Results from Incremental Evolution

The evolution results from incremental evolution on CCA and PCA PRNGs are shown in Figure 2. CCA2 and CCA4 can pass DIEHARD with 55 cells. Next, CCA3 can pass at a minimum number of 58 cells. CCA5, CCA0 and CCA1 can pass with 67, 73, and 74 cells individually. The minimum number of cells required for 1-bit PCA is remarkably higher than that of CCA. 1-bit PCA needs 109 cells with time spacing at 1 to pass DIEHARD while 2-bit PCA needs 71 cells without any spacing. The configurations of CCA/PCA to pass DIEHARD are shown in Table 3.

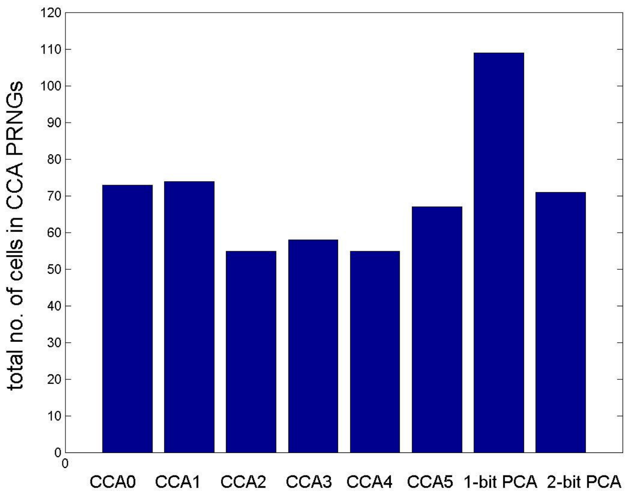

Fig. 2 Evolution results from incremental evolution

Table 3. Configuration of CCA/PCA PRNGs

	CCA0	CCA1	CCA2	CCA3	CCA4	CCA5	1-bit PCA	2-bit PCA
cn	14	14	12	10	11	16	/	/
ncb	87	88	77	68	66	83	109	142
cs	0	0	0	0	0	0	0	0
ts	0	0	0	0	0	0	1	0
L	73	74	55	58	55	67	109	71

Legends: cn: no. of controllable cells; cs: cell spacing parameter; ts: time spacing parameter. ncb: number of control bits per CCA/PCA, including rule control bits and cell control bits per each CCA/PCA.

Referring to Table 3, we can see that the evolved 55-cell CCA2/CCA4 and 58-cell CCA3 could pass DIEHARD without any spacing. Tomassini et al. have shown in [15] that 64-cell 2-d CA could pass DIEHARD without spacing while it is unknown whether or not 2-d CA with fewer cells could pass DIEHARD. Thus, we may claim that CCA2-CCA4 could compete with 2-d CA not only in randomness quality but also in output efficiency. The performance of CCA0/CCA1/CCA5 is similar to that of 2-bit PCA and better than that of 1-bit PCA.

4.3 Discussions

We have shown that the evolved CCA can compete with conventional 1-d/2-d CA in randomness, structure simplicity, and output efficiency. And the performance of different types of CCA varies greatly. CCA2/CCA4 obviously outperform CCA0/CCA1. One possible reason may lie in that the action embedded in type 0/1 controllable cells is not so powerful as that of type 2/4 controllable cells. That is to say, the performance of CCA depends much on the action of controllable cells. It is worth to explore whether there are some other controllable cell actions that could lead to better CCA performance.

We have discussed how to adjust the length of CA during incremental evolution in section 3.3. In Algorithm 2, the new cells are randomly initialized and added to the chromosomes when the length of CA is increased. According to our previous experiment results, we find that having too many or too few controllable cells could degrade the randomness of CCA. Thus, when adding new cells (bits) to the chromosomes, we could first check the number of controllable cells included in the chromosomes and then decide whether to add basic cells or controllable cells. If the ratio of controllable cells to basic cells is low, we could add a few more controllable cells to the chromosomes. Otherwise, we could add basic cells. Moreover, we could also choose the location for the cells to be added. Generally, the number of connected controllable cells will not be greater than 4. Hence, we could interleave basic cells and controllable cells when adding bits to the chromosomes. Similarly, we could also choose the location and type of cells to be deleted when the length of CCA is decreased.

5. Conclusion

In this paper, we have discussed the randomness, structural complexity, and output efficiency of CCA/PCA. These three aspects are evolved together to find some CCA that could generate good random number sequences at a low complexity and high output efficiency. The evolution process involves two algorithms. MOGA is employed to evolve fixed-length CCA. An incremental evolutionary algorithm is applied to find the minimal length of CCA/PCA to pass DIEHARD. The merit of incremental evolution lies in its self-adjusting ability, which saves time and effort for searching and computation.

This evolution process is also applied to 1-bit/2-bit PCA. The comparison on CCA and 1-d PCA shows that CCA outperform PCA not only in randomness but also in structure simplicity and efficiency. Besides, CCA could compete with the evolved 2-d CA in the literature in both randomness and output efficiency. Comparison among CCA shows that CCA2/CCA4 outperform the other types of CCA. Some future work could be done to explore what leads to the good performance of these two CCA. The methodology developed here is not limited to CCA/PCA only, it can be used to evolve other CA PRNGs or serve as a yardstick to compare their performance.

Acknowledgement The authors would like to thank Tai Lim Lee for his initial implementation work on incremental evolution.

References

[1] Barry Shackleford, Motoo Tanaka, Richard J. Carter, and Greg Snider, High-performance cellular automata random number generators for embedded probabilistic computing systems, In Proc. of the 2002 NASA/DOD Conference on Evolvable Hardware, 2002.

[2] C. M. Fonseca and P. J. Fleming, Genetic algorithm for multiobjective optimization: formation, discussion and generalization, In Proc. of the 5th ICGA, 1993, pp. 416-423.

[3] Carlos A. Coello Coello, An updated survey of evolutionary multiobjective optimization techniques: state of the art and future trends, In Proc. of the 1999 Congress on Evolutionary Computation, Vol. 1, 1999, pp. 3-13.

[4] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

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

[6] D. R. Chowdhury, I. S. Gupta and P. Pal Chaudhuri, A class of two-dimensional cellular automata and applications in random pattern testing, Journal of Electrical Testing: Theory and Applications, Vol. 5, 1994, pp. 65-80.

[7] Dragan Cvetkovic, Ian Parmee, Preference and application in evolutionary multiobjective optimization, IEEE Transactions on Evolutionary Computation, Vol. 6, Issue 1, Feb. 2002, pp. 42-57.

[8] G. Marsaglia, “Diehard”, http://stat.fsu.edu/~geo/diehard.html, 1998.

[9] Hisashi Tamaki, Hajime Kita and Shigenobu Kobayashi, Multi-objective optimization by genetic algorithms: a review, In Proc. of IEEE International Conference on Evolutionary Computation, 1996, pp. 517-522.

[10] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation, Vol. 6, No.2, April 2002, pp. 182-197.

[11] M. Matsumoto, Simple cellular automata as pseudorandom m-sequence generators for built-in self-test, ACM Trans. on Modeling and Computer Simulation, Vol. 8, No. 1, 1998, pp. 31-42.

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

[13] M. Mihaljevic and Hideki 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, 1999, pp. 32-39.

[14] Marco Tomassini, Moshe Sipper, Mose Zolla and Mathieu Perrenoud, Generating high-quality random numbers in parallel by cellular automata, Future Generation Computer Systems, Vol. 16, 1999, pp. 291-305.

[15] Marco Tomassini, Moshe Sipper and Mathieu Perrenoud, On the generation of high-quality random numbers by two-dimensional cellular automata, IEEE Trans. Comput., Vol. 49, 2000, pp. 1146-1151.

[16] Moshe Sipper and Marco Tomassini, Generating parallel random number generators by cellular programming, International Journal. Modern Physics, Vol. 7, No.2, 1996, pp.181-190.

[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, 1989, pp. 1466-1473.

[18] P. D. Hortensius, R. D. Mcleod, Werner Pries, D. Michael Miller and H. C. Card, Cellular automata-based pseudorandom number generators for built-in self-test, IEEE Transactions on Computer-Aided Design, Vol. 8, No. 8, 1989, pp. 842-859.

[19] S. Nandi, B. K. Kar, and P. Pal Chaudhuri, Theory and applications of cellular automata in cryptography, IEEE Trans. Computers. 43, 1994, pp. 1346-1357.

[20] S. Wolfram, Theory and Applications of Cellular Automata: Including Selected Papers 1983-1986, World Scientific publishing Co., Inc., River Edge, NJ. 1986.

[21] Sheng-Uei Guan and Shu Zhang, "An Encryption Method based on Dynamic Cellular Automata", Proceedings the International ICSC Congress on Intelligent Systems and Applications, University of Wollongong, Australia, # 1514-074(CD number: ISBN 3-906454-24-X), December 12-15, 2000.

[22] Sheng-Uei Guan and Shu Zhang, A family of controllable cellular automata for pseudorandom number generation, to appear in International Journal of Modern Physics, Computer.

[23] Von Neumann, J., “The general and logical theory of automata”, In J. von Neumann Collected Works”, A. Taub, Ed.

[24] ENT test suite, http://www.fourmilab.ch/random

Appendix

1. The configurations of the evolved CCA0-CCA5 with the minimal length to pass DIEHARD are presented in Table 4.

Table 4 The configurations of evolved CCA0-CCA5

	Configuration
73-cell CCA0	00000110001000010000001000000010100000001000000001000000101100 00000101000
74-cell CCA1	00010100000001001000000011000000100000101000000100000000100010 001000000010
55-cell CCA2	0100100000000001001100000001100001000001000000011010000
58-cell CCA3	0010011000010000001010000000000100000000010000001000010000
55-cell CCA4	0010110000000000100100000100010000000010000101000010000
67-cell CCA5	00000001000110000000010000010001000000010001100011100010001000 11000

Legends: ‘1’ stands for a controllable cell; ‘0’ stands for a basic cell.

Rules used in CCA0-CCA5: rule 90 and 150; rule control CA uses rule 30; cell control CA uses rule 105.

2. The configurations of the 1-bit/2-bit PCA evolved to pass DIEHARD:

109-cell 1-bit PCA: uses rule 90 and 150; rule control CA uses rule 30.

71-cell 2-bit PCA: uses rule 90, 105, 150, and 165. rule control CA1 uses rule 30; rule control CA2 uses rule 105.