Correspondence

Extracting Boolean Rules from CA Patterns

Yingxu Yang and S. A. Billings

***Abstract*—A multiobjective genetic algorithm (GA) is introduced to identify both the neighborhood and the rule set in the form of a parsimonious Boolean expression for both one- and two-dimensional cellular automata (CA). Simulation results illustrate that the new algorithm performs well even when the patterns are corrupted by static and dynamic noise.**

***Index Terms*—Boolean identification, cellular automata, genetic algorithms, spatio-temporal systems.**

I. Introduction

Cellular automata (CA) are mathematical models for complex natural systems containing large numbers of simple identical components with local interactions. Since the pioneering work of John von Neumann during the 1950s [1], cellular automata have been largely employed as a modeling class to approximate nonlinear discrete and continuous dynamical systems in a wide range of applications [2]–[5]. However the inverse problem of determining the CA that satisfies general sets of prespecified constraints [6] has received relatively little attention. One of the most essential problems in this case is the identification of CA, i.e., how to learn the underlying rule that governs the local behavior of cells from temporal slices of the global evolution of the spatio-temporal pattern.

CA in the classical sense are autonomous systems, i.e., there are no external inputs exerting an influence on the evolution. It is only possible to observe the evolution of CA as a series of snapshots of the global states at various times in a certain finite interval of the evolution. An identification procedure can then be established based on using these fixed global states. In CA identification, it is assumed that a given spatio-temporal pattern (a pattern that evolves in both space and time) Ω has a dimension d ($d \geq 1$) and can be described by a cellular automaton. Identification of a CA involves determining a minimal description of the CA Λ that precisely simulates Ω such that the size of the neighborhood of Λ is as small as possible. It is therefore necessary to determine not only the CA rule but also the structure of the CA neighborhood. Ideally, the identification technique should produce a concise expression of the CA rule. This ensures that the model is parsimonious and can be readily interpreted for hardware realization of the cellular automaton. Richards *et al.* [7] proposed a method for extracting CA rules from given spatio-temporal patterns using a genetic algorithm (GA). Adamatskii [8] discussed the complexity of identification of cellular automata and presented sequential and parallel algorithms for computing the local transition table. However, neither of these authors obtained a clear neighborhood structure or parsimonious rule expression.

In digital circuit design [9]–[14], small Boolean expressions are searched to reproduce given data tables. But these solutions are concerned with minimizing known Boolean functions and do not address the general problem of determining the CA rule from an observed possibly noisy complex multidimensional pattern. The latter problems are much more complex, as discussed by Richards *et al.* [7], and a solution would offer an important step forward in the modeling of spatially extended systems that arise in such diverse fields as pattern formation, fluidic mixing, brain imaging and in data compression problems.

This paper considers the identification of Boolean expressions for one–dimensional (1-D) and two–dimensional (2-D) CA. An evolutionary algorithm is proposed using a multiobjective genetic algorithm to extract a precise local Boolean expression of the CA rule from given spatio-temporal patterns blurred by noise. The remainder of the paper is organized as follows. In Section II, the definition of a group of 1-D and 2-D CA is introduced. Section III reformulates the Boolean expression for 1-D CA rules and extends these to the 2-D case. The GA search for Boolean expressions of CA rules is then presented with an emphasis on the construction of parsimonious forms of CA rules. Simulation results are contained in Section IV, and Section V discusses the efficiency of the algorithm.

II. One- and Two-Dimensional Cellular Automata

A cellular automaton is composed of three parts: a discrete lattice, a neighborhood and a rule for local transitions. Attention in this paper is restricted to binary CA where the cells can only take binary values.

A. Neighborhoods

The neighborhood of a cell is the set of all the cells capable of directly influencing the evolution. Sometimes this includes the cell itself. The neighborhoods shown in Fig. 1 are used often and have proper names. CA neighborhoods can take cells from various spatial and temporal scales. For simplicity, this paper only considers neighborhoods composed of cells from time step $t-1$, but the results are not restricted to this case.

B. Local Rules

The local transition rule updates all cells synchronously by assigning to each cell, at a given time step, a value which depends only on the neighborhood. The most common method to define the local transition rule is to use a transition table analogous to a truth table. In this truth table, the rule is labeled by assigning neighborhoods in ascending numerical order and specifying which neighborhoods map to zero and which to one. The truth table is also called the component form of the CA rule. The component form of a three-site 1-D rule R is shown as follows:

$$\begin{array}{cccccccc} 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\ r_0 & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7 \\ 2^0 & 2^1 & 2^2 & 2^3 & 2^4 & 2^5 & 2^6 & 2^7 \end{array}$$

where the first row gives all the possible states the cells within the neighborhood could take. The r_i's in the second row are the rule components which can take only binary numbers with 1's indicating that the components are included in the rule and 0's indicating otherwise. The last row shows the coefficients associated with the corresponding components. The rule R can be defined as

Manuscript received July 10, 1998; revised February 14, 1999 and April 23, 2000. This work was supported by the University of Sheffield and U.K. EPSRC. This paper was recommended by Associate Editor J. Oomen.

The authors are with the Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield S1 3JD, U.K. (e-mail: yingxu@acse.shef.ac.uk; s.billings@sheffield.ac.uk).

Publisher Item Identifier S 1083-4419(00)06717-0.

1083–4419/00$10.00 © 2000 IEEE

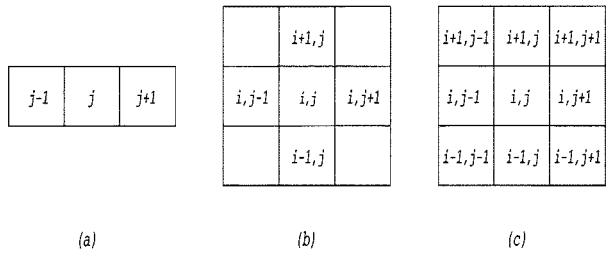


Fig. 1. Examples of most frequently used 1-D and 2-D neighborhoods. (a) 1-D von Neumann neighborhood. (b) 2-D von Neumann neighborhood. (c) 2-D Moore neighborhood.

$R = (r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7)$. The numerical label D assigned to R is given by $D(R) = \sum_{s=0}^{2^3-1} r_s 2^s$, which is simply the sum of the coefficients associated with all nonzero components. All this applies to the rule R with any three-site neighborhoods irrespective of the neighborhood structures. For example, a very often used 1-D three-site rule $Rule22$ is defined as $Rule22 = (01\,101\,000)$ and the numerical label $D(Rule22) = 2^1 + 2^2 + 2^4 = 22$.

III. Extracting Boolean Rules Using Genetic Algorithms

A. Boolean Form of CA Rules

For a 2-D CA, the local transition can be denoted as $s_{new}(i, j) = f_{lt}(\boldsymbol{N}(i, j))$, where $s_{new}(i, j)$ is the updated state of $cell(i, j)$ at time step t and $\boldsymbol{N}(i, j)$ represents the states of the cells within the neighborhood of $cell(i, j)$ at time step $t - 1$. Reducing the dimensions in this expression will yield models for 1-D CA as a special case. The local transition function f_{lt} is equivalent to the local transition table of length 2^n, where n is the size of the neighborhood. Therefore, f_{lt} can be viewed as a Boolean function of n variables, and CA rules can be expressed as Boolean rules. There are two different ways of constructing Boolean rules, using *NOT*, *AND*, *OR*, and *NOT*, *AND*, *XOR* logical operators, respectively. This paper only considers the latter formulation because this usually involves fewer logical operations than the former formulation for the same rule. For example, the Boolean form of $Rule22 = (01\,101\,000)$, which is illustrated in Table I, is $s_{new}(j) = C \oplus D$. For all n-site 1-D CA rules, there is a simple procedure for the construction of the Boolean expression in terms of *NOT*, *AND* and *XOR* operators given the component form (see [15], Algorithm 1.1, p. 30 for details). The *NOT* operator can be removed from the obtained Boolean expression by applying $NOT(a) = 1 \oplus a$. Furthermore, applying $0 \oplus a = a$ and $(a \oplus b) * c = a * c \oplus b * c$, it is then straightforward to state, e.g., that all the 1-D three-site binary CA can be represented by a Boolean function with only *AND* and *XOR* operators of the form

$$\begin{aligned} s_{new}(j) = {} & a_0 \oplus a_1 s(j-1) \oplus a_2 s(j) \oplus a_3 s(j+1) \\ & \oplus a_4(s(j-1) * s(j)) \oplus a_5(s(j) * s(j+1)) \\ & \oplus a_6(s(j-1) * s(j+1)) \\ & \oplus a_7(s(j-1) * s(j) * s(j+1)) \end{aligned} \tag{1}$$

where a_i $(i = 0, \cdots, 7)$ are binary numbers and $a_i = 1$ indicates that the following term is included in the Boolean function while $a_i = 0$ indicates that the following term is not included. Note that the number of possible expressions in (1) is $2^8 = 256$, which is exactly the number of all three-site 1-D rules. This implies that the representation in (1) is unique: one set of $\{a_i, i = 0, \cdots, 7\}$ corresponds to one and only one CA rule.

TABLE I
THE BOOLEAN FORM OF $Rule22$ USING *NOT*, *AND*, AND *XOR* OPERATORS. $B = s(j-1) \oplus s(j)$, $C = B \oplus s(j+1)$, $D = s(j-1) * s(j) * s(j+1)$—$*$, AND $\oplus$ REPRESENT THE *AND* AND *XOR* OPERATOR, RESPECTIVELY

$\mathbf{N}(j)$	$s_{new}(j)$	B	C	D	$C \oplus D$
000	0	0	0	0	0
001	1	0	1	0	1
010	1	1	1	0	1
011	0	1	0	0	0
100	1	1	1	0	1
101	0	1	0	0	0
110	0	0	0	0	0
111	0	0	1	1	0

This can be extended to multidimensional CA. For example any 2-D CA with a five-site von Neumann neighborhood can be represented by a Boolean expression:

$$\begin{aligned} s_{new}(i, j) = {} & a_0 \oplus a_1 s(i-1, j) \oplus \cdots \\ & \oplus a_{31}(s(i-1, j) * \cdots * s(i+1, j)). \end{aligned} \tag{2}$$

Extending this further, every CA with an n-site neighborhood $\{cell(x_1), \cdots, cell(x_n)\}$ may be written as

$$s_{new}(x_j) = a_0 \oplus a_1 s(x_1) \oplus \cdots \oplus a_P(s(x_1) * \cdots * s(x_n)) \tag{3}$$

where $P = 2^n - 1$ and $cell(x_j)$ is the cell to be updated.

B. Extracting Boolean Rules using a Genetic Algorithm

Equation (3) is important because it significantly reduces the complexity of CA identification by using a reduced set of logical operators and candidate terms. The difficulty in identifying multidimensional CA is also decreased because higher-dimensional CA rules are reduced to an equation which depends on the size of the neighborhood rather than the dimensionality. Assume that the only *a priori* knowledge is the dimension of the CA, which can be obtained from examining the spatio-temporal patterns [7]. Then the emerging difficulty lies in how to determine which terms should be included in the Boolean expression and which should be discarded. The problem is very similar to the term selection problem in structure detection for nonlinear system identification [16]. However, in CA identification all the terms are combined by the *XOR* operator and are therefore nonlinear-in-the-parameters, whereas in nonlinear system identification all the items are combined by the ordinary addition operator and can often be configured to be linear-in-the-parameters. This difference increases the difficulty of the CA term selection problem.

Note that CA term selection is different from and more difficult than Boolean function minimization for which many useful techniques are available, especially in the digital circuit design literature [9]–[14]. CA term selection involves both identifying and minimizing Boolean functions while methods related to Boolean function minimization usually only consider deriving the equivalent minimum expression from already known Boolean functions. The identification is difficult because of the logical terms, particularly when cellular automata with large size neighborhoods and, therefore, a large number of candidate logical terms, are involved.

In the present study, this problem is solved by evolving a genetic algorithm [17], [18] in the search for appropriate terms through the space of logical models constructed upon *AND* and *XOR* operators. This algorithm is implemented as follows.

1) Population: In the current application, each candidate Boolean rule is encoded using a chromosome. Each candidate term is then represented by a bit in this chromosome. The ith chromosome is defined

as a $1 \times (P+1)$ binary vector c_i, where $P = 2^n - 1$ and n is the size of the largest neighborhood under consideration. Each entry in the ith chromosome corresponds to a term in the following term set:

$$\begin{aligned} &c_i(1) \to 1,\ c_i(2) \to s(x_1),\ \cdots,\ c_i(n+1) \to s(x_n), \\ &c_i(n+2) \to s(x_1) * s(x_2),\ \cdots, \\ &c_i(P+1) \to s(x_1) * \cdots * s(x_n) \end{aligned}$$

where $c_i(j) = 1$ indicates that the associated term has been selected and $c_i(j) = 0$ indicates otherwise. Define

$$f = [1 \quad s(x_1) \quad \cdots \quad s(x_1) * \cdots * s(x_n)] \quad C = [c_1 \quad \cdots \quad c_{np}]$$

where np is the number of chromosomes in the population.

2) Multiobjective Fitness Function: The fitness function is designed to measure the performance of Boolean rules represented by the chromosomes in regenerating the observed spatio-temporal patterns. Before introducing the fitness function for the CA term selection problem, the vector *XOR* operator $\oplus$ is defined as follows:

$$\begin{aligned} &[\, a_1 \quad \cdots \quad a_n \,] \oplus [\, b_1 \quad \cdots \quad b_n \,]^T \\ &\quad = (a_1 * b_1) \oplus \cdots \oplus (a_n * b_n) \end{aligned}$$

where $*$ denotes the *AND* operator. For example,

$$\begin{aligned} &[\, a_1 \quad 0 \quad a_3 \,] \oplus [\, b_1 \quad b_2 \quad b_3 \,]^T \\ &\quad = (a_1 * b_1) \oplus (0 * b_2) \oplus (a_3 * b_3) = (a_1 * b_1) \oplus (a_3 * b_3). \end{aligned}$$

In the present problem, an important measure of the performance of the chromosomes is the modulus of errors function defined as

$$Mer(i) = \sum_{r}^{SET} |y(i,j) - \hat{y}(i,j)| \tag{4}$$

where

r	number of data points in the data set extracted from the CA patterns;
$y(i,j)$	measured state at data point j for chromosome i;
$\hat{y}(i,j) = c_i \oplus f_j$	predicted state.

If Mer is chosen as the fitness function, a solution with the least modulus of errors will be found. However, it is not guaranteed that the associated neighborhood is correct and minimal. This is because there may be more than one model that produces a minimum Mer. The initial assumed neighborhood almost always encompasses more cells than are actually in the real neighborhood. A 1-D rule $Rule30$ with a von Neumann neighborhood, for example, is most concisely represented as

$$s_{new}(j) = s(j-1) \oplus s(j) \oplus s(j+1) \oplus (s(j) * s(j+1)) \tag{5}$$

while during the search for this rule the candidate rules are often searched for over a larger neighborhood. For example, candidate rules can be assumed to be operating on a five-site neighborhood $\{cell(j-2), cell(j-1), cell(j), cell(j+1), cell(j+2)\}$ and of the form

$$\begin{aligned} s_{new}(j) = a_0 &\oplus a_1 s(j-2) \oplus \cdots \\ &\oplus a_{31}(s(j-2) * \cdots * s(j+2)). \end{aligned} \tag{6}$$

It is highly likely that more than one expression including (5) may be selected from (6) to match the patterns. The solution is therefore not necessarily unique, and this often leads to a false extension of the neighborhood.

Note that a rule with a larger neighborhood cannot be represented by a rule with a smaller neighborhood while a rule with a smaller neighborhood can often be expressed by a rule with a larger neighborhood. Therefore, the best model is always the smallest model amongst all the possible models chosen. This is the principle of parsimony. Thus, another search objective must be added to direct the algorithm to produce a parsimonious logical model with minimal modulus of errors. An efficient approach to accommodate the second search objective is to employ a multiobjective fitness function. In the present study, the two search objectives are to minimize Mer and to minimize the number of terms in all models with the same Mer. The multiobjective fitness function in this study is based on a ranking scheme according to the concept of Pareto optimality [17]. This will guarantee equal probability of reproduction to all nondominant chromosomes and should generate a solution nearest to the optimal. The multiobjective fitness function is constructed as follows.

a) Each chromosome in the current population is ranked with respect to Mer. The chromosome with the least Mer occupies the first position, the chromosome with the second least Mer occupies the second position and so on. Chromosomes with the same error share the same rank. So

RANK	1	$\cdots$	i	i
ERROR	$Mer(1)$	$\cdots$	$Mer(i)$	$Mer(i+1)$
RANK	i	$\cdots$	np	
ERROR	$Mer(i+2)$	$\cdots$	$Mer(np)$	

with

$$\begin{aligned} Mer(1) < \cdots < Mer(i) &= Mer(i+1) \\ &= Mer(i+2) < \cdots < Mer(np). \end{aligned}$$

b) Define the structure function $St(i)$ for the ith chromosome as $St(i) = \sum_j^{P+1} c_i(j)$. Resort the orders of chromosomes sharing the same rank in proportion to the associated $St(i)$ and keep the ranking of the remainder unchanged. Thus

RANK	1	$\cdots$	i	$i+1$
ERROR	$Mer(1)$	$\cdots$	$Mer(i)$	$Mer(i+1)$
STRUCTURE	$St(1)$	$\cdots$	$St(i)$	$St(i+1)$
RANK	$i+1$	$\cdots$	np	
ERROR	$Mer(i+2)$	$\cdots$	$Mer(np)$	
STRUCTURE	$St(i+2)$	$\cdots$	$St(np)$	

with

$$St(1) < \cdots < St(i) < St(i+1) = St(i+2) < \cdots < St(np).$$

c) The multiobjective fitness function of the ith chromosome is finally defined as

$$fit(i) = \frac{\mathrm{MAX}(rank(i)) - rank(i)}{\mathrm{MAX}(rank(i)) - \mathrm{MIN}(rank(i))}. \tag{7}$$

The ranking technique results in a search with a preference toward minimizing Mer. The structure function St will not have any impact on the first few steps of the search and the fitness of each chromosome in those steps is determined exclusively by Mer because all the chromosomes are highly likely to hold various ranks at that initial stage. Only after certain chromosomes have converged to a similar Mer is it possible to rearrange the ranking at that Mer according to the associated St. This search is therefore able to always select chromosomes with the minimum structure within the span of the lowest Mer. Hence, chromosomes with a parsimonious logical expression and minimum error will remain in the latest population to yield the final solution.

To avoid the algorithm becoming trapped at a local optima, two subpopulations will be introduced which evolve in parallel with the main population [19]. The subpopulations are evolved separately under two

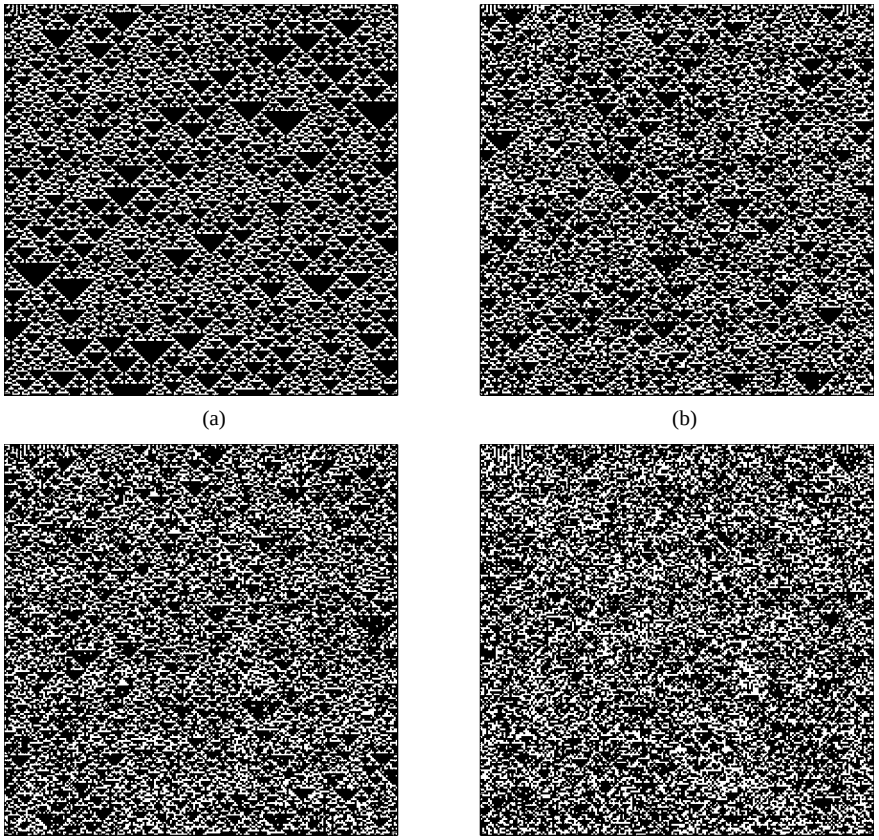


Fig. 2. Noise-free pattern and static noise contaminated patterns produced by 1-D $Rule22$. (a) $p = 0$. (b) $p = 0.05$. (c) $p = 0.1$. (d) $p = 0.2$.

different search objectives of minimizing Mer and St, respectively. Each candidate in the main population is produced by genetic communication between the two subpopulations and is subject to evaluation by the ranking technique.

For details of reproduction, crossover, and mutation see [18] and [20].

3) Summary of the CA Term Selection Algorithm: The CA term selection algorithm can be summarized as follows.

a) Generate three initial population sets $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$ with each consisting of np individuals. Set the current generation number $i = 1$.
b) Compute Mer for each individual in $\mathcal{P}$. Calculate St in each chromosome in $\mathcal{Q}$. Rank the individuals in $\mathcal{R}$ using the ranking technique. Calculate the fitness value.
c) Apply the parent selection technique to $\mathcal{P}$ and $\mathcal{Q}$.
d) Employ crossover and mutation to $\mathcal{P}$ and $\mathcal{Q}$ separately to produce the corresponding offspring sets $\mathcal{P}'$ and $\mathcal{Q}'$. Employ crossover and mutation to $\mathcal{P}$ and $\mathcal{Q}$ combined to produce the offspring set $\mathcal{R}'$ for the population set $\mathcal{R}$.
e) Calculate the corresponding fitness values for the chromosomes in the offspring sets $\mathcal{P}'$, $\mathcal{Q}'$ and $\mathcal{R}'$. Select the np fittest individuals from both the population sets $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{R}$ and the corresponding offspring sets $\mathcal{P}'$, $\mathcal{Q}'$ and $\mathcal{R}'$, respectively, by comparing the fitness values. Reset $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{R}$ using the corresponding newly selected np individuals. Set the generation number $i = i + 1$ and nullify the three offspring sets $\mathcal{P}'$, $\mathcal{Q}'$ and $\mathcal{R}'$.
f) Return to c) and repeat until a prespecified number of generations has been reached.

IV. Simulation Analysis

A. The Effects of Noise

In cellular automata, noise is a form of imperfection which at a critical magnitude is able to induce an essential phase transition that can suddenly change the behavior of the CA. Static noise can be added to a spatio-temporal pattern by first evolving a deterministic CA rule and then randomly flipping a limited number of binary values according to a specified probability p, where $p = q_1/q_2$, q_1 is the number of cells to be flipped, and q_2 is the total number of cells in the spatio-temporal pattern. This is referred to as static noise because it is added after the CA evolution. Fig. 2 shows the noise-free pattern ($p = 0$) for the 1-D $Rule22$ with von Neumann neighborhood and the same pattern corrupted by noise with probabilities of switching $p = 0.05$, $p = 0.1$, and $p = 0.2$, respectively. All these were developed on a 200 × 200 lattice with time evolution from top to bottom, and with a periodic boundary condition, i.e., the lattice is taken as a circle in the horizontal dimension, so the first and last sites are identified as if they lay on a circle of finite radius. The evolution started from an initial condition of a randomly generated binary vector.

Unlike static noise, which is added to the CA patterns after the evolution, dynamic noise is directly involved in the development of the spatio-temporal patterns by specifying that one or more (not all) of the rule components of the rule be 1 with a probability p and be 0 with probability $(1 - p)$, where $p = w_1/w_2$, w_1 is the number of the prespecified rule components to be filled in by 1, and w_2 is the number of the prespecified rule components. This may exhibit a transition depending on the probability p. Fig. 3 shows the transition from a simple 1-D $Rule184$ with von Neumann neighborhood to a chaotic $Rule60$

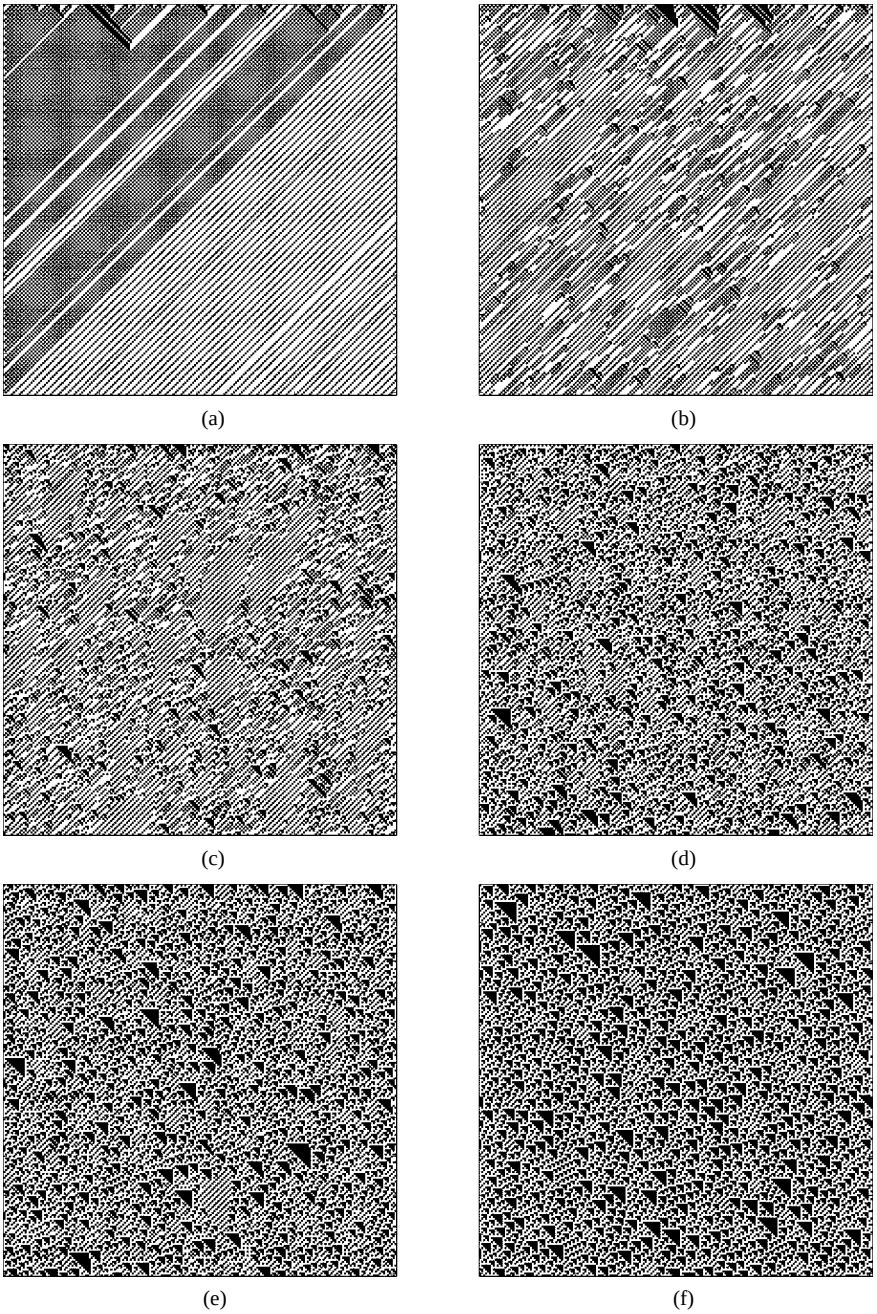


Fig. 3. The transition from $Rule184$ ($p = 0$) to $Rule60$ ($p = 1$) with p varying to indicate different noise densities (a) $p = 0$, (b) $p = 0.1$, (c) $p = 0.5$, (d) $p = 0.65$, (e) $p = 0.8$ and (f) $p = 1.0$.

under the transition rule in Table II. Note that the maximum noise density for the transition is 0.5. For example, when $p = 0.6$, the transition behaves more like $Rule60$ than $Rule184$ and, therefore, the noise density for the transition should be considered as $1 - p = 0.4$ for $Rule60$ rather than 0.6 for $Rule184$.

B. Extracting Boolean Rules from 1-D CA Patterns

1) From Patterns Corrupted by Static Noise: Assume the neighborhood structure of a class of 1-D CA is defined by $\{cell(j-2), cell(j-1), cell(j), cell(j+1), cell(j+2)\}$. Given the spatio-temporal patterns in Fig. 2, the CA term selection algorithm was used to produce the results in Table III. A typical run using data extracted from the noise-free pattern in Fig. 2(a) is shown in Fig. 4. In each case, 100 trials were conducted using different initial assignments. The program was terminated after 120 generations had been reached. Except for method a with $p = 0$, the optimal solutions were found in all the 100 trials. It can be seen from the results in Table III that when $p = 0$, method a tended to find suboptimal solutions for the neighborhood and the rule. This is also illustrated in Fig. 4(a), where although Mer has finally settled to zero, St shows no sign of further decreasing through the remaining generations. The Boolean rules found using method a varied

TABLE II
TRANSITION RULE FROM $Rule184$ TO $Rule60$ WHERE $0 < p < 1$

$Neighbourhoods$	000	001	010	011	100	101	110	111
Rule184	0	0	0	1	1	1	0	1
$Transition$	0	0	p	1	1	1	0	$1-p$
Rule60	0	0	1	1	1	1	0	0

TABLE III
SUMMARY OF RESULTS OBTAINED IN EXTRACTING $Rule22$ FROM THE PATTERNS IN FIG. 2, USING METHOD a: AN UNMODIFIED GENETIC ALGORITHM WITH Mer AS THE FITNESS FUNCTION, AND METHOD b: THE CA TERM SELECTION ALGORITHM. THE "GENERATIONS" COLUMN INDICATES THE NUMBER OF GENERATIONS REACHED WHEN THE OPTIMAL SOLUTION WAS FOUND. AV.R.T. REPRESENTS THE AVERAGE RUN TIME IN THE SEARCH FOR THE OPTIMAL SOLUTION IN ONE TRIAL. ONE HUNDRED TRIALS WERE MADE FOR EACH PROBLEM. THE OPTIMAL SOLUTION IS ALREADY KNOWN

method	p	criteria	min	max	mean	std.dev.	av.r.t.
a	0	generations	40	87	65.92	12.14	45.61 *min.*
		modulus of errors	0	0	0	0	
		structure	8	15	10.83	1.82	
b	0	generations	8	36	22.54	9.01	24.52 *min.*
		modulus of errors	0	0	0	0	
		structure	4	4	4	0	
b	0.1	generations	12	45	26.18	16.25	28.16 *min.*
		modulus of errors	9	14	11.10	1.01	
		structure	4	4	4	0	
b	0.2	generations	39	101	75.25	28.44	59.86 *min.*
		modulus of errors	18	31	24.63	5.81	
		structure	4	4	4	0	

with different trials, but all involved an incorrectly extended neighborhood. For example, the rule produced from the evolution shown in Fig. 4(a) is

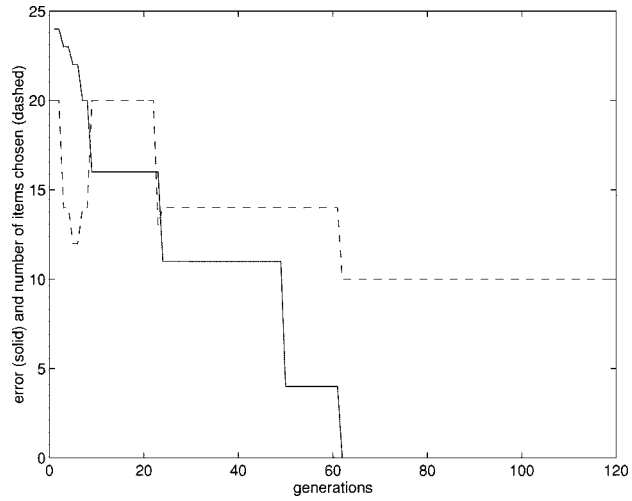
$$\begin{aligned} s_{new}(j) = {} & s(j-1) \oplus s(j) \oplus s(j+1) \oplus (s(j-1) * s(j+1)) \\ & \oplus (s(j-1) * s(j) * s(j+2)) \\ & \oplus (s(j-1) * s(j+1) * s(j-2)) \\ & \oplus (s(j-1) * s(j+1) * s(j+2)) \\ & \oplus (s(j-1) * s(j) * s(j+1) * s(j-2)) \\ & \oplus (s(j-1) * s(j) * s(j-2) * s(j+2)) \\ & \oplus (s(j-1) * s(j+1) * s(j-2) * s(j+2)). \end{aligned} \tag{8}$$

In contrast, when $p = 0$ method b generated only one Boolean rule in all 100 trials, this Boolean rule is

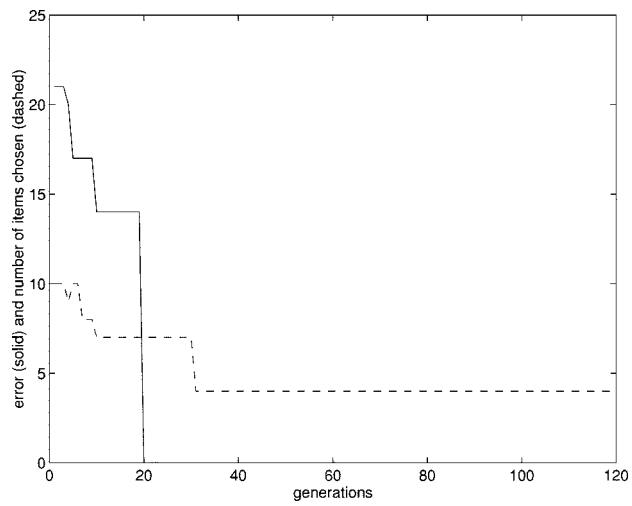
$$s_{new}(j) = s(j-1) \oplus s(j) \oplus s(j+1) \oplus (s(j-1) * s(j) * s(j+1)). \tag{9}$$

Although the Boolean rules in both (8) and (9) produce the same correct truth table of $Rule22$, the former covers a five-site neighborhood $\{cell(j-2), cell(j-1), cell(j), cell(j+1), cell(j+2)\}$ while the real neighborhood should be $\{cell(j-1), cell(j), cell(j+1)\}$. The neighborhood structure of the former is therefore suboptimal. This result was obtained because method a selected the rules with the minimum Mer without considering the structure of the neighborhood. In contrast, the Boolean rule in (9) was obtained using a search also taking into account the number of terms in the candidate rule, and this produced a rule exactly the same as listed in [15] with a correct neighborhood and a parsimonious Boolean expression. This is also illustrated in Fig. 4(b) where St diminishes after Mer has settled on zero. Furthermore, the modulus of errors using method b converged considerably faster than using method a because in method b, subpopulations were incorporated to accelerate the convergence. This effect is demonstrated in Table III, where the average run-time using method b was almost half the time using method a. It can also be seen from Table III that although Mer did not converge to zero for either $p = 0.1$ or $p = 0.2$, due to random flipping of some cell values, the correct and parsimonious Boolean expression in (9) was still obtained in both cases. This suggests that the CA term selection algorithm is not sensitive to static noise when the noise density is within a certain amplitude, in this case, $p \leq 0.2$.

2) From Patterns Corrupted by Dynamic Noise: Dynamic noise is immediately involved in the CA evolution and can considerably affect the development. The patterns contaminated by dynamic noise are therefore much more complicated. Ideally, an identification procedure should be designed to remain insensitive to these disturbances and to recover the underlying rule in as much is possible. Table IV shows the results of a search for the appropriate CA rule using data generated in the transition from $Rule184$ to $Rule60$ shown in Fig. 3. In each case, 100 trials were conducted using different initial assignments. The program was terminated when 200 generations had been reached. For simplicity, only the corresponding average and standard deviation are listed in Table IV. The search result for $p = 0$ in Table IV produces



(a)



(b)

Fig. 4. Search process for $Rule22$ using the noise-free pattern, and methods a and b, respectively, (a) using method a and (b) using method b. Legend: The dashed line is the evolution of the number of terms in the chromosome with the best fitness. The solid line is the evolution of the modulus of error of the chromosome with the best fitness.

TABLE IV
SUMMARY OF RESULTS OBTAINED IN EVOLVING THE TRANSITION FORM $Rule184$ TO $Rule60$ USING THE CA TERM SELECTION ALGORITHM

p	generations		modulus of errors		structure		av.r.t.
	mean	std.dev.	mean	std.dev.	mean	std.dev.	
0	41.13	11.12	0	0	3	0	50.28 *min.*
0.1	63.17	20.10	7.16	2.71	4	0	77.19 *min.*
0.65	27.88	9.19	24.32	4.41	2	0	28.31 *min.*
0.8	97.62	34.59	6.68	2.82	2	0	89.98 *min.*
1	123.65	36.16	0	0	2	0	121.64 *min.*

$s_{new}(j) = s(j-1) \oplus (s(j-1) * s(j)) \oplus (s(j) * s(j+1))$, which is exactly the same as the Boolean form of $Rule184$ listed in [15]. The result in Table IV for $p = 0.1$ produces

$$s_{new}(j) = s(j-1) \oplus (s(j-1) * s(j)) \oplus (s(j) * s(j+1)) \oplus (s(j-1) * s(j) * s(j+1))$$

which generates the rule components (01 011 100). Note that in comparison with the correct rule components for $Rule184$ (01 011 101), the last bit in the identified rule components is incorrect. The reason for this appears to lie in the nature of the rule itself. Rules that produce simple patterns of self-repetition or shifting imply that a certain number of combinations of the states of the cells within the neighborhood have probably never appeared in the existing data set. Hence, the search is unlikely to learn that particular behavior and most probably indiscriminately selects a rule from a range of rules satisfying only other combinations. For $Rule184$, even though the evolution is slightly complicated, and the data set extracted from the noise-free pattern is rich enough to allow the identification of the correct rule (00 011 101), the combination of the states of the cells within the neighborhood (111), which corresponds to the last component, appears infrequently and the noise simplifies the data set by eliminating (111). Subsequently, the search failed to learn the behavior of (111) and, hence, produced a wrong result. A similar phenomena appears in other simple 1-D rules such as $Rule46$, $Rule116$, $Rule72$, and $Rule172$. The search may repeatedly produce incorrect rules even after the size of the data set has been enlarged. Consequently, under certain combination of rules and initial conditions, the optimal solution can be deduced only provided sufficient data are available.

However, for $p = 0.8$, where the noise density is $1 - 0.8 = 0.2$ for the rule transition, the search result is $s_{new}(j) = s(j-1) \oplus s(j)$. This result is correct and parsimonious and exactly the same as the Boolean expression for $Rule60$ listed in [15], even though the noise level is 10% higher than in $p = 0.1$. This appears to be because of the chaotic nature of $Rule60$, which is able to generate patterns complicated enough so that even after the patterns are contaminated by a relatively high density of noise, the data set still contains sufficient information to correctly characterize the behavior. When $p = 1$, the result is $s_{new}(j) = s(j-1) \oplus s(j)$, which is the same as for $p = 0.8$ and is the parsimonious Boolean expression for $Rule60$. Note that for $p = 0.65$, the search also converged to the correct result $s(j) = s(j-1) \oplus s(j)$ despite the high noise density of $1 - 0.65 = 0.35$ for $Rule60$.

The search result for the transition rule at $p = 0.8$ is also shown in Table V. Although the data were contaminated by a dynamic noise with density $1 - 0.8 = 0.2$, which had substantial impact on the growing patterns (the growth of the noise corrupted patterns was 100% faster than the noise-free patterns), and the error did not converge to zero, the search produced the same correct and parsimonious Boolean rule as identified in the noise-free case ($p = 1$). This suggests that the CA term selection algorithm is robust in the presence of dynamic noise even for the 2-D case.

C. Extracting Boolean Rules from 2-D CA Patterns

Table V shows the results of the search for CA rules using data produced by both the evolution of a deterministic 2-D rule $Rule(01\,101\,010 \quad 11\,101\,110 \quad 01\,001\,111 \quad 11\,100\,100)$ and a transition $(01\,101\,010 \quad 1\,110\,111(1-p) \quad 01\,001\,111 \quad p1\,100\,100)$ of the same rule at $p = 0.8$ over the five-site von Neumann neighborhood in Fig. 1(b). The neighborhood was initially assumed as the nine-site Moore neighborhood in Fig. 1(c). In each case, 100 trials were conducted using different initial assignments. The program was terminated when 800 generations had been reached. The result in Table V for $p = 1$ produces a Boolean expression of the *XOR* combination of the following 20 terms:

TABLE V
SUMMARY OF RESULTS OBTAINED IN EXTRACTING A 2-D CA RULE AND THE SAME RULE CORRUPTED BY A DYNAMIC NOISE/IN TRANSITION AT $p = 0.8$ USING THE CA TERM SELECTION ALGORITHM

p	generations		modulus of errors		structure		av.r.t.
	mean	std.dev.	mean	std.dev.	mean	std.dev.	
1	513.16	49.67	0	0	20	0	8.25 *hr.*
0.8	600.36	57.25	6.40	1.63	20	0	9.64 *hr.*

$$\begin{aligned}
B &= s(i, j-1), \quad C = s(i, j)\\
D &= s(i, j+1), \quad E = s(i-1, j)\\
F &= s(i+1, j) * s(i, j+1)\\
G &= s(i, j-1) * s(i, j)\\
H &= s(i, j-1) * s(i, j+1)\\
I &= s(i, j-1) * s(i+1, j)\\
J &= s(i, j) * s(i, j+1)\\
K &= s(i+1, j) * s(i, j-1) * s(i, j)\\
L &= s(i+1, j) * s(i, j-1) * s(i, j+1)\\
M &= s(i+1, j) * s(i, j) * s(i, j+1)\\
N &= s(i+1, j) * s(i, j) * s(i-1, j)\\
O &= s(i+1, j) * s(i, j+1) * s(i-1, j)\\
P &= s(i, j-1) * s(i, j) * s(i, j+1)\\
Q &= s(i, j) * s(i, j+1) * s(i-1, j)\\
R &= s(i+1, j) * s(i, j-1) * s(i, j) * s(i, j+1)\\
S &= s(i+1, j) * s(i, j-1) * s(i, j+1) * s(i-1, j)\\
T &= s(i+1, j) * s(i, j) * s(i, j+1) * s(i-1, j)\\
U &= s(i+1, j) * s(i, j-1) * s(i, j)\\
&\quad * s(i, j+1) * s(i-1, j).
\end{aligned}$$

Table VI shows the tabular form of this identified Boolean rule. It can be seen that this rule covers a five-site von Neumann neighborhood, which has been extracted from the assumed nine-site Moore neighborhood and is correct. The truth table produced from the identified rule is also correct. Therefore, according to the principle of parsimony, the identified Boolean rule is optimal. However as shown in Table V the average run time is much longer than in the 1-D noise-free case. This is because the growth of the size of the neighborhood will inevitably induce a considerable increase in the number of possible terms that can be included in the Boolean expression and, hence, an increase in the complexity of the search.

D. Extracting Boolean Rules from Patterns Produced by a Large Set of CA Rules

The CA term selection algorithm was tested over a large set of CA rules with various neighborhoods of randomly chosen radius. Some of the 1-D results are summarized in Table VII. Because the numerical label and the component form of the rules grow astronomically with even a slight increase in the size of the neighborhood, only rules with relatively small neighborhoods were specified. For each rule, both the noise-free case and the case where the pattern has been corrupted by a static noise at $p = 0.2$ were considered. One hundred runs were conducted for each problem, and the search was terminated after 1000 generations. The starting neighborhood was assumed as a nine-site neighborhood $\{cell(j-4), cell(j-3), cell(j-2), cell(j-1),$

TABLE VI
TABULAR FORM OF THE IDENTIFIED 2-D BOOLEAN RULE

$\mathbf{N}(i,j)$	$s_{new}(i,j)$	B	$\oplus C$	$\oplus D$	$\oplus E$	$\oplus F$	$\oplus G$	$\oplus H$	$\oplus I$	$\oplus J$	$\oplus K$	$\oplus L$	$\oplus M$	$\oplus N$	$\oplus O$	$\oplus P$	$\oplus Q$	$\oplus R$	$\oplus S$	$\oplus T$	$\oplus U$
00000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00001	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
00010	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
00011	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00100	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
00101	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
00110	1	0	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
00111	0	0	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
01000	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
01001	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
01010	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
01011	0	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	0	0	0	0	0
01100	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
01101	1	1	0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
01110	1	1	0	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	1	1
01111	0	1	0	1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0
10000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10001	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
10010	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10011	0	0	0	1	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
10100	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
10101	1	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
10110	1	0	1	0	0	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1
10111	1	0	1	0	1	0	0	0	0	1	1	1	0	1	0	0	0	0	0	1	1
11000	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
11001	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
11010	1	1	1	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1
11011	0	1	1	0	1	0	0	1	0	0	0	1	1	1	0	0	1	1	0	0	0
11100	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
11101	1	1	0	0	1	1	0	0	1	1	0	0	0	1	1	1	1	1	1	1	1
11110	0	1	0	1	1	0	1	0	0	1	0	1	0	0	0	1	1	0	0	0	0
11111	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

TABLE VII
SUMMARY OF RESULTS OBTAINED IN EVOLVING SOME 1-D CA RULES WITH VARIOUS SIZES OF NEIGHBORHOOD. n INDICATES THE SIZE OF THE NEIGHBORHOOD

n	rule	p	generations		modulus of errors		structure		av.r.t.
			mean	std.dev.	mean	std.dev.	mean	std.dev.	
3	$Rule22$	0	347.15	38.24	0	0	4	0	5.27 $hr.$
		0.2	219.27	35.11	12.13	4.98	4	0	3.48 $hr.$
	$Rule54$	0	230.49	29.78	0	0	4	0	3.54 $hr.$
		0.2	253.46	27.14	13.25	6.44	4	0	3.79 $hr.$
4	$Rule179$	0	392.76	41.65	0	0	7	0	5.80 $hr.$
		0.2	440.03	40.32	15.58	5.12	7	0	6.57 $hr.$
	$Rule924$	0	346.82	31.59	0	0	9	0	5.35 $hr.$
		0.2	369.90	30.64	10.11	5.98	9	0	5.71 $hr.$
5	$Rule1$	0	488.16	43.38	0	0	15	0	7.63 $hr.$
		0.2	460.03	44.56	20.17	7.76	15	0	7.29 $hr.$
	$Rule2$	0	499.74	40.87	0	0	18	0	7.76 $hr.$
		0.2	538.92	42.31	15.68	6.99	18	0	8.32 $hr.$
6	$Rule3$	0	611.15	53.78	0	0	25	0	9.69 $hr.$
		0.2	572.26	50.51	23.85	7.74	25	0	8.91 $hr.$
	$Rule4$	0	633.50	41.16	0	0	19	0	10.24 $hr.$
		0.2	659.66	53.54	19.96	5.58	19	0	10.69 $hr.$
7	$Rule5$	0	554.59	46.78	0	0	16	0	8.63 $hr.$
		0.2	660.27	51.53	28.22	4.18	16	0	10.47 $hr.$
	$Rule6$	0	700.53	59.20	0	0	58	0	11.38 $hr.$
		0.2	651.84	32.72	30.11	8.52	58	0	10.31 $hr.$
8	$Rule7$	0	643.20	29.88	0	0	44	0	10.28 $hr.$
		0.2	602.33	27.84	25.14	7.30	44	0	9.81 $hr.$
	$Rule8$	0	568.29	32.25	0	0	30	0	8.72 $hr.$
		0.2	637.48	29.54	31.51	6.58	30	0	10.30 $hr.$
9	$Rule9$	0	710.35	65.47	0	0	52	0	11.49 $hr.$
		0.2	685.46	40.33	36.58	7.24	52	0	10.84 $hr.$
	$Rule10$	0	740.58	56.29	0	0	54	0	11.77 $hr.$
		0.2	690.43	48.59	34.20	7.95	54	0	10.96 $hr.$

$cell(j),\ cell(j+1), cell(j+2),\ cell(j+3),\ cell(j+4)\}$. The identified neighborhoods and the truth tables produced by the corresponding Boolean rules are all correct. Therefore, according to the principal of parsimony, the identified Boolean rules are all optimal.

Simulations over a large set of rules suggest that the average run time relies more on the size of the assumed neighborhood than on the size of the actual neighborhood. For example, for $Rule22$, the average run-time in Table VII, where the initial neighborhood was assumed as nine-site, is considerably larger than in Table III, where the initial neighborhood was assumed as five-site. However, as has also been indicated in Table VII, under the same neighborhood assumption, rules with more cells in the actual neighborhood tend to require more generations to converge to the optimal solution. Because the CA term selection algorithm discriminates only the number of cells in the neighborhood rather than the position each individual cell occupies, the results for 2-D rules were very similar to the 1-D case and are therefore not listed in this paper.

V. CONCLUSIONS

A solution to the inverse problem in cellular automata has been proposed using a multiobjective evolutionary algorithm. Both 1- and 2-D cellular automata have been investigated, and it has been shown that the CA rule, in the form of a parsimonious Boolean expression, can be identified by carefully formulating the GA search procedure. The simulation results illustrate the efficiency of the new algorithm and demonstrate that the correct neighborhood and CA rule can be determined in the presence of both static and dynamic noise.

REFERENCES

[1] J. von Neumann, "The general logical theory of automata," in *Cerebral Mechanisms in Behavior—The Hixon Symposium*. New York: Wiley, 1951.

[2] L. Brieger and E. Bonomi, "A stochastic cellular automaton simulation of the nonlinear diffusion equation," *Phys. D*, vol. 47, no. 1–2, pp. 159–168, 1992.

[3] G. Hernandez and H. J. Herrann, "Cellular automata for elementary image enhancement," *Graph. Models Image Process.*, vol. 58, no. 1, pp. 82–89, 1996.
[4] S. Bhattachariee and S. Sinha, "Cellular automata based scheme for solution of Boolean equations," *Proc. Inst. Elect. Eng. Comput. Digital Tech.*, vol. 143, no. 3, pp. 174–180, 1996.
[5] S. Surka and K. P. Valavanis, "A cellular automata model for edge relaxation," *J. Intell. Robot. Syst.*, vol. 4, no. 4, pp. 379–391, 1991.
[6] W. Li, N. H. Packard, and C. G. Langton, "Transition phenomena in cellular automata rule space," in *Cellular Automata: Theory and Experiment*, H. Gutowitz, Ed. Cambridge, MA: MIT Press, 1991.
[7] F. C. Richards, "Extracting cellular automaton rules directly from experimental data," *Phys. D*, vol. 45, pp. 189–202, 1990.
[8] A. I. Adamatskii, "Complexity of identification of cellular automata," *Autom. Remote Control*, pt. 2, vol. 53, no. 9, pp. 1449–1458, 1992.
[9] W. V. Quine, "A way to simplify truth functions," *Amer. Math. Monthly*, vol. 62, pp. 627–631, 1955.
[10] E. J. McCluskey Jr., "Minimization of Boolean functions," *Bell Syst. Tech. J.*, vol. 35, pp. 1417–1444, 1956.
[11] S. C. Gupta, "A fast procedure for finding prime implicants of a Boolean function," *Int. J. Electron.*, vol. 51, no. 5, pp. 701–709, 1981.
[12] S. R. Perkins and T. Rhyne, "An algorithm for identifying and selecting the prime implicants of a multiple-output Boolean function," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 1215–1218, Nov. 1988.
[13] P. P. Chu, "Applying neural networks to find the minimum cost coverage of a Boolean function," *VLSI Design*, vol. 3, no. 1, pp. 13–19, 1995.
[14] S. Chattopadhyay, S. Roy, and P. P. Chaudhuri, "KGPMIN: An efficient multilevel multioutput AND-OR-XOR minimizer," *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 257–265, Mar. 1997.
[15] B. H. Voorhees, *Computational Analysis of One-dimensional Cellular Automata*, ser. World Scientific Series on Nonlinear Science. Singapore: World Scientific, 1996.
[16] S. A. Billings, S. Chen, and M. J. Korenberg, "Identification of MIMO nonlinear systems using a forward-regression orthogonal estimator," *Int. J. Control*, vol. 49, no. 6, pp. 2157–2189, 1989.
[17] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. New York: Springer-Verlag, 1994.
[18] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.
[19] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.
[20] A. J. Chipperfield and P. J. Fleming, "Genetic algorithms in control systems engineering," *Control Comput.*, vol. 23, pp. 88–94, 1996.

The Certainty Factor-Based Neural Network in Continuous Classification Domains

LiMin Fu

***Abstract*—The integration of certainty factors (CFs) into the neural computing framework has resulted in a special artificial neural network known as the CFNet. This paper presents the cont-CFNet, which is devoted to classification domains where instances are described by continuous attributes. A new mathematical analysis on learning behavior, specifically linear versus nonlinear learning, is provided that can serve to explain how the cont-CFNet discovers patterns and estimates output probabilities. Its advantages in performance and speed have been demonstrated in empirical studies.**

***Index Terms*—Certainty factor, classification, fuzzy set theory, machine learning, neural network, probability estimation.**

I. Introduction

The certainty factor (CF) model has been applied to traditional expert systems for management of uncertainty [2], [17]. To map this model into a neural computing framework has generated a number of successful applications [4]–[8], [11], [13]. The term CFNet has been coined by Fu [8] to refer to an artificial neural network whose activation function is based on the CF model. An important result is that the CFNet can generalize more effectively than neural networks using the sigmoid activation function [7]. Recent research has shown that this network can discover the domain rules with better accuracy than the decision-tree approach [8]. Because certainty factors have traditionally been applied to rule-based environments for handling symbolic data, the CFNet has been tested in domains where instances are described by discrete nominal attributes. It would be a new challenge to apply the CFNet to domains where the data involve continuous numerical attributes. In order to realize this capability, a new system named the cont-CFNet exploits the strength of fuzzy representation. Here, a brief background on fuzzy neural networks is worthwhile.

Based on Zadeh's fuzzy set theory [19], fuzzy logic views each predicate as a fuzzy set. In fuzzy logic, a linguistic variable like "size" can have several linguistic values like "small," "medium," or "large." Each linguistic value is viewed as a fuzzy set associated with a membership function, which can be triangular, bell-shaped, or of another form. The degree of membership can be interpreted as the degree of *possibility*, which evades the requirement of satisfying the probability axioms.

The relationship between fuzzy systems and neural networks has drawn much attention since both are trainable systems capable of handling uncertainty and imprecision and both have found may successful applications. Their complimentary roles suggested in [10] and [14] have led to so-called fuzzy neural networks for handling the fuzzy (inexact) nature of inference involving symbols (symbolic inference). In such networks, an input pattern is enhanced via fuzzy representation. Furthermore, when the decision function or boundary involves curves, a smooth membership function at fuzzy neural units allows close modeling or approximation.

Manuscript received July 11, 1999; revised May 28, 2000. This work was supported by the National Science Foundation under Grant ECS-9615909. This paper was recommended by Associate Editor W. Pedrycz.

The author is with the Department of Computer and Information Sciences, University of Florida, Gainesville, FL 32611 USA (e-mail: fu@cise.ufl.edu).

Publisher Item Identifier S 1083-4419(00)06719-4.

1083–4419/00$10.00 © 2000 IEEE