Journal of Cellular Automata, Vol. 2, pp. 1–20
Reprints available directly from the publisher
Photocopying permitted by license only

Identifying Cellular Automata Rules

KEN-ICHI MAEDA*, AND CHIAKI SAKAMA†

Department of Computer and Communication Sciences
Wakayama University
Sakaedani, Wakayama 640 8510, Japan

Received: Nov 1, 2006; Accepted: Nov 9, 2006.

This paper studies a method for identifying cellular automata rules (CA rules). Given a sequence of CA configurations, we first seek an appropriate neighborhood of a cell and collect cellular changes of states as *evidences*. The collected evidences are then classified using a *decision tree*, which is used for constructing CA transition rules. Conditions for classifying evidences in a decision tree are computed using *genetic programming*. We perform experiments using several types of CAs and verify that the proposed method successfully identifies correct CA rules.

Keywords: Cellular automata, identification problem, genetic programming, decision tree

1 INTRODUCTION

Cellular automata (CAs) 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 simulating various complex systems in the real world [8, 10], and for modeling advanced computation such as massively parallel computers and evolutionary computation [7]. However, complex behavior of CAs is difficult to understand, which makes hard to design CAs having desired behavior. The task of designing CAs 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

*Current address: Hitachi Systems & Services, Ltd., Tokyo.
†Contact Author: sakama@sys.wakayama-u.ac.jp

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 want to output cell-state transition rules (CA-rules) from an input sequence of configurations. Reconstruction of CA-rules from input configurations is known as the *identification problem* [1]. Automatic discovery of CA-rules is also studied in the context of *density classification task* [5]. 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 configuration. Several researchers pursue the problem using *genetic programming* [2] and *evolutionary computation* [3, 9]. The purpose of this paper is also identifying CA-rules, but our goal is different from those studies. The objective of this study is to develop techniques for finding CA-rules which reflect cellular changes in observed CA configurations. Technically, our goal is achieved by the following steps. Given a sequence of CA configurations we first seek an appropriate neighborhood of a cell using a heuristic function, and collect cellular changes of states as *evidences*. The collected evidences are then classified using a *decision tree* which is used for constructing CA-rules. Conditions for classifying evidences 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.

This paper is a revised and extended version of [4]. The rest of this paper is organized as follows. Section 2 presents a brief introduction of cellular automata. Section 3 provides an algorithm for identifying CA-rules. Section 4 shows experimental results to verify the proposed method. Section 5 analyzes experimental results and discusses related issues. Section 6 summarizes the paper.

2 CELLULAR AUTOMATA

A *cellular automaton* (CA) has a n-dimensional cellular space and consists of a regular grid of *cells*. *Neighborhood* of a cell consists of the surrounding adjacent cells, including the cell itself. For 1-dimensional CAs, a cell is connected to r local *neighbors* in each side, including itself, where r is called a *radius*. In this case, the *size* of a neighborhood is $2r + 1$ (Figure 1(a)). For 2-dimensional CAs, two different types of neighborhoods of the radius 1 exist. The *von Neumann neighborhood* has the size 5, consisting of the central cell, cells adjacent to above and below, and to right and left (Figure 1(b)). The *Moore neighborhood* has the size 9, consisting of the central cell and the surrounding 8 adjacent cells (Figure 1(c)).

Each cell in a CA has a finite number of possible *states*. For instance, in 2-state CAs a cell has either 0 or 1, white or black, and so on. The

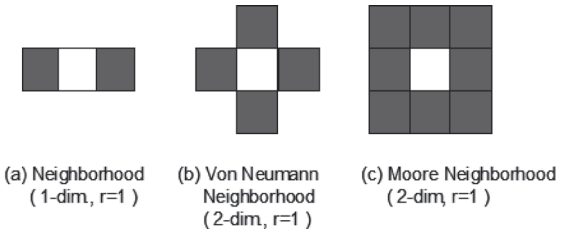


FIGURE 1
Neighborhoods.

state of a cell in the next time step is determined by its current state and the states of its surrounding cells. Such transitions of cellular states are represented by *cell-state transition rules* (called *CA-rules*). The state of each cell changes synchronously in discrete time steps according to local and identical CA-rules. The collection of finite number of cells in the grid at some time step is called a *configuration*. S_t $(0 \leq t \leq n)$ represents a configuration of a CA at a time step t. A configuration S_t consists of cells $C^t_{x,y}$, where x and y represent coordinates of cells in the configuration (Figure 2).

3 IDENTIFYING CA-RULES

Given a sequence of configurations $S_0, \ldots, S_n$ as an input, we want to output CA-rules which, applied to the initial configuration S_0, 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 evidences from input configurations.
3. Construct a decision tree to classify evidences and extract CA-rules.

In the following subsections, we explain techniques using 2-dimensional 2-state CAs, but the techniques are applied to m-dimensional n-state CAs without major changes.

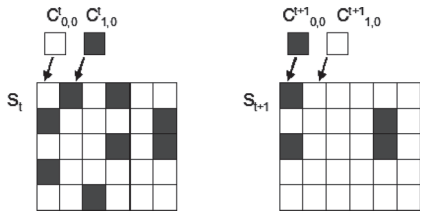


FIGURE 2
Configurations of a CA.

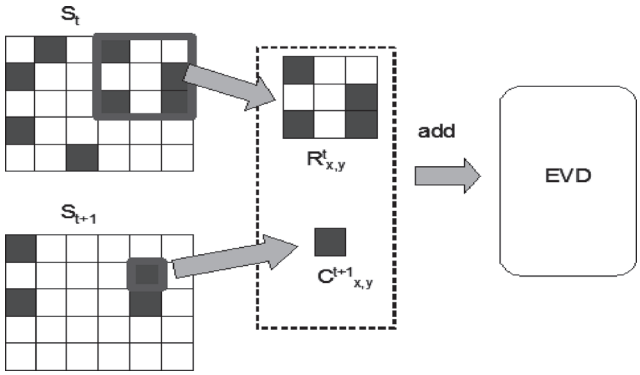


FIGURE 3
Collecting evidences.

3.1 Collecting Evidences

We first describe a method of collecting evidences. An *evidence* 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 cell at the next time step $t+1$.[1] Evidences are collected using the following procedure.

Procedure: *Collecting Evidences.*

Input: a sequence of configurations $S_0, \ldots, S_n$.
Output: a set EVD of evidences.

Initially put $EVD = \emptyset$, 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]
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 EVD, add it to EVD.
4. Iterate the above 1–3 steps for all the coordinates (x, y) of each configuration $S_0, \ldots, S_{n-1}$.

When a configuration S_t contains N cells, there are N evidences. Thus, the number of evidences stored in EVD becomes $N \times n$. Figure 3 illustrates an example of a 2-dimensional 2-state CA in which evidences are collected using the Moore neighborhood.

[1] $R^t_{x,y}$ is often abbreviated as R when no confusion arises in the context.
[2] In a configuration, each edge on the grid is assumed to be linked to the edge in the opposite side.

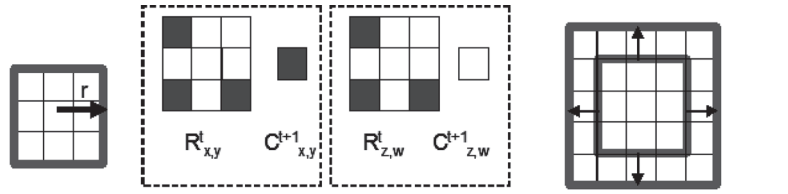


FIGURE 4
Neighborhood expansion.

3.2 Deciding a Neighborhood

There are several candidates for an appropriate neighborhood of a cell. A necessary condition for selecting a neighborhood is that it uniquely determines cell-state transitions. It is also preferable that a neighborhood does not contain redundant cells which are irrelevant to the cellular changes of states.

Taking care of these conditions, a neighborhood of a cell is decided by the following procedures.

Procedure: *Deciding the Radius of a Neighborhood.*

Input: a sequence of configurations $S_0, \ldots, S_n$.
Output: a radius r.

1. Put a neighborhood as a square of the radius $r = 1$.
2. Evidences are collected in the set EVD from all the coordinates (x, y) of configurations $S_0, \ldots, S_{n-1}$ using the neighborhood.
3. If there are two different evidences $(R^t_{x,y}, C^{t+1}_{x,y})$ and $(R^t_{z,w}, C^{t+1}_{z,w})$ in EVD such that $R^t_{x,y} = R^t_{z,w}$ and $C^{t+1}_{x,y} \neq C^{t+1}_{z,w}$, then set $r := r + 1$ and go to step 2. Otherwise, the procedure ends.

Step 3 of the above procedure illustrates the case where a neighborhood does not uniquely determine the next state of a cell. In this case, the radius r is increased by one and the collection/test loop is restarted. Figure 4 illustrates an example of a 2-dimensional CA in which r is increased from 1 to 2.

A neighborhood with the radius computed by the above procedure may contain redundant cells which are irrelevant to the cellular changes of states. Such redundant cells are removed by the following procedure.

Procedure: *Removing Redundant Cells*.

Input: a sequence of configurations $S_0, \ldots, S_n$, and a neighborhood R as the square of the radius r computed by the above procedure.

Output: an optimized neighborhood.

1. Produce a set CN of candidates of neighborhoods which are obtained by deleting an arbitrary cell from R.
2. Collect evidences EVD' from all the coordinates (x, y) of configurations $S_0, \ldots, S_{n-1}$ using a neighborhood R' in CN.
3. For each R' in CN, apply the following heuristic function:

$$Eval\_NB = Neighbors(EVD') \times 1/N$$

   where $Neighbors(EVD')$ returns 0 if there are evidences in EVD' such that the next states differ with the same neighborhood; otherwise it returns 1. N is the number of cells which constitute the neighborhood R'.
4. Let $Eval\_NB_0$ be the value computed by EVD using the input neighborhood R. If a new neighborhood R' in CN gets a value $Eval\_NB$ which is higher than $Eval\_NB_0$, replace R with R'.
5. Repeat the above steps 1–4 until no candidate of a neighborhood in CN gets a higher value $Eval\_NB$ than R'.

In the above procedure, the first step includes a non-deterministic choice of a cell to be deleted. In practice, it is realized by removing a cell one by one from left to right, up to down, for instance. The function $Eval\_NB$ gives a higher score to a neighborhood which classifies every evidences with fewer cells. Starting from the initial R, the function is used for finding appropriate neighborhood by hill-climbing search.

3.3 Classification Conditions

If we construct CA-rules from arbitrary evidences, they become relations between a neighborhood and the next state of a cell. Figure 5 illustrates an example which presents three different rules constructed by three different evidences.

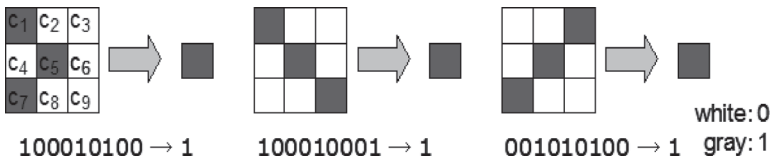


FIGURE 5
CA-rules constructed by evidences.

However, such enumeration of bit-strings is difficult to read and the number of rules becomes huge according to the increase of evidences. To construct simplified CA-rules, we classify evidences by their neighborhood patterns. For instance, the state changes in Figure 5 are represented such that: "For input cells $c_1c_2 \ldots c_9$, if three of the odd-numbered cells c_i $(i = 1, 3, 5, 7, 9)$ are 1, then the next state of c_5 is 1"; or "if the current state of c_5 is 1 and there is an even number of surrounding cells with the state 1, the next state of c_5 is 1", and so on. Such conditions on neighborhoods are represented by *classification conditions*. Given a neighborhood R, a classification condition is represented by $Con(R)$. Classification conditions are used for classifying evidences in the process of constructing a decision tree.

A *decision tree* is popularly used in the field of machine learning to discriminate among objects. A tree consists of *nodes* and *edges*. The top node is the *root* and the bottom nodes are *leaves*. A node is connected to its *children* nodes by edges. The *depth* of a node in a tree is the number of edges from the root to that node. All nodes with the depth i lie in the *i-th layer*. The *depth of a tree* is the maximum depth of all nodes in the tree. A decision tree starts from the root node and expands it by splitting objects into new nodes. Leaves of the tree represent sets of objects with an identical classification.

A decision tree used here has the following properties:

1. Each i-th layer except leaf nodes has a classification condition $Con_i(R)$.
2. Each node $N_{i,j}$ except the root node has a *condition value* $V_{i,j}$, which is computed by $Con_{i-1}(R)$, and the *next state* $NC_{i,j}$ of a cell (Figure 6).

Classification conditions are computed using *genetic programming* (GP). A classification condition in GP is expressed by a tree structure (called a *condition tree*). Figure 7 illustrates the condition tree representing the

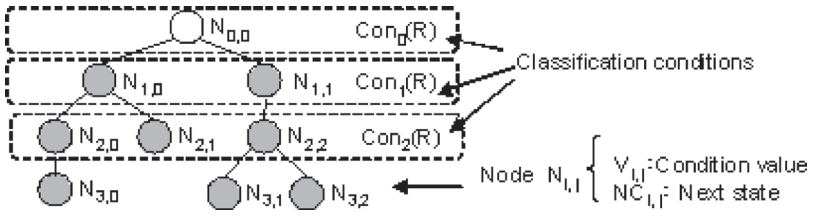


FIGURE 6
A decision tree.

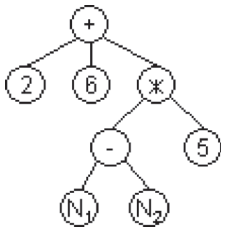


FIGURE 7
A condition tree.

condition $2 + 6 + ((N_1 - N_2) \times 5)$.[3] Each node of a condition tree is either an *operator node* or an *elemental leaf*. An operator node has arithmetic operators $(+, -, *, /)$ and has an operator node or an elemental leaf as a child. An elemental leaf is either an integer 0 – 10 or a state of a cell in the neighborhood. For instance, in a 2-state CA the state of a cell is represented by 0 or 1. Elemental leaves have no children.

We use more than one classification condition to classify evidences, so that GP uses a condition tree which is obtained by connecting several condition trees. For example, when a decision tree has three classification conditions $Con_0(R) = 2 + (N_1 - N_2) \times 5$, $Con_1(R) = ((N_2 - 4) \times N_3)/5$, and $Con_2(R) = N_0 + N_1 + 5$ at each layer, the condition tree becomes the structure as shown in Figure 8.

GP applies the following genetic operations to a condition tree to find classification conditions which correctly classify all evidences.

- *Mutation*: Each node in a tree is changed at a certain rate. Three types of mutation: change of nodes, addition of new nodes, and deletion of existing nodes, are used.
- *Crossover*: Two condition trees are selected and a subtree of each tree is exchanged.

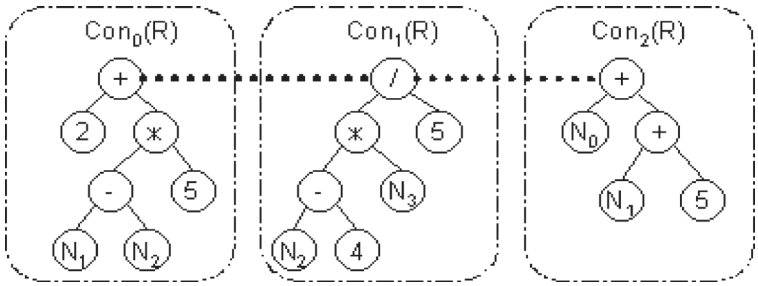


FIGURE 8
A condition tree used by GP.

[3] $*$ represents multiplication.

A condition tree generated by GP is evaluated using the following function:

$$Eval\_CT = [1 + c \times (p/C\_Node + q/D\_Node)] \times Num(EVD),$$

where $c = 1$ if every evidence in EVD is classified; otherwise, $c = 0$.

In the above, $Num(EVD)$ is the number of evidences in EVD which are correctly classified by a decision tree; C_Node is the number of nodes in a condition tree; D_Node is the number of nodes in a decision tree; and p, q are parameters to adjusts evaluation.

The function gives a higher score to a condition tree which successfully classifies more evidences. A condition tree is extended until it finds a classification condition which classifies all evidences correctly. Once a condition tree which correctly classifies all evidences is found ($c = 1$), the function gives a higher score to a condition tree which has fewer nodes in C_Node or a decision tree with fewer nodes in D_Node. Thus, the function leads in the direction to simplify a condition tree or a decision tree, maintaining the capability that all evidences are successfully classified. Parameters p and q are used to adjust priority between the size of a condition tree and the size of a decision tree. When $p > q$, priority is given to a condition tree. In this case, the number of nodes of a condition tree decreases, but the number of nodes of a decision tree may increase. This leads to the increase of the number of discovered CA-rules. When $p < q$, on the other hand, the number of nodes of a decision tree decreases, but the number of nodes of a condition tree may increase. As a consequence, the conditions of discovered CA-rules may become complex.

3.4 Building Decision Trees

Given a neighborhood 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 procedure for building a decision tree is as follows.

Procedure: *Building a Decision Tree.*

Input: a set EVD of evidences 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 evidence $(R^t_{x,y}, C^{t+1}_{x,y})$ from EVD, do the following.

1. At the root node $N_{0,0}$, if it has a child node $N_{1,j}$ $(j \geq 0)$ with $V_{1,j} = Con_0(R^t_{x,y})$, apply the steps 2–3 to $N_{1,j}$. Otherwise, add the new node $N_{1,j}$ with $V_{1,j} = Con_0(R^t_{x,y})$ and $NC_{1,j} = C^{t+1}_{x,y}$ to the tree.
2. For each node $N_{i,j}$ $(1 \leq i < l)$, if there is a node $N_{i+1,h}$ with $V_{i+1,h} = Con_i(R^t_{x,y})$, apply the steps 2–3 to $N_{i+1,h}$. Else if there is no

node $N_{i+1,h}$ with $V_{i+1,h} = Con_i(R^t_{x,y})$ and it holds that $NC_{i,j} \neq C^{t+1}_{x,y}$, add a new node $N_{i+1,h}$ with $V_{i+1,h} = Con_i(R^t_{x,y})$ and $NC_{i+1,h} = C^{t+1}_{x,y}$. Otherwise, the evidence $(R^t_{x,y}, C^{t+1}_{x,y})$ is successfully classified.

3. If there is a node $N_{l,j}$ with $NC_{l,j} = C^{t+1}_{x,y}$, the evidence $(R^t_{x,y}, C^{t+1}_{x,y})$ is successfully classified. Otherwise, the construction of a decision tree fails.

The procedure searches a node $N_{i+1,j}$ having the condition value $V_{i+1,j}$ that is equal to the classification condition $Con_i(R^t_{x,y})$. When the node $N_{i+1,j}$ has the next state $NC_{i+1,j}$ which is not equal to $C^{t+1}_{x,y}$, the tree is expanded by adding a new node. The depth l of a decision tree is determined by the number of classification conditions.

Figure 9 illustrates an expansion of a decision tree. In (a), suppose that the node $N_{1,0}$ has the condition value $V_{1,0} = Con_0(R^t_{x,y})$ and the next state $NC_{1,0} \neq C^{t+1}_{x,y}$. Then, the tree is expanded by adding a new node $N_{2,0}$ as (b), where $V_{2,0} = Con_1(R^t_{x,y})$ and $NC_{2,0} = C^{t+1}_{x,y}$. In this case, the node $N_{2,0}$ represents evidences which satisfy the classification conditions $Con_0(R^t_{x,y}) = V_{1,0}$ and $Con_1(R^t_{x,y}) = V_{2,0}$, and have the next state $NC_{2,0}$ which is different from $NC_{1,0}$. By contrast, the node $N_{1,0}$ in (b) represents evidences which satisfy $Con_0(R^t_{x,y}) = V_{1,0}$ but $Con_1(R^t_{x,y}) \neq V_{2,0}$, and have the next state $NC_{1,0}$. In this way, a tree is expanded by introducing additional classification conditions until every evidence 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 $C^{t+1}_{x,y}$, 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 evidences 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.

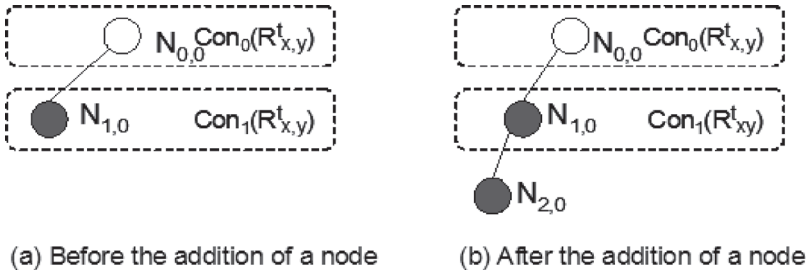


FIGURE 9
Expansion of a tree.

Once a decision tree is successfully built, it can classify all evidences from EVD. Then, the tree can decide the next state of a cell with respect to a given neighborhood. 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 two 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 $V_{i,j} = Con_{i-1}(R^t_{x,y})$,
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 6, 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 $Con_1(R^t_{x,y}) = V_{2,2}$;
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 conduct three experiments as follows.

1. 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.
2. Given 2-dimensional 3-state CA configurations produced by a 2-dimensional 3-state CA, find 2-dimensional 3-state CA-rules which reproduce the same configurations.

3. 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.

The purpose of these experiments is as follows. In the first experiment, we verify that our procedure can find the original CA-rules which produce observed 2-dimensional 2-state configurations. In the second experiment, we verify this fact for 2-dimensional 3-state CAs. In the third experiment, we show that our procedure can discover new CA-rules which produce observed configurations but have different dimensions from the original ones.

In these experiments, we used the following parameters for computing classification conditions by GP.

- A population consists of ten individuals representing condition trees.
- Using the function $Eval\_CT$ as a fitness function, select two individuals from the population by the elitist strategy.
- Select two individuals from the population by the roulette wheel selection and crossover them at a randomly selected cut-point. Repeat this step four times and eight individuals are newly produced.
- For these eight individuals, a mutation is performed at the probability 5% to each node of a condition tree. A mutation causes change, addition, and deletion of a node in a condition tree at an equal rate. If deletion is performed on a node having children, those children are also deleted.
- To evaluate a condition tree, parameters are set in the evaluation function $Eval\_CT$ as $p = q = 500$, i.e., the number of nodes in a decision tree and the number of nodes in a condition tree have the same priority for deciding an appropriate condition tree.
- The number of generations is 20,000.

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

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

- A neighborhood consists of nine square cells in the Moore neighborhood.
- 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 two surrounding cells of the state 1, the next state of the cell does not change; (2) else if the central cell has exactly three 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.

Note that the above CA-rules are used in the *Game of Life* [6].

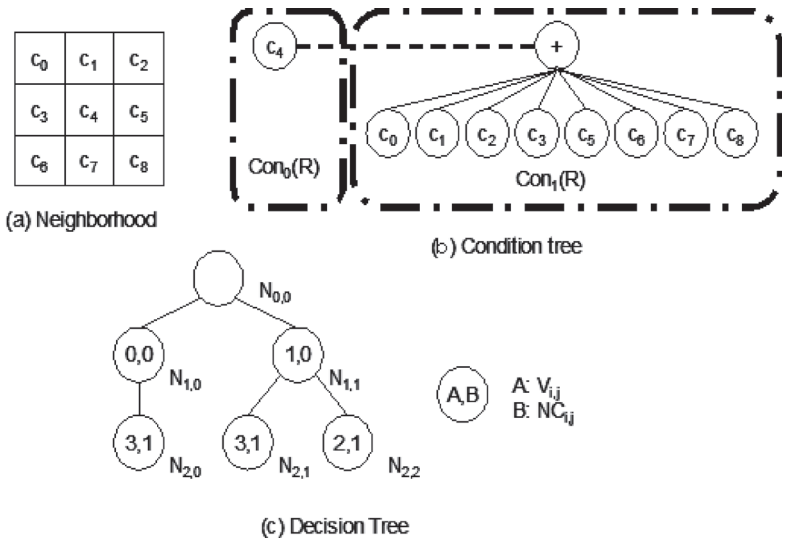


FIGURE 10
Experimental result.

From the input configurations $S_0, \ldots, S_{20}$, the neighborhood, the condition tree, and the decision tree were constructed as shown in Figure 10.

First, evidences are collected and a neighborhood is fixed. The Moore neighborhood (a) uniquely determines the cellular changes, so it is selected as an appropriate one. A condition tree was then constructed using GP. The produced condition tree (b) represents the classification conditions such that

$$Con_0(R) = c_4,$$
$$Con_1(R) = c_0 + c_1 + c_2 + c_3 + c_5 + c_6 + c_7 + c_8,$$

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 $NC_{i,j}$.

Each node of this decision tree represents 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 $NC_{2,2} = 1$.

Comparing these five rules with the original CA-rules which produced the configurations,

- $N_{2,0}$ 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 produced 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.

4.2 Discovering the Original 2-dimensional 3-state CA-rules

In this experiment, we use the following 2-dimensional 3-state CA.

- A neighborhood consists of five square cells in the von Neumann neighborhood.
- The state of a cell is one of the three states: 0, 1 or 2.
- A configuration consists of 100×100 cells.
- The initial configuration S_0 has 4 cells with the state 1 in the coordinates (x, y) with $50 \leq x,\ y \leq 51$; all the other cells have the state 0.
- A sequence of configurations $S_0, \ldots, S_{20}$ are produced by the CA-rules as follows: for each cell $C^t_{x,y}$ in S_i, consider the sum of the states of all cells in the neighborhood $R^t_{x,y}$ (written as $\sum R^t_{x,y}$). Then, cell-state transitions are specified as: (1) when $\sum R^t_{x,y} \leq 1$, $C^{t+1}_{x,y}$ becomes 0; (2) when $\sum R^t_{x,y} = 2$ or $\sum R^t_{x,y} = 3$, $C^{t+1}_{x,y}$ becomes 1; (3) when $4 \leq \sum R^t_{x,y} \leq 6$, $C^{t+1}_{x,y}$ becomes 2; (4) when $\sum R^t_{x,y} \geq 7$, $C^{t+1}_{x,y}$ becomes 0.

Figure 11 illustrates a part of S_{20} produced by the above CA-rules.

From the input configurations $S_0, \ldots, S_{20}$, the neighborhood, the condition tree, and the decision tree of Figure 12 were obtained.

The von Neumann neighborhood (a) is selected as an appropriate neighborhood. The condition tree (b) represents the classification condition:

$$Con_0(R) = c_0 + c_1 + c_2 + c_2 + c_3 + c_4.$$

The decision tree (c) expresses the following if-then rules:

$$N_{1,0}\text{: if } Con_0(R) = 0\,,\ \text{then } NC_{1,0} = 0,$$

$$N_{1,1}\text{: if } Con_0(R) = 1\,,\ \text{then } NC_{1,1} = 0,$$

$$N_{1,2}\text{: if } Con_0(R) = 2\,,\ \text{then } NC_{1,2} = 1,$$

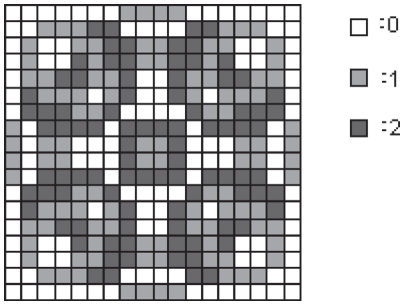


FIGURE 11
A 2-dimensional 3-state CA.

$$N_{1,3}\text{: if } Con_0(R) = 3\,,\ \text{then } NC_{1,3} = 1,$$
$$N_{1,4}\text{: if } Con_0(R) = 4\,,\ \text{then } NC_{1,4} = 2,$$
$$N_{1,5}\text{: if } Con_0(R) = 5\,,\ \text{then } NC_{1,5} = 2,$$
$$N_{1,6}\text{: if } Con_0(R) = 6\,,\ \text{then } NC_{1,6} = 2,$$
$$N_{1,k}\text{: if } Con_0(R) = k\,,\ \text{then } NC_{1,k} = 0 \quad (7 \le k \le 12).$$

Comparing these 13 rules with the original CA-rules, the following correspondences are observed.

- $N_{1,0}$ and $N_{1,1}$ correspond to the rule (1).
- $N_{1,2}$ and $N_{1,3}$ correspond to the rule (2).

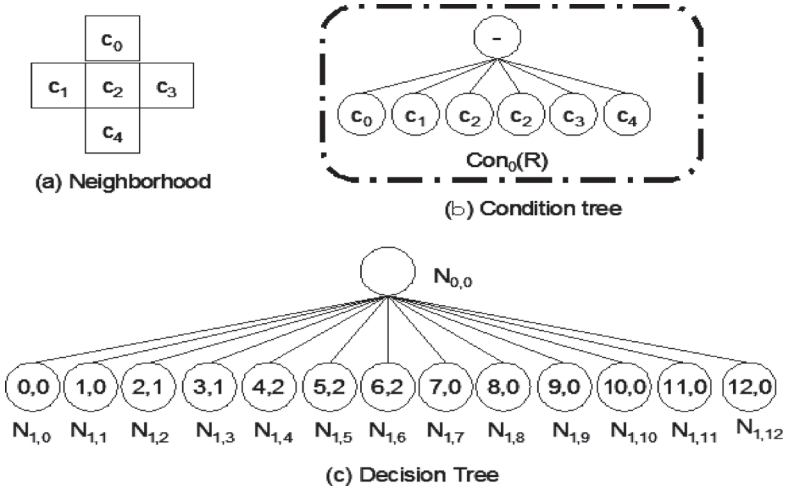


FIGURE 12
Experimental result.

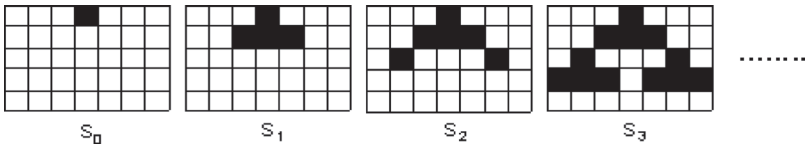


FIGURE 13
Evolution of a 1-dimensional CA.

- $N_{1,4}$, $N_{1,5}$, and $N_{1,6}$ correspond to the rule (3).
- $N_{1,k}$ with $7 \leq k \leq 12$ correspond to to the rule (4).

Thus again, discovered CA-rules coincide with the CA-rules which produced the input configurations. The result of this experiment shows that the proposed techniques are also effective in 2-dimensional 3-state CAs.

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

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

- A neighborhood consists of three 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 13 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 $(0 \leq i \leq 20)$ is identified with a 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 14. The meaning of the figure is the same as that of Figure 10.

It is worth noting that the obtained neighborhood (a) is 2-dimensional. The condition tree (b) means that

$$Con_0(R) = c_3,$$
$$Con_1(R) = c_0 + c_1 + c_2.$$

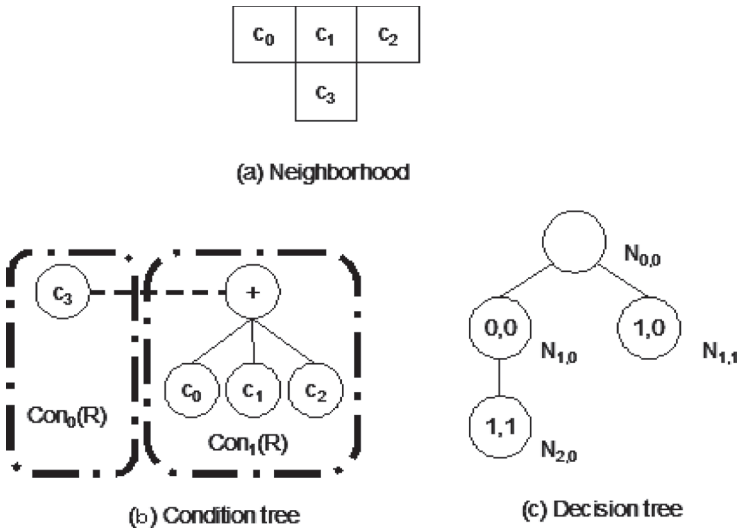


FIGURE 14
Experimental result.

The decision tree (c) represents the following if-then rules:

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

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 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.

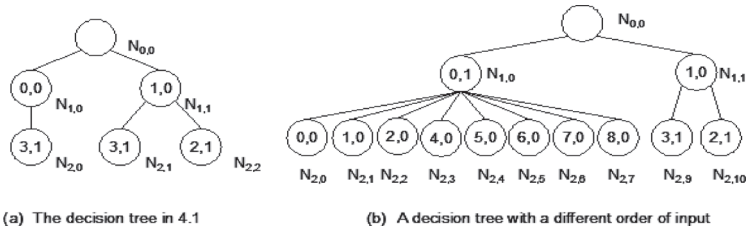
Let S'_0 be the 2-dimensional configuration which corresponds to the 1-dimensional configuration S_0. Then, applying the above rules to the initial configuration S'_0 produces configurations $S'_1, \ldots, S'_{20}$, which coincide with the input configurations. This result shows that the procedure discovers 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 a different dimension.

5 DISCUSSION

The results of experiments show that the proposed techniques compute appropriate neighborhoods and reproduce proper CA-rules which generate input CA configurations. We discuss some issues which need more elaboration.

First, the evaluation function $Eval\_CT$ for constructing a condition tree can find classification conditions which classify every evidence correctly, but there is no guarantee to find the simplest conditions. This is due to the fact that the function has three different parameters for optimization, the number of correctly classified evidences, the number of nodes in a condition tree, and the number of nodes in a decision tree. Consequently, the function has more than one peak, each of which has a steep landscape and long-distanced with one another. In this situation, GP may get stuck in local optima, and as a consequence, a condition tree or a decision tree often becomes huge and complicated. To evaluate classification conditions, if we only take care of classifying every evidence, a condition tree may include redundant nodes and have an intricate structure. On the other side, if we give preference to simplifying the structure of a tree, search goes wrong by deleting nodes before finding a correct condition tree. Thus, the correctness of classification and the simplicity of a structure are both important for evaluating condition trees. To find an appropriate solution by GP, it is considered to increase the number of individuals/generations, which would lead to diversity in the population and brings more chance to find the optimal solution.

The second problem is that the structure of a decision tree generally depends on the order of input evidences. This is because, in the process of constructing a decision tree, an input evidence decides the next state of a node, which changes the structure of a subtree connected to the node. For instance, Figure 15(a) shows the decision tree which is built in Section 4.1. By contrast, Figure 15(b) illustrates a different decision tree which is constructed using the same classification conditions with a different order of input evidences. In (a) the node $N_{1,0}$ is firstly added to the tree using a evidence with the next state 0, then the node $N_{2,0}$ is



(a) The decision tree in 4.1 (b) A decision tree with a different order of input

FIGURE 15
Decision trees with different orders of input evidences.

added using another evidence with the next state 1. In (b), on the other hand, the node $N_{1,0}$ is firstly added to the tree using a evidence with the next state 1, then nodes $N_{2,0}, \ldots, N_{2,7}$ are added using evidences with the next state 0. In this example, when the classification condition $Con_0(R)$ has the condition value 0, the condition $Con_1(R)$ generates children in which the number of nodes with the next state 0 is larger than the number of nodes with the next state 1. As a result, a decision tree which has a completely different structure (but has the same meaning) is constructed. Thus, under the same classification conditions decision trees with different structures exist in general. This also affects the evaluation of a condition tree, since the evaluation function $Eval\_CT$ has a parameter which reflects the number of nodes of a decision tree. A negative side-effect of this is that the function happens to produce a condition tree which fits into a particular order of input evidences. A possible solution to this problem is to change the order of evidences in the process of evaluating a condition tree using GP.

6 CONCLUSION

In this paper, we developed techniques for identifying cellular automata rules. We first extracted evidences from input CA configurations then built a decision tree using classification conditions. A decision tree correctly classifies all evidences 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. Moreover, it is shown that 2-dimensional 2-state new CA-rules are discovered from the input 1-dimensional 2-state CAs. The results of this paper show that our method can not only find the original CA-rules, but discover new CA-rules which reproduce observed configurations.

The present technique is extended in several ways. For instance, this paper considered CAs in which the next state of a cell is determined by the current state of the cell and its neighborhood. On the other hand, there are CAs such that the next state of a cell depends on the preceding multiple states of neighborhoods. To cope with such CAs, it is necessary to collect evidences having multiple states of neighborhoods. A classification condition is then extended to include neighborhood changes, and a decision tree is constructed using these extended conditions accordingly. Another extension is to find CA-rules from multiple sequences of CA-configurations. In this case, multiple sequences of CA-configurations are given as an input, and evidences are collected in the same manner as the case of a single sequence. Using those evidences, classification conditions are computed and a decision tree is built.

Several issues remain for future work. One important issue is an application to real-life problems. The purpose of this research is to give an algorithm for automatic generation of CA-rules which reproduce the input configurations. To verify the effect, in the experiments we used input CA configurations which are produced by existing CA-rules. In real-life problems, however, input configurations generally include noise. The goal is then to discover unknown CA-rules, if any, by effectively removing noise. When evidences include noise, it may not be possible to classify every evidence just by extending a neighborhood of a cell. One possible solution in this case is to fix the size of a radius, and search an appropriate neighborhood within the radius. When there are evidences such that the next states of a cell differ with the same neighborhood, the majority of patterns having the same next state are considered correct evidences and other fewer patterns are considered noise. We will work on further refinement of techniques and experiments in more complex problems.

REFERENCES

[1] Adamatzky, A. (1994). *Identification of Cellular Automata*. Taylor& Francis, London.

[2] Andre, D., Forrest, H. Bennett, III., and Koza, J. R. (1996). Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In: *Proceedings of the First Annual Conference on Genetic Programming,* MIT Press, 3–11.

[3] Juillé, H., and Pollack, J. (1998). Coevolving the "ideal" trainer: application to the discovery of cellular automata rules. In: *Proceedings of the third annual conference of genetic programming,* 519–527.

[4] Maeda, K., and Sakama, C. (2003). Discovery of cellular automata rules using cases. In: *Proceedings of the 6th International Conference on Discovery Science,* Lecture Notes in Artificial Intelligence 2843, Springer-Verlag, 357–364.

[5] Mitchell, M., Hraber, P. T., and Crutchfield, J. P. (1993). Revisiting the edge of chaos: evolving cellular automata to perform computations. *Complex Systems, 7,* 89–130.

[6] Sigmund, K. (1993). *Games of Life, Explorations in Ecology, Evolution, and Behavior,* Oxford University Press.

[7] Sipper, M. (1997). *Evolution of parallel cellular machines: the cellular programming approach,* Lecture Notes in Computer Science 1194, Springer-Verlag.

[8] Toffoli, T., and Margolous, N. (1987). *Cellular Automata Machines,* MIT Press.

[9] Werfel, J., Mitchell, M., and Crutchfield, J. P. (2000). Resource sharing and coevolution in evolving cellular automata. *IEEE Transactions on Evolutionary Computation, 4,* 388–393.

[10] Wolfram, S. (2002). *Cellular Automata and Complexity,* Perseus Publishing.