Discovery of Cellular Automata Rules Using Cases

Ken-ichi Maeda* and Chiaki Sakama †

Department of Computer and Communication Sciences
Wakayama University
† sakama@sys.wakayama-u.ac.jp

Abstract. *Cellular automata* (CAs) are used for modeling the problem of adaptation in natural and artificial systems, but it is hard to design CAs having desired behavior. To support the task of designing CAs, this paper proposes a method for automatic discovery of cellular automata rules (*CA-rules*). Given a sequence of CA configurations, we first collect cellular changes of states as *cases*. The collected cases are then classified using a *decision tree*, which is used for constructing CA-rules. Conditions for classifying cases in a decision tree are computed using *genetic programming*. We perform experiments using several types of CAs and verify that the proposed method successfully finds correct CA-rules.

1 Introduction

Cellular automata (CAs) [4] are discrete dynamical systems whose behavior is specified by the interaction of local cells. Because of their simple mathematical constructs and distinguished features, CAs have been used for modeling advanced computation such as massively parallel computers and evolutionary computation, and also used for simulating various complex systems in the real world. On the other hand, complex behavior of CAs is difficult to understand, which makes hard to design CAs having desired behavior. The task of designing CAs usually requires domain knowledge of a target problem and it is done by human experts manually and experientially. This task becomes harder as a target problem becomes complex, since there are a number of possible automata to specify behavior. The difficulty also comes from the feature of CAs such that a small change of local interaction would affect the global behavior of a CA, and the result of emergence depends on the initial configuration of cells.

To automate CA designing, we develop techniques for automatic discovery of CA-rules which reflect cellular changes in observed CA configurations. Reconstruction of CA-rules from input configurations is known as the *identification problem* [1]. However, we aim at not only reconstructing the original CA-rules but also discovering new CA-rules. Automatic discovery of CA-rules is also studied in the context of *density classification task* [3]. The objective of this task is to find a 1-dimensional 2-state CA that can classify the density of 1's in the initial

* Current address: Hitachi System& Services, Ltd. ke-maeda@hitachi-system.co.jp

configuration, which is different from our goal. Technically, our goal is achieved by the following steps. Given a sequence of CA configurations we first collect cellular changes of states as *cases*. The collected cases are then classified using a *decision tree* which is used for constructing CA-rules. Conditions for classifying cases in a decision tree are computed using *genetic programming*. We perform experiments using several types of CAs and verify that the proposed method not only reconstructs the original CA-rules, but also discovers new CA-rules.

The rest of this paper is organized as follows. Section 2 presents a brief introduction of cellular automata. Section 3 provides techniques for automatic discovery of CA-rules. Section 4 shows experimental results to verify the proposed method, and Section 5 summarizes the paper.

2 Cellular Automata

A *cellular automaton* (CA) consists of a regular grid of *cells*, each of which has a finite number of possible *states*. The state of each cell changes synchronously in discrete time steps according to local and identical transition rules (called *CA-rules*). The state of a cell in the next time step is determined by its current state and the states of its surrounding cells (called a *neighborhood* of a cell). The collection of all cellular states in the grid at some time step is called a *configuration*. A CA has an *n-dimensional* cellular space.

Consider a sequence of configurations $S_0, \ldots, S_n$ in which each configuration has a finite number of cells. Here, S_0 is the initial configuration of a CA and S_t $(0 \leq t \leq n)$ represents the configuration of the CA at a time step t. A configuration S_t consists of states of a cell $C^t_{x,y}$, where x and y represent the coordinates of the cell in the configuration (Figure 1).

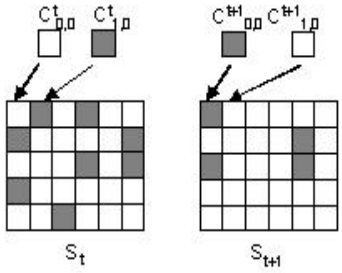

Fig. 1. Configurations of a CA

Given a sequence of configurations as an input, we want to output CA-rules which, applied to the initial configuration, reproduce the change of patterns of input configurations. We pursue the goal in the following steps.

1. Determine an appropriate neighborhood of a cell.
2. Collect cellular changes of states as *cases* from input configurations.
3. Construct a *decision tree* to classify cases and extract CA-rules.

In the next section, we explain techniques using 2-dimensional 2-state CAs, but the techniques are applied to m-dimensional n-state CAs in general.

3 Discovering CA-Rules

3.1 Collecting Cases

We first describe a method of collecting cases. A *case* is defined as a pair $(R^t_{x,y}, C^{t+1}_{x,y})$ where $R^t_{x,y}$ is a neighborhood of a cell $C^t_{x,y}$ at some time step t and $C^{t+1}_{x,y}$ is the state of the cell at the next time step $t+1$.[1] Cases are collected using the following procedure.

Procedure: *Collecting cases.*
Input : a sequence of configurations $S_0, \ldots, S_n$.
Output : a set **CASE** of cases.

Initially put **CASE** $= \emptyset$, and do the following.

1. From the configuration S_t, choose a cell $C^t_{x,y}$ and extract its neighborhood $R^t_{x,y}$, where $R^t_{x,y}$ contains $C^t_{x,y}$ as a central cell.
2. From the configuration S_{t+1}, extract the cell $C^{t+1}_{x,y}$.
3. If the pair $(R^t_{x,y}, C^{t+1}_{x,y})$ is not in **CASE**, add it to **CASE**.
4. Iterate the above 1–3 steps for all the coordinates (x, y) of each configuration $S_0, \ldots, S_{n-1}$.

Figure 2 illustrates an example of a 2-dimensional 2-state CA in which cases are collected using a neighborhood of the 3×3 square.

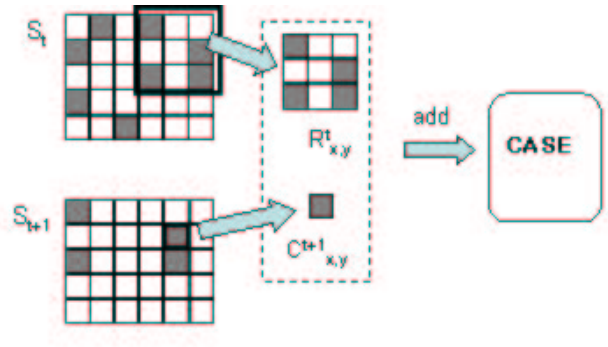

Fig. 2. Collecting Cases

In the above, a neighborhood is selected such that (i) it uniquely determines the next state of a target cell, and (ii) it does not contain cells which are irrelevant to the cellular changes of states. The condition (i) is examined by checking whether there exist two different cases $(R^t_{x,y}, C^{t+1}_{x,y})$ and $(R^t_{z,w}, C^{t+1}_{z,w})$ in **CASE** such that $R^t_{x,y} = R^t_{z,w}$ and $C^{t+1}_{x,y} \neq C^{t+1}_{z,w}$. If there is one, the neighborhood is changed by increasing its size. To remove redundant cells in (ii), we use hill-climbing search with a heuristic function which gives a higher score to a neighborhood that can distinguish every case with fewer cells.

[1] The scripts t and (x, y) are often omitted when they are not important in the context.

3.2 Decision Trees

To construct CA-rules, we classify cases using a *decision tree*. A decision tree used here has properties such that (i) each layer except the lowest one has a *classification condition* $Con_i(R)$ where R is a neighborhood; and (ii) each node $N_{i,j}$ except the root node has a *condition value* $V_{i,j}$ and the *next state* $NC_{i,j}$ of a cell (Figure 3).

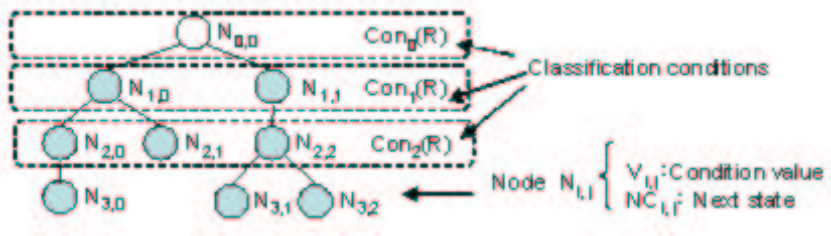

Fig. 3. A decision tree

Classification conditions classify cases according to the state of a neighborhood. For example, given a neighborhood R of the 3×3 square, the classification condition $Con(R) = \Sigma_{c_i \in R}\ c_i$ represents the sum of the states of cells in R. Classification conditions are computed using *genetic programming* (GP). In GP classification conditions are expressed by a tree structure (called a *condition tree*) in which $Con_0(R)$, $Con_1(R)$, $Con_2(R), \ldots$ are cascaded (cf. Figure 4(b)). GP applies genetic operations to a condition tree to find classification conditions which correctly classify all cases. A condition value $V_{i,j}$ is calculated by $Con_{i-1}(R)$ and $NC_{i,j}$ represents the state of a cell at the next time step.

Given a neighborhood $R^t_{x,y}$ of a cell in a configuration S_t, a decision tree returns the next state of the cell in the configuration S_{t+1} as an output. A decision tree is built using cases as follows.

Procedure: *Building a decision tree.*
Input : a set **CASE** of cases where (R_k, C_k) $(k \geq 1)$ is an element from **CASE**, and classification conditions $Con_0(R), \ldots, Con_{l-1}(R)$ $(l > 0)$.
Output : a decision tree with the depth l.

Set the initial tree as the root node $N_{0,0}$. For every case (R_k, C_k) from **CASE**, do the following.

1. At the root node $N_{0,0}$, if it has a child node $N_{1,j}$ $(j \geq 0)$ with $V_{1,j} = Con_0(R_k)$, apply the steps 2–3 to $N_{1,j}$. Otherwise, add the new node $N_{1,j}$ with $V_{1,j} = Con_0(R_k)$ and $NC_{1,j} = C_k$ to the tree.
2. For each node $N_{i,j}$ $(1 \leq i < l)$, if there is a node $N_{i+1,h}$ with $V_{i+1,h} = Con_i(R_k)$, apply the steps 2–3 to $N_{i+1,h}$. Else if there is no node $N_{i+1,h}$ with $V_{i+1,h} = Con_i(R_k)$ and it holds that $NC_{i,j} \neq C_k$, add a new node $N_{i+1,h}$ with $V_{i+1,h} = Con_i(R_k)$ and $NC_{i+1,h} = C_k$. Otherwise, the case (R_k, C_k) is successfully classified.
3. If there is a node $N_{l,j}$ with $NC_{l,j} = C_k$, the case (R_k, C_k) is successfully classified. Otherwise, the construction of a decision tree fails.

The procedure searches a node $N_{i+1,j}$ which has the value $V_{i+1,j}$ equal to $Con_i(R_k)$. When the node $N_{i+1,j}$ has the next state $NC_{i+1,j}$ which is not equal to C_k, the tree is expanded by adding a new node.[2] The depth l of a decision tree is determined by the number of classification conditions. A decision tree is expanded by introducing additional classification conditions until every case is classified. At the bottom level of a tree, no such expansion is performed. So, if there is a node $N_{l,j}$ such that the next state $NC_{l,j}$ does not coincide with C_k, the construction of a decision tree fails. This means that classification conditions used for the construction of a decision tree are inappropriate, then it is requested to re-compute new classification conditions. Those classification conditions which fail to classify all cases are given penalties in the process of the evaluation of GP, and they are not inherited to the next generation. This enables us to find more appropriate classification conditions.

Once a decision tree is successfully built, it can classify all cases from **CASE**. When a neighborhood $R^t_{x,y}$ is given as an input to a decision tree, the tree outputs the next state $NC_{i,j}$ of a node $N_{i,j}$ satisfying the following conditions: (1) every antecedent node $N_{k,l}$ $(0 < k < i)$ of $N_{i,j}$ satisfies the condition $V_{k,l} = Con_{k-1}(R^t_{x,y})$ and $N_{i,j}$ satisfies the condition $V_{i,j} = Con_{i-1}(R^t_{x,y})$, and (2) $N_{i,j}$ has no child node $N_{i+1,h}$ satisfying $V_{i+1,h} = Con_i(R^t_{x,y})$. For example, in the decision tree of Figure 3, given the input $R^t_{x,y}$, the tree outputs the next state $NC_{2,2}$ of $N_{2,2}$ if the next conditions are met: (1) $Con_0(R^t_{x,y}) = V_{1,1}$ and $Con_1(R^t_{x,y}) = V_{2,2}$, and (2) $Con_2(R^t_{x,y}) \neq V_{3,1}$ and $Con_2(R^t_{x,y}) \neq V_{3,2}$. The first condition ensures that the condition values of the nodes $N_{1,1}$ and $N_{2,2}$ satisfy the classification conditions $Con_0(R^t_{x,y})$ and $Con_1(R^t_{x,y})$, respectively. The second condition ensures that the condition values of the nodes $N_{3,1}$ and $N_{3,2}$ do not satisfy $Con_2(R^t_{x,y})$. Using these conditions a decision tree effectively searches a node which has the condition value satisfying classification conditions with respect to the input $R^t_{x,y}$, and outputs the next state of a cell. The node $N_{2,2}$ represents the following if-then rule:

$$\text{if } Con_0(R^t_{x,y}) = V_{1,1} \text{ and } Con_1(R^t_{x,y}) = V_{2,2} \text{ and } Con_2(R^t_{x,y}) \neq V_{3,1} \\ \text{and } Con_2(R^t_{x,y}) \neq V_{3,2} \text{ then } NC_{2,2}.$$

Every node except the root node represents such an if-then rule. Given an input $R^t_{x,y}$ a decision tree is built so as to satisfy the condition of only one such rule, so that the output $NC_{i,j}$ is uniquely determined by the input.

4 Experiments

To verify the effect of the proposed techniques, we present the results of two experiments such that: (a) given 2-dimensional 2-state CA configurations produced by a 2-dimensional 2-state CA, find 2-dimensional 2-state CA-rules which reproduce the same configurations; and (b) given 2-dimensional 2-state CA configurations produced by a 1-dimensional 2-state CA, find 2-dimensional 2-state CA-rules which reproduce the same configurations.

[2] A similar way of expanding a decision tree is in [2].

The purpose of these experiments is as follows. In the experiment (a), we verify that our procedure can find the original CA-rules which produce observed 2-dimensional 2-state configurations. In the experiment (b), on the other hand, we show that our procedure can discover new CA-rules which produce observed configurations but have different dimensions from the original one.

4.1 Finding the Original 2-dimensional 2-state CA-rules

In this experiment, we use the following 2-dimensional 2-state CA.

- A neighborhood consists of 9 square cells: a central cell and 8 orthogonally and diagonally adjacent cells. The state of a cell is either 0 or 1.
- A configuration consists of 100×100 cells. The initial configuration S_0 is randomly created.
- A sequence of configurations $S_0, \ldots, S_{20}$ are produced by the CA-rules such that: (1) if the central cell has exactly 2 surrounding cells of the state 1, the next state of the cell does not change; (2) else if the central cell has exactly 3 surrounding cells of the state 1, the next state of the cell is 1; (3) otherwise, the next state of the central cell is to 0.[3]

From the input configurations $S_0, \ldots, S_{20}$, the neighborhood, the condition tree, and the decision tree were constructed as shown in Figure 4.

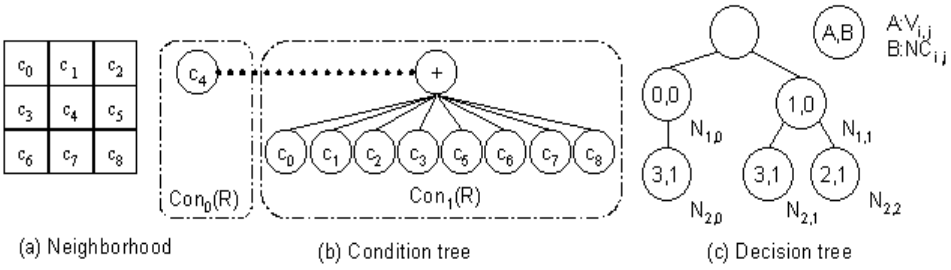

Fig. 4. Experimental result

The condition tree represents the classification conditions such that $Con_0(R) = c_4$ and $Con_1(R) = c_0+c_1+c_2+c_3+c_5+c_6+c_7+c_8$, where each c_i corresponds to a cell in the neighborhood (a), and takes the value of either 0 or 1. The decision tree (c) was built from this classification condition. In each node, a value in the left-hand side expresses the condition value $V_{i,j}$ and a value in the right-hand side expresses the next state $NC_{i,j}$.

Nodes of this decision tree represent the following if-then rules:

$N_{1,0}$: if $Con_0(R) = 0$ and $Con_1(R) \neq 3$ then $NC_{1,0} = 0$,
$N_{2,0}$: if $Con_0(R) = 0$ and $Con_1(R) = 3$ then $NC_{2,0} = 1$,
$N_{1,1}$: if $Con_0(R) = 1$ and $Con_1(R) \neq 3$ and $Con_1(R) \neq 2$ then $NC_{1,1} = 0$,
$N_{2,1}$: if $Con_0(R) = 1$ and $Con_1(R) = 3$ then $NC_{2,1} = 1$,
$N_{2,2}$: if $Con_0(R) = 1$ and $Con_1(R) = 2$ then $NC_{2,2} = 1$.

[3] This is known as the *Game of Life*.

Comparing these 5 rules with the original CA-rules, $N_{2,0}$ and $N_{2,1}$ correspond to the rule (2); $N_{2,2}$ corresponds to the rule (1) in which the state of the central cell is 1; $N_{1,0}$ and $N_{1,1}$ correspond to the rule (3) and the rule (1) in which the state of the central cell is 0. Thus, it is verified that the CA-rules constructed by the decision tree coincide with the original CA-rules which produce the input configurations. The result of this experiment shows that in 2-dimensional 2-state CAs the original CA-rules are reproduced by a sequence of input configurations. We also conducted a similar experiment for 2-dimensional 3-state CAs, and verified that the proposed method successfully finds the original CA-rules.

4.2 Discovering New 2-dimensional 2-state CA-rules

In this experiment, we use the following 1-dimensional 2-state CA.

- A neighborhood consists of 3 square cells: a central cell and its adjacent neighbors on each side. The state of a cell is either 0 or 1.
- A configuration consists of 100 arrayed cells. The initial configuration S_0 has a centered cell with the state 1 and all the other cells have the state 0.
- A sequence of configurations $S_0, \ldots, S_{20}$ are produced by the CA-rules such that: (1) if a neighborhood contains exactly one cell of the state 1, the next state of the central cell is 1; (2) otherwise the next state of the cell is 0.

Such a 1-dimensional CA produces 2-dimensional patterns. Figure 5 illustrates an example of an evolving 1-dimensional CA with 7 cells with 3 times applications of CA-rules. Thus, a 1-dimensional configuration S_i $(0 \leq i \leq 20)$

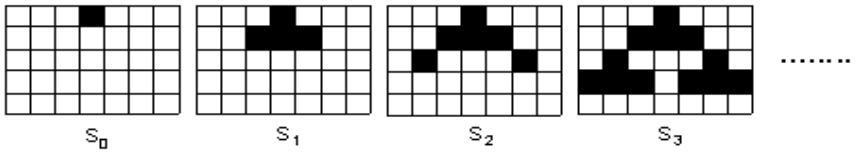

Fig. 5. Evolution of a 1-dimensional CA

is identified with the corresponding 2-dimensional configuration S'_i which is obtained by vertically arranging S_j $(j \leq i)$ downward in the 100×21 grid.

We used such 2-dimensional configurations $S'_0, \ldots, S'_{20}$ as an input. As a result, we obtained the neighborhood, the condition tree, and the decision tree of Figure 6. The meaning of the figure is the same as that of Figure 4.

It is worth noting that the obtained neighborhood (a) is 2-dimensional. The condition tree (b) means that: $Con_0(R) = c_3$ and $Con_1(R) = c_0 + c_1 + c_2$. The decision tree (c) represents the following if-then rules:

$$
\begin{aligned}
N_{1,0}: &\ \text{if } Con_0(R) = 0 \text{ and } Con_1(R) \neq 1 \text{ then } NC_{1,0} = 0,\\
N_{2,0}: &\ \text{if } Con_0(R) = 0 \text{ and } Con_1(R) = 1 \text{ then } NC_{2,0} = 1,\\
N_{1,1}: &\ \text{if } Con_0(R) = 1 \text{ then } NC_{1,1} = 1.
\end{aligned}
$$

Viewing c_3 as the central cell in the neighborhood, these rules are interpreted as follows: $(N_{1,0})$ When the state of a central cell is 0 and the number of cells

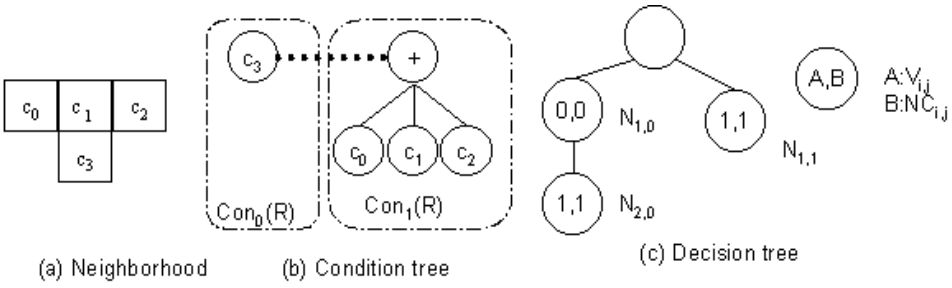

Fig. 6. Experimental result

of the state 1 in the neighborhood is not 1, the state remains 0; ($N_{2,0}$) When the state of a central cell is 0 and the number of cells of the state 1 in the neighborhood is exactly 1, the state changes to 1; ($N_{1,1}$) When the state of a central cell is 1, the state remains 1. Applying these rules to the initial configuration S'_0 produces configurations $S'_1, \ldots, S'_{20}$, which coincide with the input configurations. This result shows that the procedure discovers new 2-dimensional CA-rules which reproduce the same pattern produced by a 1-dimensional CA. By this experiment, it is observed that the proposed method not only reproduces the original CA-rules, but can also discover new rules in different dimensions.

5 Conclusion

In this paper, we developed techniques for automatic generation of cellular automata rules. We first extracted cases from input CA configurations then built a condition tree and a decision tree. A decision tree correctly classifies every case and expresses CA-rules which reproduce the input CA configurations. We showed by experiments that the proposed method can successfully find the original CA-rules which generate given 2-dimensional, 2/3-state CA configurations. It is also shown that 2-dimensional 2-state new CA-rules are discovered from the input 1-dimensional 2-state CAs. In this paper, we performed experiments to reconstruct CA-rules from input CA configurations which are produced by existing CA-rules. On the other hand, in real-life problems input configurations generally include noise, the goal is then to discover unknown CA-rules if any. In this case, the relevant neighborhood of a cell would probably fail to converge for moderate amount of noise and it would be necessary to introduce an appropriate threshold on the number of conflicting cases. To apply the proposed method in practice, we will work on further refinement of techniques in future research.

References

1. A. Adamatzky. *Identification of Cellular Automata*. Taylor& Francis, London, 1994.
2. B. Liu, M. Hu, and W. Hsu. Intuitive representation of decision trees using general rules and exceptions. In: *Proc. AAAI-2000*, pp. 615–620, MIT Press, 2000.
3. M. Mitchell, P. T. Hraber, and J. P. Crutchfield. Revisiting the edge of chaos: evolving cellular automata to perform computations. *Complex Systems* 7, pp. 89–130, 1993.
4. T. Toffoli and N. Margolous. *Cellular Automata Machines*. MIT Press, 1987.