CONFIGURABLE CELLULAR AUTOMATA FOR PSEUDORANDOM NUMBER GENERATION

MARIE THERESE QUIETA AND SHENG-UEI GUAN

Department of Electrical and Computer Engineering
National University of Singapore
3 Engineering Drive 3, Singapore 117576

Abstract

This paper proposes a generalized structure of cellular automata (CA) – the configurable cellular automata (CoCA). With selected properties from programmable CA (PCA) and controllable CA (CCA), a new approach to cellular automata is developed. In CoCA, the cells are dynamically reconfigured at run-time via a control CA. Reconfiguration of a cell simply means varying the properties of that cell with time. Some examples of properties to be reconfigured are rule selection, boundary condition, and radius. While the objective of this paper is to propose CoCA as a new CA method, the main focus is to design a CoCA that can function as a good pseudorandom number generator (PRNG). As a PRNG, CoCA can be a suitable candidate as it can pass 17 out of 18 Diehard tests with 31 cells. CoCA PRNG's performance based on Diehard test is considered superior over other CA PRNG works. Moreover, CoCA opens new rooms for research not only in the field of random number generation, but in modeling complex systems as well.

Key words: random number generation, cellular automata, configurable cellular automata

1 Introduction

Cellular automata (CA) was first introduced by John von Neumann [28] in the late 1940s and initiated in the early 1950s to provide modeling and simulation for complex systems capable of self-reproduction. Later on, some researchers maintained active interest in the field and subsequent developments went on. Several research activities have confirmed that CA's inherent parallel architecture provides high-performance computational simulation environments which can be used for solving real-world problems in science and engineering [15]. Few distinct real-world examples are simulations of macroscopic phenomena and biochemical phenomena. In computer simulations, CA has been used in cryptography [11], [20], VLSI testing [12], and pseudorandom number generation [1], [3]-[5], [7]-[9], [14], [17], [18].

For over a decade, one active application of CA is in pseudorandom number generation. Motivations for these works are ascribed to the aspect of CA which can be easily implemented in hardware as they are simple, regular, localized, and are essentially made up of networks of Boolean functions. CA-based pseudorandom number generators (PRNGs) have been studied extensively [1], [3]-[5], [7]-[9], [14], [17], [18] and for the past years, they have been shown to offer superb performance and efficiency. They are superior over other pseudorandom number workhorses like linear congruential generators (LCGs) and linear feedback shift registers (LFSRs) [9], [20].

In this paper, a novel CA which we coin the Configurable CA (CoCA), is proposed with the objective of obtaining good CA-based PRNGs in a more flexible way. In CoCA, CA cells can be configured at run-time via a configuration control CA. CA parameters to

be configured are selected based on some standard CA properties [21] and some new CA properties recently proposed in [4][5].

The rest of this article is organized as follows: Section 2 gives an overview of cellular automata and pseudorandom number generators. Section 3 presents the new approach to CA – the configurable CA. and demonstrates some preliminary experiments on CoCA as a PRNG, Section 4 discusses the evolutionary approach to CoCA which provides analysis on CoCA performance as a random number generator, Section 5 presents some new rooms for further research, and finally, Section 6 concludes the paper.

2 Cellular Automata PRNGs

2.1 Cellular Automata Overview

Cellular automata are discrete dynamic autonomous systems consisting of an array of cells where each cell is in any one of its permissible states, s$\in$ {0,1} for Boolean CA. The cells are updated synchronously at discrete time steps (clock cycle) by certain rule functions. The state of a CA, $X(t)$, at time t is defined as the n-tuples formed from the states of the individual cells, $X(t) = [x_1(t), x_2(t), ..., x_n(t)]$ where n is the number of CA cells. A CA is considered autonomous since it *evolves* from its previous state to its next state. Changing the initial conditions of the CA may result in somewhat different upshots as it evolves in time.

A CA's behavior is completely specified in terms of local relation. The transition rule is a function of the previous states (i.e. at previous time step, t-1) of its k neighbors for a k-neighborhood CA. Normally, each cell's neighborhood considers itself and the cells physically closest to it. For a 3-neighborhood 1-d CA with n number of cells, the

neighborhood of each cell considers itself and the left and right cells directly connected to it. Each cell state transition is given by the equation: $x_i(t) = f_i(x_{i-1}(t-1), x_i(t-1), x_{i+1}(t-1)) \forall i = 1, 2, ..., n$ where f_i represents the transition rule for the ith cell.

In accordance with Wolfram's convention [20], transition rule functions are defined in Boolean forms. For a k-neighborhood CA, there are 2^k possibilities of combining the state values of neighbors. Each combination has an equivalent next state value for a certain cell x_i, depending on the rule function used by that particular cell. For a 3-neighborhood CA, there are $2^3 = 8$ combinations of neighbor states. Performing a rule function on each of the 8 combinations would result in 8 next-state values of x_i. The transition rule names of CA are based on the decimal equivalent of 8 next state values. For instance, if a certain rule function is to 00011110_2 (equivalent to 30_{10}), then the rule function is named rule 30. The most commonly used rule functions in pseudorandom number generation are rules 30, 90, 105, 150, and 165.

If all the cells in a CA obey the same rule, then that CA is said to be *uniform*; otherwise, it is *nonuniform* or *hybrid* [7]. A CA is said to be a *periodic boundary* CA if the extreme cells (leftmost and rightmost 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 (or right) cell [21] and a constant value of '0' or '1' is assigned to its supposed-to-be right (or left) neighboring cell. Research on CA PRNGs has shown that *nonuniform* CA [8] and *periodic boundary* CA [16] give better randomness quality. In this work, *nonuniform periodic boundary* CA is used.

A CA is said to be a *programmable* CA (PCA) if it uses a *control* CA to determine the rules of each cell. A control CA is essentially just another basic CA which is usually of uniform nature. The rule function used by each cell changes with time and is decided by the control CA. PCA is, in fact, a nonuniform CA because all its cells collectively use different rule functions. A PCA may use m-bit control CA, where $m \geq 1$. For each cell, there are 2^m rules to choose from, thereby, allowing less probability of correlations among the cells. As a PRNG, 2-bit PCA has been explored by some researchers [17]. As expected, it showed better performance than a 1-bit PCA PRNG. However, increasing m is not practical as it introduces more hardware implementation costs. For benchmarking purposes, 2-bit PCA is also considered in this paper.

2.2 Randomness Tests

The pseudorandom bits are obtained from CA by sampling the cell state values at certain time steps. Time spacing (*ts*) and cell spacing (*cs*) are often used to avoid correlations among the pseudorandom numbers obtained. The *ts* parameter is the number of time steps in between when CA cells are sampled as output. For example, if the output bits are extracted from the CA PRNG every 2 time steps (*ts* = 2), only CA at specific time steps, $X(t)$, $X(t+2)$, $X(t+4)$, …, $X(t+2i)$ are utilized. The *cs* parameter is the number of cells in between two consecutive output cells in a CA PRNG. For example, if the *cs* parameter is set at 3 (*cs* = 3), then, cells $x_i(t)$, $x_{i+4}(t)$, $x_{i+8}(t)$, $x_n(t)$ are used as output cells. It has been shown that time and cell spacing significantly improves the performance of CA PRNGs [4], [5]. This shows that output methods are non-trivial in pseudorandom number generation.

Like in previous research [3]-[6], two well-known randomness test suites are utilized in this work: the ENT Test [2] and the Diehard test [10].

ENT test is a collective term for three tests: chi-square, entropy, and serial correlation coefficient (SCC). The overall evaluation for the ENT test can be obtained from the *F value* as given in Equation (1). In comparing good quality CA PRNGs, the entropy (*ent*) and SCC values normally have comparable results with minimal discrepancies unlike the chi-square value. Because the chi-square test is an important indication of randomness, it is given the highest weightage in the calculation of *F*.

$$F = (entropy - 7) * 30\% + (1 - |SCC|) * 30\% + f(chi - square) * 40\% \quad (1)$$

$$\text{where: } f(chi - square) = \begin{cases} 0; & \text{if chi-square } > 90\% \text{ or } < 10\% \\ 1; & \text{if } 10\% < \text{chi-square} < 90\% \end{cases}$$

The Diehard test, which is known to be a very stringent test, is also used to further evaluate the randomness of CoCA PRNGs. The Diehard Test is a battery of tests in which each test calculates *p-values*. There are 18 tests all in all including those tests that were not mentioned in Tomassini's work [18]. These are the OPSO test, OQSO test, and DNA test, which are said to be the more difficult tests to pass as there are many *p-values* to consider. In this work, a pass is considered if the *p-value* is not 0 or 1.

It is observed that if a CA PRNG has $F < 0.9$ (assuming that number of initial pseudorandom seeds, $S = 100$), the CA PRNG has low chances of passing the Diehard test [4], [5]. Moreover, generating random bits to be tested under Diehard test is more time-consuming than generating for ENT test. This is because Diehard test requires more random numbers than the ENT test. Thus, in this work, some comparisons are made using ENT test first, e.g., in preliminary experiments of CoCA PRNG. Diehard test is

used to verify the results of CoCA PRNGs which gives good ENT test results. In the end, Diehard test is still the gauge of a good pseudorandom number generator.

2.3 CA PRNGs (1-d CA/PCA, 2-d CA, SPCA, CCA)

Research on CA PRNGs has diversified from internal properties (e.g., locality, rule selection, and programmability) to external structures like 1-d string, 2-d array, and 3-d solid. Combinations of some internal properties with the external structure property has opened a wide range of research on CA as a PRNG, e.g., varying rule selection of 1-d CA pioneered the research on 1-d PCA. Hence, research interest on CA PRNGs intensifies from time to time as new developments continue to emerge.

In the last decade, research interests on CA PRNGs were focused on 1-d CA. The first work on CA as PRNG can be credited to Wolfram [20]. He used uniform rule 30 1-d CA and showed that it can produce fairly random temporal bit sequences. Later, Hortensius et al. studied the first nonuniform CA PRNG, the programmable CA (PCA) [7]. In their work, Hortensius et al. explored on 1-bit control PCA using rule 90 and rule 150, coined as PCA 90-150. Hortensius et al. also researched on PCA 30-45 showing that PCA 90-150 has better potential than PCA 30-45 in pseudorandom number generation [8]. Shortly after Hortensius' work, Tomassini et al. further explored PCA 90-165 [14] and 2-bit control PCA referred to as PCA 90-105 [17]. Results showed that the latter of Tomassini's work is better than Hortensius' work. But they are still not comparable to classical generators (e.g. LCG) in terms of randomness quality and cannot pass Diehard test by Marsaglia [10], the most stringent randomness test known at present.

While some works were concentrated on thorough searching for good PRNGs using 1-d CA, some researchers shifted their exploration to 2-d CA PRNGs. Chowdhury et al. first proposed a methodology which generates pseudorandom number using 2-d CA [1]. They showed that 2-d CA PRNG is superior to 1-d CA PRNG using the same number of cells. Tomassini et al. worked on 2-d CA as well [17]. In their work, he studied time spacing parameters and recommended time spacing of *2* for practicality.

Apparently, previous works on 1-d CA are directed to varying rule and rule selection methods. Guan et al. recently proposed new types of CA, controllable CA (CCA) [4]-[5] and self-programmable CA (SPCA) [3]. CCA was designed for the purpose of disregarding the tradeoff between randomness quality and structural complexity. This is because CCA PRNG is one-dimensional but the performance can compete with that of 2-d CA PRNGs. On the other hand, SPCA can also compete with 2-d CA PRNGs and CCA PRNGs. But in terms of cost-effectiveness, SPCA is somewhat less attractive than CCA PRNGs since the former uses memory cells to store the previous state values (way back to *t-2* state values) of the cells. As will be seen later, CoCA PRNG is essentially a generalized form of CA PRNG. Hence, it can not be simply compared with other CA PRNGs based on randomness quality alone. Complexity and hardware implementation should also be taken into consideration. Moreover, a CoCA may be combined in context with the related work like SPCA and 2-d CA PRNGs for further improvement of CA PRNG's randomness quality.

3 Configurable Cellular Automata

3.1 Configurable Cellular Automata Overview

Configurable cellular automata (CoCA) is a generalized CA in which the CA cells are dynamically reconfigured at run-time via a control CA. Reconfiguration of a cell simply means varying the properties of that cell with time. From conventional CA theories of Wolfram [21] and some novel CA PRNG properties in [4]-[5], five CA properties can be derived. Each item is discussed in detailed below.

Rule Selection (RS). The transition rule to be used by each cell can be reconfigured at run-time. In this paper, rules can be selected from rules 30, 105, 165, and 195. These rules are chosen based on the performance in previous works. Varying rule selection in CoCA is equivalent to a 2-bit PCA. Thus, for the RS item, 2 bits of control CA are allotted for each cell.

Status and Reference (SR). The SR item defines the state of each cell at a specific time step. If the status of a cell is normal (S=1), then, the cell is updated at that particular time step. Otherwise, it is not updated. If a cell is referenced (R=1), then, that cell will be used by its left and right neighbors in their state transition at that particular time step. Otherwise, the left and right neighbors will use the nearest referenced cell's state value in their state transition. The status and reference properties are derived from the CCA PRNG properties. The 'reference' property dynamically changes the neighborhood of CA. [4]-[5] show that using status and reference properties in some cells can improve the randomness of CA PRNGs. For the SR item, 2 bits are allotted for each cell.

Boundary Condition (BC). In this paper, boundary condition does not refer to the neighborhood characteristics of extreme cells (leftmost and rightmost) in a CA. Rather, BC here refers to the connections of cells with its neighbors. Basically, there are three types of boundary conditions: *null boundary*, *mirror boundary*, and *normal boundary*. If

a cell assumes a null BC, then, that cell does not use the state values of neighbors in its state computation. A constant value of '0' or '1' is assigned to its supposed-to-be left and right neighbors. If a cell assumes a mirror BC, the cell's left and right neighbors treat that cell as a mirror, i.e. the left (right) neighbor uses its own state in state computation to replace the state value of the cell assuming a mirror BC. If a cell assumes a normal BC, then, that CA cell is updated as usual using its left and right neighbors. Like SR, the mirror and null BCs are also derived from CCA properties. For BC, since there are left BC (LBC) and right BC (RBC), 4 bits of control CA are allotted for each cell.

Output Status (OU). Output method, as mentioned earlier, is very important in pseudorandom number generation. Instead of using time spacing and cell spacing, bit sampling is used in CoCA. This time, the cells are not sampled in a regular manner as in *ts* and *cs*. The output status determines if a cell is to be used as an output cell or not, for a particular time step. For the OU item, 1 bit of control CA is allotted for each cell.

Considering the four items, the total number of control bits for each cell is 9 bits. The control bits are generated by a uniform CA with the structure as shown in Figure 1.

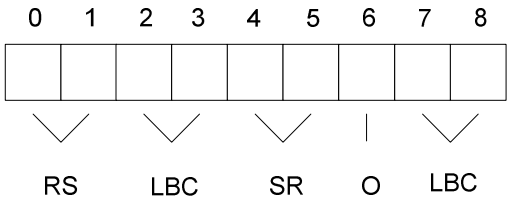


Figure 1 Control CA Structure

Depending on how frequent the cells are reconfigured, CoCA can take on three basic configuration methods:

1. *Fixed configuration* (*F*). The configurations of the cells are pre-assigned and do not change dynamically with time. Normally, CA PRNGs are tested with different

initial seeds. So, for each initial seed tested, a different CA configuration is used. Because CA cell configurations do not change with time, configuration method F does not need a control CA.

2. *Configured at run-time per time step* (*C*). The cells can be reconfigured at every discrete time step. While the control CA cells change states at every time step, its cell state values are used to reconfigure the cells of CoCA.
3. *Configured at run-time per T time step* (*E*). The cells can also be reconfigured at every T time step. Configuration method C is actually a special case of configuration method E, where T=1. Both configuration methods require a control CA so that cell configurations are dynamically changed with time.

Before proceeding on to the experiments, a few terminologies for CoCA are now established.

Uniform cell configuration is when all the cells in CoCA use the same configuration, thereby requiring one 9-cell control CA.

Non-uniform cell configuration is when each cell uses a unique configuration. For a CoCA of length *L*, the required number of 9-cell control CA is *L*.

Non-item-dependent configuration means that the items are reconfigured at the same time depending on the configuration method used (C or E). For example, if E configuration method with T=3 is chosen, all the items RS, SR, BC and OU are reconfigured at the same time for every 3 discrete time steps.

Item-dependent configuration means that the items are not reconfigured at the same time. For example: F configuration method may be used by item RS; C configuration method may be used by item SR; and E configuration may be used by item OU.

Given the definitions and considerations of CoCA, several approaches in the implementation of configurable CA are introduced. The approaches are summarized below:

Approach 1. *Uniform cell configuration and non-item dependent for F, C, and E configuration methods.* In this approach, using the C and E configuration methods require a 9-cell control CA. The four items RS, SR, BC, and OU are all updated at the same time.

Approach 2. *Uniform cell configuration and item-dependent for C and E configuration methods*. Being item-dependent or not does not affect the use of F configuration method. Thus, in this approach, only C and E configuration methods are considered. Both methods require a 9-cell control CA because of uniform cell configuration. But in this approach, the items RS, SR, BC and OU are not updated at the same time.

Approach 3. *Nonuniform cell configuration and non-item dependent for F, C, and E configuration methods.* Because of nonuniform cell configuration, each cell would need a 9-cell uniform CA. So, for an L-cell CoCA, L 9-cell control CA is required. All the four items in this approach are updated at the same time for all cells.

Approach 4. *Nonuniform cell configuration and item-dependent for C and E configuration methods*. This approach also requires L 9-cell control CA to control the configuration. C and E configuration methods are applied in this approach. The items are not updated at the same time.

3.2 Preliminary Experiments on CoCA

The succeeding experiments will show a study of the four different approaches that were presented in the previous section. For the reason that the four different approaches are very much associated with each other, the experimental setups may introduce confusion to the reader. To avoid such confusion, the approach will be clearly identified in each experiment. Also, it is worth noting that each set of experimental results serves as a foundation of the next experiment. To provide rational analysis, the summary of the course of experiments is described as follows. First, the items to be reconfigured are analyzed individually by following CoCA Approach 4. Second, CoCA Approach 1 and Approach 2 are studied and compared with a fixed CA. Thirdly, CoCA Approach 3 and Approach 4 are compared by varying the number of configurable cells. Lastly, genetic algorithms are applied to CoCA PRNGs in order to find good CoCA PRNGs that can pass Diehard test. The head start of analyzing CoCA PRNGs is based on ENT results. Specifically, the preliminary experiments provide some analysis via ENT results. Similar to other works [4], [5], [17], Diehard test is applied after the evolution process to ensure that the evolved PRNGs are of good quality.

3.2.1 Item-based Analysis

Each of the four items has a significant effect in the randomness of bits produced by CoCA PRNGs. It is important then to analyze the effect of reconfiguring each item on CA. In the following experiments, CoCA items are studied by reconfiguring each item at run-time while fixing the others at the *usual CA configuration*. By *usual CA configuration*, it means that: 1) all the cells are updated and referenced during every time step (S=1, R=1), 2) all the cells use a normal boundary condition (BC), and 3) all the cells are used as output cells. A *nonuniform cell configuration* where each cell uses a

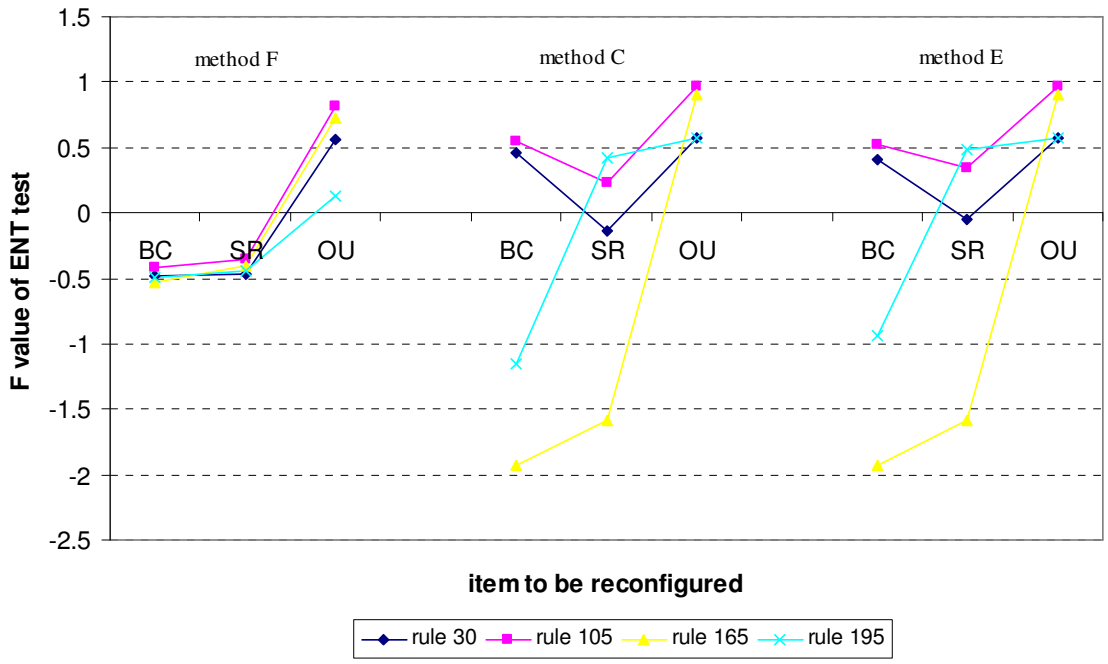
unique control CA is assumed here. For benchmarking purposes, if configuration method E is used, the item/s is/are updated every T=3 time step. There are L 9-cell control CA used in experimenting on configuration methods C and E as described in the later part of Section 3.1. Each 9-cell control CA is *a uniform CA* which uses rule 30 as the transition rule function.

Figures 2 and 3 compare the performance of CoCA PRNGs based on the ENT test. Each point in the graph is an F value (see Section 2.2) of the ENT test after averaging the F value of 300 initial seeds fed into a CoCA. Each graph is divided into three sets via the configuration methods: method F, method C, and method E. A point corresponding to BC in the x-axis under the method C set and rule 105 in the legend means that it is a CoCA where only item BC is reconfigured every time step, transition rule is fixed at rule 105, and other items are fixed at the usual CA configuration (i.e. all the cells are updated and referenced at every time step and all the cells are used as output cells). If, for example, a point corresponds to SR in the x-axis under the method E set and mirror, normal in the legend, then that point represents a CoCA where only item SR is reconfigured every T=3 time step, mirror and normal BCs are used by left and right neighbors, all the cells are used as output cells. If a fixed rule is necessary, rule 105 is used as the transition rule in the CoCA experiments shown in Figure 3. The use of rule 105 is motivated by the results shown in Figure 2, which will be discussed later.

Essentially, the CoCA PRNGs described in Figures 2 and 3 are considered CoCA PRNGs following Approach 4, *nonuniform cell configuration and item-dependent for C and E*. These CoCA PRNGs are in fact item-dependent CoCA because one item is

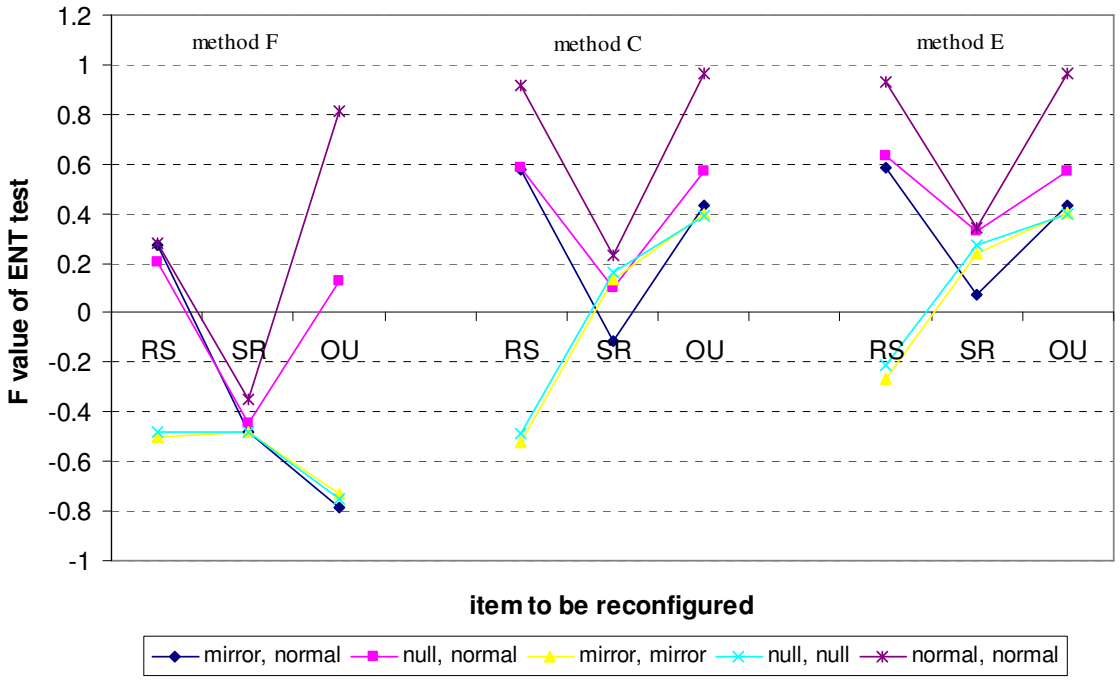
reconfigured every T time step (T=1 for method C, T=3 for method E) while the remaining items are kept at a fixed, preset configuration.

Aside from providing an item-based analysis of CoCA PRNG, another motivation of performing such experiments is to find a *default configuration* that will strengthen the performance of CoCA following Approach 4. Later, it will be shown that some cells should be kept at a fixed configuration and the other cells should be reconfigured at run-time so as to improve the performance of CoCA PRNG. Therefore, a fixed *default configuration* for some cells is required to achieve the maximum performance of CoCA. The *default configuration* considers the best choices for each item, as decided by these experiments.



Note: F value is calculated using the equation shown in Section 2.2. If the entropy value of a CoCA PRNG is less than 7, then F would most likely have a negative value.

Figure 2 ENT test results of CoCA PRNGs (Approach 4) under fixed RS



Note: *F* value is calculated using the equation shown in Section 2.2. If the entropy value of a CoCA PRNG is less than 7, then *F* would most likely have a negative value.

Figure 3 ENT test results of CoCA PRNGs (Approach 4) under fixed BC

Analysis of Figure 2. The performance of CoCA PRNG reconfiguring item BC is dependent on the transition rule that is used. Looking at the results from method C and method E, if the transition rule is fixed at either rule 105 or rule 30, CoCA PRNG reconfiguring item BC can give better ENT performance than if either rule 165 and rule 195 is used. This shows that the selection of transition rule function is crucial in designing PRNGs, as empirically proven in the past works. Particularly in CoCA PRNGs, this suggests that rule functions that are selected are closely related with the boundary conditions of the left and right neighbors. Recall that transition rules are functions of either 1) left neighbor and right neighbor, 2) left neighbor and the cell itself, 3) right neighbor and the cell itself, or 4) left, right neighbor and the cell itself. In [4], some of the rules are summarized using Boolean functions. It can be observed that rules 105 and 30

use left, right and the cell itself in the next state transition, whereas, rules 165 and 195 use left and right neighbor only.

Another important observation is that, CoCA reconfiguring item SR in method C and method E set of results are likewise affected by the selection of the fixed transition rule. Going back to the definition of SR item, status S refers to the updating of the cell, while reference R is related with dynamic changing of neighborhood. For some reasons, the rules 105 and 30 in CoCA reconfiguring item SR are negatively affected, if compared with CoCA reconfiguring item BC. On the other hand, rules 165 and 195 in CoCA reconfiguring item SR are improved. This shows that rule selection is also a very important factor in designing good CoCA PRNGs.

If item OU is to be reconfigured, the ENT results under different transition rules are somehow good and close with each other as compared to the ENT results of items BC and SR being reconfigured, which are greatly affected fixed rule used. This is because item OU is basically a sampling method which is not directly related with the transition rule functions of the CoCA.

Overall, if a cell is to be fixed at a certain rule, the best choice is *rule 105* as exemplified by the set of results from method F, C, and E.

Analysis of Figure 3. In Figure 3, the results clearly showed that mirror and null are not good choices for boundary conditions. Moreover, RS is the most affected item if boundary conditions are fixed at different choices of BC, as shown in method C and method E sets. Considering CoCA reconfiguring item SR, the ENT performances under different boundary conditions are comparable, but somehow inferior compared to CoCA

reconfiguring either item RS or OU. Like in Figure 2, CoCA reconfiguring item OU gives good performance regardless of the fixed boundary condition used.

Overall, *normal* boundary conditions for both left and right neighbors are the best choices for item BC.

Summary of Figure 2 and Figure 3 analysis. Based on the experiment results shown in Figures 2 and 3, in order to get good quality random numbers, the CoCA items should be set at the best choices. Thus, the *default configuration* is described as follows: RS must be rule 105, BC must use the normal boundary condition, some cells must be used as output cells for item OU, and S and R (in SR) must be both equal to 1, which means that cells must be updated and referenced every time step. This idea brings us back to the basics of CA or PCA PRNGs which uses certain sampling method, e.g., cell spacing.

Table 1 ENT results (S=300) of CoCA Approach 4

CA PRNG		**chi-square**	**ent**	**1-scc**
Fixed CA		0.76	7.981102	0.990827
Method F	CoCA - item BC	0	4.74076	0.861363
	CoCA - item RS	0	6.987152	0.941298
	CoCA - item SR	0	4.978203	0.851827
	CoCA - item OU	0.56	7.974581	0.991619
Method C	CoCA - item BC	0	7.918321	0.991268
	CoCA - item RS	0.91	7.980783	0.991814
	CoCA - item SR	0	6.120713	0.936092
	CoCA - item OU	0.94	7.981306	0.992248
Method E	CoCA - item BC	0	7.764403	0.988617
	CoCA - item RS	0.851667	7.980282	0.991679
	CoCA - item SR	0	7.191303	0.959354
	CoCA - item OU	0.938333	7.981544	0.991534

Note: CoCA item – x means that only item x is reconfigured.

Table 1 shows a detailed ENT result comparison of CoCA PRNG (Approach 4). Each CoCA item is reconfigured at run-time while other items are fixed at the *default configuration*. As mentioned previously, in F configuration method, configurations are dynamically changed not with time but with initial seed. The total number of initial seeds

to be tested is 300. Hence, for F configuration method, 300 different configurations were tested.

Figures 2 and 3, and Table 1 clearly showed that configuration method significantly affects the randomness of CoCA PRNGs. Method F gives the worst result. It is expected because the initial configurations are randomly chosen from good and not so good choices for each item, e.g., BC may be null or mirror, and RS may be rules 165 or 195. The results in the table indicate the average performance of the 300 different configurations. According to the initial experiments performed, items SR and BC should always be kept at S=1, R=1, and BC=normal. Thus, if items SR and BC do not follow these requirements, it will result in poor randomness quality.

Another important observation is that reconfiguring CoCA items dynamically, i.e. at every *T* discrete time step as in configuration methods C and E, improves CA PRNG performance. Although, the randomness results of configuring each of the items BC and RS are still not good enough. Between methods C and E, using *method C* in CoCA PRNG gives better randomness quality. This ENT result will be verified later with the Diehard test after the evolution process.

3.2.2. CoCA Approach 1 and Approach 2 at a glance

From the previous section, a default configuration has been construed. In this section, a similar experimental setup is applied where each item is reconfigured while fixing the others under the default configuration. In this case, a 9-cell control CA is used for *all* the cells, thus, leading to a CoCA PRNG following Approach 2. To assume CoCA PRNG Approach 1, an experiment involves all the items being reconfigured simultaneously, i.e. every T time step (*T*=1 for method C, *T*=3 for method E).

Table 2 shows the results of comparing CoCA Approach 1 and Approach 2. Apparently, a uniform cell configuration, regardless of item dependency, is very much inferior compared to a fixed CA. If the entropy value of a PRNG has not even reached 7 (i.e. *ent* < 7), it means that it has a very poor randomness quality. Thus, in this work, CoCA Approach 3 and 4 are the focus of a more systematic search for good CoCA PRNGs.

Table 2 ENT results (S=300) of CoCA Approach 1 and 2

CA PRNG		chi-square	ent	1-scc
Fixed CA		0.76	7.981102	0.990827
Method F	CoCA - item BC	0	4.746562	0.849972
	CoCA - item RS	0	4.563677	0.832516
	CoCA - item SR	0	4.552444	0.848316
	CoCA - item OU	0	3.790048	0.763356
	CoCA - item ALL*	0	3.805767	0.758685
Method C	CoCA - item BC	0	2.654191	0.227827
	CoCA - item RS	0	2.173383	0.15109
	CoCA - item SR	0	5.83183	0.854459
	CoCA - item OU	0	5.561192	0.85204
	CoCA - item ALL*	0	2.507474	0.129482
Method E	CoCA - item BC	0	3.78886	0.276986
	CoCA - item RS	0	2.288893	0.111186
	CoCA - item SR	0	6.268899	0.866066
	CoCA - item OU	0	5.961369	0.858318
	CoCA - item ALL*	0	2.838053	0.131395

* - CoCA item ALL is following CoCA Approach 1

3.2.3. CoCA Approach 3 and Approach 4 by varying the number of configurable cells

It has been found that the number of controllable cells in CCA has a significant effect upon the randomness of a PRNG [4]-[5]. The objective of the next experiment is to test the effect of varying the number of configurable cells on the randomness of CoCA PRNGs. In essence, the number of configurable cells differentiates Approach 3 from Approach 4 in terms of item-dependency. In this work, *non-configurable cells* are cells which assume fixed preset configuration at all time steps. On the other hand,

configurable cells are those cells which are configurable in some or all items during every T time step. In a CoCA, if some cells are configurable and others are not, then, it means that some items are reconfigured every T time step and others are not. This description falls into CoCA Approach 4 where reconfiguration is item-dependent, i.e. not all items are configured at the same time. On the other hand, if all the cells in CoCA are configurable cells and the items are reconfigured at the same time every T time step, then, it falls into CoCA Approach 3, where all items of all the cells are reconfigured every T time step.

Table 3 ENT results of CoCA with 10 out of 50 configurable cells

CA PRNG		chi-square	ent	1-scc
Fixed CA		0.865	7.980588	0.992206
Method C	CoCA - item BC	0.47	7.977803	0.991389
	CoCA - item RS	0.935	7.98128	0.991699
	CoCA - item SR	0.355	7.975967	0.991452
	CoCA - item OU	0.945	7.98159	0.992734

Tables 3 and 4 are two sets of experiments which differ by the number of cells to be reconfigured. In Table 3, 10 out of 50 cells are configured every time step. In Table 4, 5 out of 50 cells are configurable cells. The purpose of finding some *default configuration* in the previous section comes in here. If a cell is non-configurable, it will assume the *default configuration*. In this way, the maximum performance of CoCA is most likely ensured because the configuration of non-configurable cells will not deteriorate nor affect to some extent the performance of the CoCA PRNG.

As an initial experiment, a simple function of *modulus* is used to select the cells to be reconfigured. Take for example *modulo-n*. A cell x_i is selected as configurable if its index i is divisible by n, i.e., i modulo n equals 0. Method C is tried and compared with Fixed CA. 300 initial seeds are tested in both sets of experiment.

Table 4 ENT results of CoCA with 5 out of 50 configurable cells

CA PRNG		chi-square	ent	1-scc
Fixed CA		0.765	7.979021	0.991184
Method C	CoCA - item BC	0.83	7.980704	0.992741
	CoCA - item RS	0.96	7.981605	0.992832
	CoCA - item SR	0.915	7.981374	0.991654
	CoCA - item OU	0.87	7.981137	0.990856

As shown in Table 3, if the number of cells to be configured is decreased, i.e., 10 configurable cells, the performance of CoCA PRNG is generally improved. But comparing individual results of all the CoCA PRNGs in Table 3, CoCA reconfiguring item BC and SR give low ENT results. Table 4 shows that further reducing the number of cells to be reconfigured further improves the performance of CoCA PRNGs specifically CoCA reconfiguring items BC and SR. This suggests that CoCA reconfiguring items BC and SR can give good performance as long as the number of configurable cells is properly aligned. It is worth noting that if the number of configurable cells of CoCA is 0, then it is equivalent to a Fixed CA.

Table 5 shows the results when the number of configurable cells of CoCA is varied and all the four items are configured at the same time.

Table 5 ENT results of CoCA with varying number of configurable cells

CA PRNG		chi-square	ent	1-scc
Fixed CA		0. 851667	7.980924	0.991852
CoCA (Method C)	5 cells	0.921667	7.981446	0.991633
	10 cells	0.675000	7.978407	0.992127
	50 cells*	0.031667	7.974198	0.991896

Note: x cells means that x cells are configurable in CoCA.
* - configuring 50 cells is equivalent to CoCA Approach 3

From the results of Tables 3 to 5, it can be concluded that CoCA can outperform fixed CA by configuring all the four items provided the following restrictions are met. Firstly, items RS and OU are to be configured for all cells. Secondly, items SR and BC

are to be configured for some cells only. Lastly, cells must be reconfigured at every T time step as in methods C and E. After the evolution process in the next section, a conclusion on the best configuration method, whether method C or E, will be reached. Conclusively, CoCA Approach 4 is the best approach to be used in random number generation. Thus, from here onwards, *CoCA Approach 4* will be the focus of searching good quality PRNGs by genetic algorithms.

4 Evolutionary Algorithm

A genetic algorithm (GA) is an iterative procedure that autonomously searches for possible solutions to a given problem with countless number of solutions. Each solution is represented in an encoded form, known as *chromosome*. GA involves a constant-size population which evolves at every evolutionary step, also known as *generation*. It starts with an initial population that is randomly or heuristically generated. During every generation, each individual is evaluated by a certain *fitness function*. The population in the next generation is then generated by selecting individuals from the previous generation according to their fitness values and transforming these individuals via genetic operators like *mutation* (randomly altering one or more values in a string of chromosome) and *crossover* (combining two chromosomes to form a novel chromosome). By repeating the procedure, an acceptable solution with the highest fitness value may be found.

Traditionally, CA PRNGs are handcrafted. The design process is however, time-consuming and inefficient. Accordingly, researchers began to use GA to evolve CA PRNGs. And like the other research on CA PRNGs [4]-[6], [17], CoCA also makes use

of genetic algorithms to search for good CA PRNG structures. The pseudo code of the genetic algorithm is shown in the Appendix.

The main objective of applying GA in CoCA PRNGs is to search for a CoCA PRNG design that is comparable with past work on CA PRNGs in terms of performance. As seen in Section 3, the randomness of a CoCA PRNG is dependent on the number of configurable cells and perhaps, the positioning of these configurable cells. It is the GA's role to find the optimal number of configurable cells in CoCA.

In Tomassini et al.'s work [17] and Guan et al.'s work [4]-[6], they used ENT test as a fitness measure. In this work, the function F described in Section 2.2 is used as the fitness function. After obtaining the best chromosomes, they are tested under the Diehard test, which, as mentioned earlier, is the foremost gauge of a good pseudorandom number generator.

The input of an evolution process is randomly generated by a C++ function. Population size is set at 16. The stopping criterion is the maximum stagnation steps. If the best chromosome in each population keeps unchanged for 200 steps continuously, the evolution process stops [6]. The 2-point crossover rate is set at 1.0. The bit mutation rate is set at 0.1. During reproduction, half of the better-performing parents and child chromosomes are copied into the next generation.

Based on the level of configurability (i.e. items to be configured) of a CoCA, three different chromosome structures are to be evolved. The three chromosome structures are designed based on the experimental results shown in the preliminary experiments section (Section 3.2). Table 6 shows the summary of the three chromosome structures.

Table 6 Summary of the three chromosome structures of GA used in CoCA PRNGs

Chromosome Structure	Length	Illustration	Legend
1	L		**0** – not configurable **1** – all items configurable
2	L*2		**00** – not configurable **01** – RS & OU configurable **10** – all items configurable
3	L*2		**00** – not configurable **01** – RS & OU configurable **10** – RS, OU, & BC configurable **11** – all items configurable

The first chromosome structure deals with the extreme case where a cell is to be determined as configurable or not. For a 1-d CoCA with L number of cells, the length of the chromosome is L bits, where, '1' indicates that a cell is configurable and '0' otherwise.

The second chromosome structure redefines the first chromosome structure such that its length is L*2 bits. A pair of bits (which allows maximum of four categories) corresponds to each cell. Three categories are designed: "not configurable", "all items configurable", and "items RS and OU configurable".

The third chromosome structure has a level of configurability which is the most flexible. The chromosome length is also L*2 and a pair of bits corresponds to a cell. But this time, the categories are more specific, i.e. “00 for not configurable”, “01 for configurable RS and OU”, “10 for configurable RS, OU, and BC”, and “11 for configurable RS, OU, BC, and SR”.

4.1 Evolving chromosome structure 1 and comparison of configuration methods C and E

In the following experiments, the configurable cells of a 50-cell CoCA PRNG Approach 4 are evolved following the aforementioned genetic algorithm via chromosome structure1. After the evolution process, the top 5 chromosomes of the CoCA are tested with five initial seeds under the Diehard test. Apart from evolving the best chromosome structures, another objective of this experiment is to compare the performance of using C configuration method and E configuration method. Experimental setup is the same as in the previous section.

Table 7 shows the average number of Diehard tests that a CoCA PRNG can pass. The results show that indeed, CoCA PRNG is better than a fixed configuration CA. Also, it can be concluded that CoCA PRNG following configuration method C (abbreviated as CoCA-C) is better than CoCA PRNG following configuration method E (abbreviated as CoCA-E) at T=3 for two reasons. First, the maximum average number of Diehard tests that CoCA-C PRNG can pass is greater than that of CoCA-E PRNG. Also, most of the chromosomes of CoCA-C can pass more number of Diehard tests, which means that CoCA-C PRNG has more chances of passing Diehard test at different number and position of configurable cells. This suggests that in method C, the reconfiguration process is more arbitrary and random as it is changed every T=1 time step.

Table 7 Diehard Results of CoCA at L=50, S=5

CA PRNG		No. of tests passed in Diehard					
		seed1	**seed2**	**seed3**	**seed4**	**seed5**	**average**
Fixed CA		10	15	5	6	16	10.4
CoCA Method E T=3	chrom1	13	14	13	10	14	12.8
	chrom2	16	15	12	12	15	14.0
	chrom3	16	16	14	12	15	14.6
	chrom4	15	14	13	12	13	13.4
	chrom5	17	14	17	16	15	15.8
CoCA Method C	chrom1	16	17	15	15	17	16.0
	chrom2	16	15	14	13	17	15.0
	chrom3	16	15	15	16	15	15.4
	chrom4	16	17	16	16	17	16.4
	chrom5	16	16	17	17	16	16.4

With the results shown in Table 7, CoCA-C PRNG is the focus of the next experiments. It is further improved by studying and designing the external structures, i.e. the length of CoCA.

4.2. Searching for good CoCA external structure

The external structure, specifically the number of cells of a CA PRNG significantly affects the randomness quality. In the following experiment, this important property of CA PRNGs is explored by finding the optimal number of cells of a CoCA PRNG that can pass Diehard test. Two sets of experiments are conducted. One uses rule 30 as transition rule function of the uniform control CA. While the other uses rule 90. The purpose of involving two different transition rule functions is to compare the effect of the control CA in CoCA random number generation.

Again, genetic algorithm is applied to evolve the number and position of configurable cells of CoCA PRNGs at *different lengths*. The evolution and CA transition parameters and conditions are the same as in the previous experiment where chromosome structure 1

is evolved. After the long evolution process, the top 5 chromosomes of each CoCA-C PRNG (under different lengths) are subjected to Diehard test.

A summary of results is shown in Table 8. In each entry (length of CoCA), the chromosome that can give the best performance in under the Diehard test is selected and recorded in the table.

Table 8 Diehard results of L-length CoCA-C PRNG

CoCA-C Length	**Average no. of Diehard tests passed**	
	Control CA Rule 30	**Control CA Rule 90**
50	16.4	16
49	12	14.8
41	12.8	11.2
35	14	14.8
33	15.4	15.2
31	14.8	15.4
29	13.8	14.4
23	13.2	13.6
19	12	12.2
15	11.6	10.2

An interesting observation from Table 8 is the trend of randomness quality as the number of cells of CoCA PRNG decreases. There is a certain optimal length (L=31,33) of CoCA that can fairly pass Diehard test. Figure 4 shows a graph of comparing the performance of CoCA PRNG with different control CA rule function.

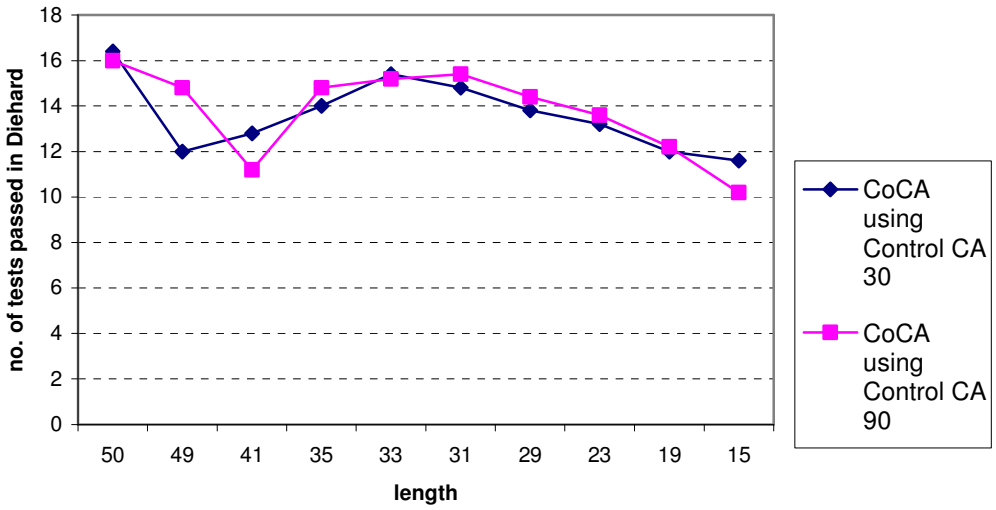


Figure 4 Diehard test performance of varying CoCA length using different rule function of Control CA

It can be seen from Figure 4 that control CA rule function somehow affects the performance of CoCA PRNG based on different lengths. Another important observation is that using rule 90 is better than using rule 30 as transition rule function for the control CA. This suggests that the CoCA, which do not use rule 90, avoids correlations between the control CA and the CoCA.

4.3. Evolving chromosome structure 2 and 3

In the previous subsections, the genetic algorithm evolves the number and position of configurable cells where the length of chromosome is equal to the number of CoCA cells. If a cell is configurable, then, all the four items are configured every time step. Otherwise, the cell is non-configurable and uses the *default configuration*. Like in the previous subsections, CoCA following Approach 4 and configuration method C is assumed in the evolution process of chromosome structures 2 and 3 shown in Table 6. Chromosome structures 2 and 3 introduce a more flexible way of assigning items to be configured for each cell. In this way, it is probable that the level of configurability will

bring in better randomness quality to CoCA PRNGs since the CoCA cells have more choices to choose from.

From the genetic algorithm point of view, the search space (number of possible solutions) increases with the length of the chromosome. It is therefore important to note that a longer chromosome structure may also be an impediment to a successful search of good chromosome structures. On account of a larger search space, longer chromosome structure may require longer evolution time as well. In this work, only 2 bits per cell is assigned in chromosome structures 2 and 3. Also, the optimal number of cells, which is L=31, is experimented on. Thus, the length of both chromosome structures 2 and 3 is L*2 = 62 bits.

Table 9 shows the results of comparing the three chromosome structures of CoCA PRNG, all following Approach 4 and configuration method C and using rule 90 for the uniform control CA. As in the previous experiments, after the evolution process, the best chromosomes are subjected to Diehard test.

Table 9 Diehard test result of 31-cell CoCA-C Method E

Chromosome structure	**Average no. of tests passed in Diehard**
1 (31 bits)	15
2 (62 bits)	16
3 (62 bits)	17

It can be concluded from Table 9 that the level of configurablity of a CoCA PRNG affects the randomness quality based on Diehard test. Chromosome structure 3 is the most flexible and configurable structure, therefore, it gives the best Diehard test result.

5 Discussion on Design Considerations for Future Works

The design of CoCA PRNGs involves several considerations. As seen in the preliminary experiments, the four items are closely interrelated with each other. For

instance, the choice of rules to be fixed affects the randomness of CoCA reconfiguring BC and SR. It was seen that rules 165 and 195 do not have good effect on reconfiguring other items. Thus, more variations to CoCA PRNGs may be experimented by using different choices of rule other than rules 165 and 195.

Another important consideration in designing CoCA PRNGs is the item-dependency of each cell. As can be observed from the experiments conducted in this work, item-dependency is also influenced by configuration method C. This means that items RS and OU of *all the cells* which are assigned '**01**' in chromosome structure 3 are updated at the same time every time step. Similarly, items RS, OU, and BC of *all the cells* which are assigned '**10**' in chromosome structure 3 are updated at the same time. More arbitrary configurations may be introduced to CoCA PRNGs by making these cells differ at the time of reconfiguration, i.e. involving configuration method E in some cells. However, as mentioned previously, longer chromosome length may introduce longer evolution time.

Finally, control CA is another area of CoCA PRNG design that can be further investigated. In this work, two rule functions for control CA are tried. Also, for Approach 3 and 4, the *L* 9-cell CA, which is one-dimensional in nature, is utilized. Further search for good CoCA PRNGs may be done by making use of other transition rule functions in the control CA rather than transition rules 30 and 90. Moreover, a 2-d CA variation [6] may be used as control CA. The 2-d CA variation may initiate more random configurations to CoCA, thereby, improving the randomness quality of CoCA PRNGs.

6 Conclusion

As a PRNG, CoCA can be a suitable candidate. A 31-cell CoCA following Approach 4 and Method C can pass 17 tests in Diehard. In this paper, CoCA is introduced for the purpose of designing a random number generator, whose primary features are flexibility and programmability in a number of aspects. CoCA PRNG has an advantage over other CA PRNGs as there can be an indefinite number of designs to be drawn out from the concept. This way, researchers can explore on different variations of a CA PRNG with a guarantee of good randomness quality. In this paper, CoCA PRNG was compared with a fixed configuration PRNG. Because CoCA PRNG is essentially a generalized form of a CA PRNG, it can not be simply compared with other CA PRNGs like PCA, CCA, SPCA, or 2-d CA. Moreover, CoCA may be combined with SPCA and 2-d CA properties for further improvement of CoCA PRNGs. For information purpose, a summary of all existing CA PRNG designs is presented in Table 10.

It is worth noting that apart from random number generation, CoCA can be applied in a wide area of applications where normal CA is applied, e.g., modeling and simulation for complex systems. CoCA is a general model that covers all aspects of the CA framework. An example would be to combine CoCA with CAM (Cellular Automata Machine, MIT) [13], a programmable chip for CA, to be used in modeling complex systems.

Table 10 Past works on CA PRNG and its performance in Diehard test

CA PRNG design	No. of cells	No. of tests passed in Diehard	Remarks
PCA [14]	50	17	Programmable in rule selection (RS)
CCA0/CCA2 [5]	50	18	Programmable in status and RS
SPCA 150/105 [3]	21	18	Programmable in RS
CoCA	31	17	Programmable in many aspects

References

[1] D.R. Chowdhury, I.S. Gupta, and P.P. Chaudhuri, A class of two-dimensional cellular automata and applications in random pattern testing. J. Electrical Testing: Theory and Applications 5, 65(1994).

[2] ENT Test, http://www.fourmilab.ch/random/.

[3] S.U. Guan S.U. and S.K. Tan, Pseudorandom number generator - The self programmable cellular automata. Knowledge-based Intelligent Information and Engineering Systems, PT 1, Proceedings (2003).

[4] S.U. Guan, and S. Zhang, A family of controllable cellular automata for pseudorandom number generation, International Journal of Modern Physic C, Vol. 13, No. 8 (2002).

[5] S.U. Guan, and S. Zhang, An evolutionary approach to the design of controllable cellular automata structure for random number generation. IEEE Trans. Evolutionary Computation 7(1): 23-36 (2003).

[6] S.U. Guan, and S. Zhang, and M.T. Quieta, 2-d CA Variation with Asymmetric-Neighborship for Pseudorandom Number Generation, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 23-3 (2004).

[7] P.D. Hortensius, R.D. Mcleod, and H.C. Card, Parallel Random Number Generation for VLSI System Using Cellular Automata. IEEE Trans. Comput. 38, 1466 (1989).

[8] 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 Transactions on Computer-Aided Design 8(8) (1989).

[9] J.C. Isaacs, R.K. Watkins, S.Y. Foo, Cellular automata PRNG: maximal performance and minimal space FPGA implementations. Engineering Applications of Artificial Intelligence 16 (5-6): 491-499 (2003).

[10] G. Marsaglia, "Diehard", http://stat.fsu.edu/~geo/diehard.html, (1998).

[11] S. Nandi, B.K. Kar, and P.Pal Chaudhuri, Theory and Applications of Cellular Automata in Cryptography. IEEE Trans. Comput. 43, 1346 (1994).

[12] W. Pries, A. Thanailakis, and H. Card, Group Properties of Cellular Automata and VLSI Applications. IEEE Trans. Comput. C-35 (12) (1986).

[13] P. Sarkar, ACM Comput. Surveys 32(1), 80(2000)

[14] M. Sipper and M. Tomassini, Generating parallel random number generators by cellular programming. Int. J. Mod. Phys. 7(2), 181 (1996).

[15] D. Talia and P. Sloot, Cellular Automata: Promise and Prospects in Computational Science. Future Generation Computer Systems 16 (1999). V-vii.

[16] T. Toffoli and N. Margolus, Cellular Automata Machines, A new environment for modeling. MIT Press Series in Scientific Compuation (1985).

[17] M. Tomassini, M. Sipper, and M. Perrenoud, On the generation of high-quality random numbers by two-dimensional cellular automata. IEEE Trans. Comput. 49, 1146(2000).

[18] M. Tomassini., M. Sipper, M. Zolla, and M. Perrenoud, Generating high-quality random numbers in parallel by cellular automata. Future Generation Computer Systems 16, 291 (1999).

[19] S. Wolfram, Cryptography with Cellular Automata, Advances in Cryptography: Proceeding of CRTPTO 85, Lecture Notes in Computer Science, Vol. 218 (1985), pp. 429-432.

[20] S. Wolfram, Theory and Applications of Cellular Automata: Including Selected Papers 1983-1986 (World Scientific Publishing Co., Inc., River Edge, N.J., 1986).

[21] J. Von Neumann, "The general and logical theory of automata", J. von Neumann Collected Works, ed. A. Taub.

Appendix - Genetic Algorithm

The evolutionary algorithm is shown below. P is the total number of chromosomes in a population. T is the total number of CA running cycles. Q is the percentage of crossover-generated chromosomes in the next generation. EL is the length of the chromosome and R is the percentage of EL that has been altered by mutation.

Initialization

randomly generate 100 initial seeds for main CA (CoCA), and control CA (if needed);

Evolution

while (stopping condition is not true) **do**

Fitness calculation

for each chromosome i (i=1 to P)

for each CA initial seed j (j=1 to 100) **do in parallel**

initialize main CA and control CA;

run CA for T time steps;

calculate the fitness F_{ij} of main CA;

end parallel for

Calculate average fitness value, F_i of the 100 initial seeds for each chromosome

end

Offspring generation

scale the fitness value using the windowing method

- roulette-wheel select from parent chromosomes, generate (P* Q%) child chromosomes by crossover;
- generate P child chromosomes (EL*R%) by mutation

Selection

Copy the first half of parent chromosomes and first half of child chromosomes to the next generation.

end while