

Evolutionary Design of Rule Changing Cellular Automata

Hitoshi Kanoh¹ and Yun Wu²

¹ Institute of Information Sciences and Electronics, University of Tsukuba,
Tsukuba Ibaraki 305-8573, Japan
kanoh@is.tsukuba.ac.jp

² Graduate School of Systems and Information Engineering, University of Tsukuba,
Tsukuba Ibaraki 305-8573, Japan
bregude@kslab.is.tsukuba.ac.jp

Abstract. The difficulty of designing cellular automata's transition rules to perform a particular problem has severely limited their applications. In this paper we propose a new programming method of cellular computers using genetic algorithms. We consider a pair of rules and the number of rule iterations as a step in the computer program. The present method is meant to reduce the complexity of a given problem by dividing the problem into smaller ones and assigning a distinct rule to each. Experimental results using density classification and synchronization problems prove that our method is more efficient than a conventional one.

1 Introduction

Recently, evolutionary computations by parallel computers have gained attention as a method of designing complex systems [1]. In particular, parallel computers based on cellular automata (CAs [2]), meaning cellular computers, have the advantages of vastly parallel, highly local connections and simple processors, and have attracted increased research interest [3]. However, the difficulty of designing CAs' transition rules to perform a particular task has severely limited their applications [4].

The evolutionary design of CA rules has been studied by the EVCA group [5] in detail. A genetic algorithm (GA) was used to evolve CAs. In their study, a CA performed computation means that the input to the computation is encoded as the initial states of cells, the output is decoded from the configuration reached at some later time step, and the intermediate steps that transform the input to the output are taken as the steps in the computation. Sipper [6] has studied a cellular programming algorithm for non-uniform CAs, in which each cell may contain a different rule. In that study, programming means the coevolution of neighboring, non-uniform CAs' rules with parallel genetic operations.

In this paper we propose a new programming method of cellular computers using genetic algorithms. We consider a pair of rules and the number of rule iterations as a step in the computer program, whereas the EVCA group considers an intermediate

step of transformation as a step in the computation. The present method is meant to reduce the complexity of a given problem by dividing the problem into smaller ones and assigning a distinct rule to each one. Experimental results using density classification and synchronization problems prove that our method is more efficient than a conventional method.

2 Present Status of Research

2.1 Cellular Automata

In this paper we address one-dimensional CAs that consist of a one-dimensional lattice of N cells. Each cell can take one of k possible states. The state of each cell at a given time depends only on its own state at the previous time step and the state of its nearby neighbors at the previous time step according to a transition rule R . A neighborhood consists of a cell and its r neighbors on either side. A major factor in CAs is how far one cell is from another. The CA rules can be expressed as a rule table that lists each possible neighborhood with its output bit, that is, the update value of the central cell of the neighborhood. Figure 1 shows an example of a rule table when $k=2$ and $r=1$. We regard the output bit “11101000” as a binary number. The number is converted to a decimal, which is 232, and we will denote the rule in Fig. 1 as rule 232. Here we describe CAs with a periodic boundary condition.

The behavior of one-dimensional CAs is usually displayed by space-time diagrams in which the horizontal axis depicts the configuration at a certain time t and the vertical axis depicts successive time steps. The term “configuration” refers to the collection of local states over the entire lattice, and $S(t)$ denotes a configuration at the time t .

Neighborhood:	111 110 101 100 011 010 001 000
Output bit:	1 1 1 0 1 0 0 0

Fig. 1. An example of a rule table ($k=2$, $r=1$).

2.2 Computational Tasks for CAs

We used the density classification and the synchronization tasks as benchmark problems [4, 7]. The goal for the density classification task is to find a transition rule that decides whether or not the initial configuration $S(0)$ contains a majority of 1s. Here $\rho(t)$ denotes the density of 1s in the configuration at the time t . If $\rho(0) > 0.5$, then within M time steps the CA should go to the fixed-point configuration of all 1s ($\rho(t \geq M) = 1$); otherwise, within M time steps it should produce the fixed-point configuration of all 0s ($\rho(t \geq M) = 0$). The value of constant M depends on the task.

The second task is one of synchronization: given any initial configuration $S(0)$, the CA should reach a final configuration, within M time steps, that oscillates between all 0s and all 1s in successive time steps.

2.3 Previous Work

The evolutionary design of CA rules has been studied by the EVCA group [3, 4] in detail. A genetic algorithm (GA) was used to evolve CAs for the two computational tasks. That GA was shown to have discovered rules that gave rise to sophisticated emergent computational strategies. Sipper [6] has studied a cellular programming algorithm for 2-state non-uniform CAs, in which each cell may contain a different rule. The evolution of rules is here performed by applying crossover and mutation. He showed that this method is better than uniform (ordinary) CAs with a standard GA for the two tasks.

Meanwhile, Land and Belew [8] proved that the perfect two-state rule for performing the density classification task does not exist. However, Fuks [9] showed that a pair of human written rules performs the task perfectly. The first rule is iterated t_1 times, and the resulting configuration of CAs is processed by another rule iterated t_2 times. Fuks points out that this could be accomplished as an “assembly line” process.

Other researchers [10] have developed a real-world application for modeling virtual cities that uses rule-changing, two-dimensional CAs to lay out the buildings and a GA to produce the time series of changes in the cities. The GA searches the sequence of transition rules to generate the virtual city required by users. This may be the only application using rule changing CAs.

3 Proposed Method

3.1 Computation Based on Rule Changing CAs

In this paper, a CA in which an applied rule changes with time is called a rule changing CA, and a pair of rules and the number of rule iterations can be considered as a step in the computer program. The computation based on the rule changing CA can thus operate as follows:

- Step 1: The input to the computation is encoded as the initial configuration $S(0)$.
- Step 2: Apply rule R_1 to $S(0)$ M_1 times;; apply rule R_n to $S(M_1+\dots+M_{n-1})$ M_n times.
- Step 3: The output is decoded from the final configuration $S(M_1+\dots+M_n)$.

In this case, n is a parameter that depends on the task, and rule R_i and the number of rule iterations M_i ($i=1, \dots, n$) can be obtained by the evolutionary algorithm described in the next section. The present method is meant to reduce the complexity of a given task by dividing the task into n smaller tasks and assigning (R_i, M_i) to the i -th task.

3.2 Evolutionary Design

Each chromosome in the population represents a candidate set of R_i and M_i as shown in Fig. 2. The algorithm of the proposed method is shown in Fig. 3.

$$\boxed{R_1 | M_1 | \dots \dots \dots | R_n | M_n}$$

Fig. 2. Chromosome of the proposed method (R_i : rule, M_i : the number of rule iteration).

```

procedure main
  initialize and evaluate population  $P(0)$ ;
  for  $g = 1$  to the upper bound of generations {
    apply genetic operators to  $P(g)$  and create a temporary
    population  $P'(g)$ ;
    evaluate  $P'(g)$ ;
    get a new population  $P(g+1)$  by using  $P(g)$  and  $P'(g)$ ;
  }
procedure evaluate
  for  $i = 1$  to the population size {
    operate CA with the rules on the  $i$ -th individual;
    calculate fitness of the  $i$ -th individual;
  }

```

Fig. 3. Algorithm of the proposed method (procedure *main* and procedure *evaluate*).

3.3 Application

In this section we describe an application that uses the CA with two rules for density classification and synchronization tasks.

Chromosome Encoding In the case of the CA with two rules, a chromosome can generally be expressed by the left part in Fig. 4. The right part in Fig. 4 shows an example of the chromosome when $k=2$ and $r=1$. Each rule and the number of iterations are expressed by the output bit of the rule table and a nonnegative integer, respectively.

$$\boxed{R_1 | M_1 | R_2 | M_2} = \boxed{11101000 | 45 | 01001001 | 25}$$

Fig. 4. Example of the chromosome of the proposed method for the CA with two rules.

Fitness calculation The fitness of an individual in the population is the fraction of the N_{test} initial configurations in which the individual produced the correct final configurations. Here the initial configurations are uniformly distributed over $\rho(0) \in [0.0, 1.0]$.

Genetic operations First, generate N_{pop} individuals as an initial population. Then the following operations are repeated for N_{gen} generations.

- Step 1: Generate a new set of N_{test} initial configurations, and calculate the fitness on this set for each individual in the population.
- Step 2: The individuals are ranked in order of fitness, and the top N_{elit} elite individuals are copied to the next generation without modification.
- Step 3: The remaining ($N_{pop} - N_{elit}$) individuals for the next generation are formed by single-point crossovers between the individual randomly selected from N_{elit} elites and the individual selected from the whole population by roulette wheel selection.
- Step 4: The rules R_1 and R_2 on the offspring from crossover are each mutated at exactly two randomly chosen positions. The numbers of iterations M_1 and M_2 are mutated with probability 0.5 by substituting random numbers less than an upper bound.

4 Experiments

4.1 Experimental Methods

The experiments were conducted under the following conditions: the number of states $k=2$, the size of lattice $N=149$, and $M_1+M_2=149$ for the CA; the population size $N_{pop}=100$, the upper bound of generations $N_{gen}=100$, the number of elites $N_{elit}=20$, and the number of the initial configuration for test $N_{test}=100$ for the GA.

4.2 Density Classification Task ($r=1$)

Table 1 shows the best rules in the population at the last generation. The obtained rules agree with the human written rules shown by Fuks [9], which can perform this task perfectly.

Table 1. Rules obtained by the genetic algorithm for the density classification task ($r=1$, $k=2$, $N=149$).

	R_1	M_1	R_2	M_2
Experiment-1	184	124	232	25
Experiment-2	226	73	232	76

4.3 Density Classification Task ($r=3$)

Figure 5 shows the comparison of the proposed method with the conventional method that uses only one rule as the chromosome [5]. Each point in the figure is the average of ten trials. It is seen from Fig. 5 that the fitness of the former at the last generation (0.98) is higher than that of the latter (0.91). Table 2 shows the best rules obtained by the proposed method, and Fig. 6 shows examples of the space-time diagrams for these rules. The rules in Table 2 have been converted to hexadecimal.

Table 2. The best rules obtained by the genetic algorithm for the density classification task ($r=3$, $k=2$, $N=149$).

R_1	M_1	R_2	M_2
01000100111C0000 000004013F7FDFFB	101	97E6EFF6E8806448 4808406070040000	48

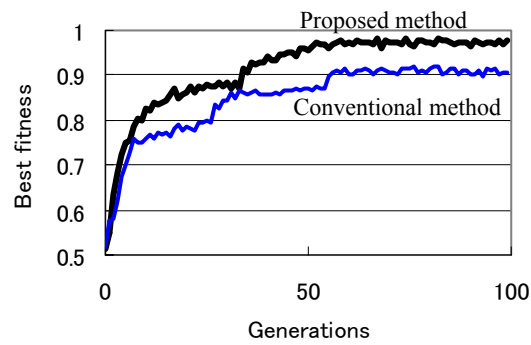


Fig. 5. The best fitness at each generation for density classification tasks ($r=3$, $k=2$, $N=149$).

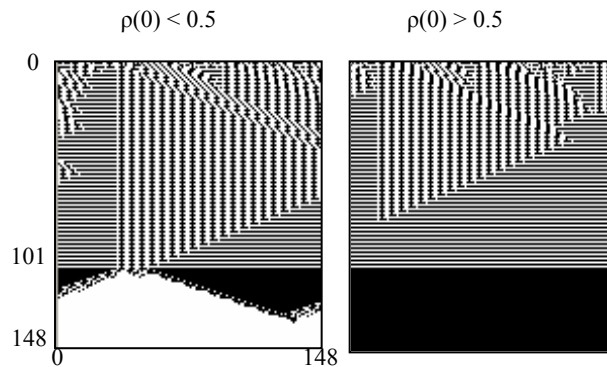


Fig. 6. Examples of space-time diagrams for a density classification task ($r=3$, $k=2$, $N=149$).

4.4 Synchronization Task ($r=3$)

Figure 7 shows the results of the same comparison as in Fig 5 for the synchronization task. The fitness of the proposed method reaches 1.0 by the 3rd generation, whereas the conventional method requires more than 20 generations.

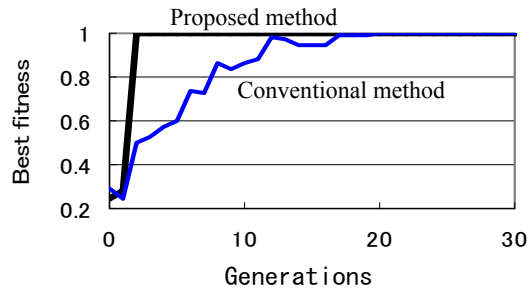


Fig. 7. The best fitness at each generation for the synchronization task ($r=3, k=2, N=149$).

5 Conclusions

In this paper we proposed a new programming method of cellular computers. The experiments were conducted using a rule changing CA with two rules. In a forthcoming study, the performance of a CA with more than two rules will be examined.

References

1. Alba, E. and Tomassini, M.: Parallelism and Evolutionary Algorithms. IEEE Transactions on Evolutionary Computation, Vol. 6, No. 5 (2002) 443-462.
2. Wolfram, S.: A New Kind of Science. Wolfram Media Inc. (2002).
3. Sipper, M.: The Emergence of Cellular Computing. IEEE Computer, Vol. 32, No. 7 (1999).
4. Mitchell, M., Crutchfield, J., and Das, R.: Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work. Proceedings of the First International Conference on Evolutionary Computation and Its Applications (1996).
5. Mitchell, M., Crutchfield, J. P., and Hraber, P.: Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments. Physica D 75 (1994) 361-391.
6. Sipper, M.: Evolving of Parallel Cellular Machines: The Cellular Programming Approach. Lecture Notes in Computer Science Vol. 1194. Springer-Verlag (1997).
7. Das, R., Crutchfield, L., Mitchell, M., and Hanson, J.: Evolving Globally Synchronized Cellular Automata. Proceedings of the 6-th ICGA (1995) 336-343.
8. Land, M. and Belew, R.: NO Two-State CA for Density Classification Exists. Physical Review Letters 74 (1995) 5148.
9. Fuks, H.: Solution of the Density Classification Problem with Two Cellular Automata Rules. Physical Review E, 55(3) (1997) 2081-2084.
10. Kato, N., Okuno, T., Suzuki, R., and Kanoh, H.: Modeling Virtual Cities Based on Interaction between Cells. IEEE International Conference on Systems, Man, and Cybernetics (2000) 143-148.