Fuzzy Cellular Automata for Modeling Pattern Classifier

Pradipta MAJI[†a)] *and* P. Pal CHAUDHURI[†], *Nonmembers*

SUMMARY This paper investigates the application of the computational model of Cellular Automata (CA) for pattern classification of real valued data. A special class of CA referred to as Fuzzy CA (FCA) is employed to design the pattern classifier. It is a natural extension of conventional CA, which operates on binary string employing boolean logic as next state function of a cell. By contrast, FCA employs fuzzy logic suitable for modeling real valued functions. A matrix algebraic formulation has been proposed for analysis and synthesis of FCA. An efficient formulation of Genetic Algorithm (GA) is reported for evolution of desired FCA to be employed as a classifier of datasets having attributes expressed as real numbers. Extensive experimental results confirm the scalability of the proposed FCA based classifier to handle large volume of datasets irrespective of the number of classes, tuples, and attributes. Excellent classification accuracy has established the FCA based pattern classifier as an efficient and cost-effective solutions for the classification problem.

key words: *cellular automata (CA), fuzzy cellular automata (FCA), classifier, genetic algorithm (GA), decision tree*

1. Introduction

Over the years, the computational model of CA has been proposed to study the general phenomenological aspects, including communication, computation, construction, growth, reproduction, competition, and evolution [1], [2]. CA also provides an excellent tool for modeling physical phenomena by reducing them to their basic, elemental laws (rules) [3], [4]. From the days of Von Neumann [5] to the recent days of Wolfram's recent book 'A New Kind of Science' [6], CA has attracted researchers from diverse disciplines.

Interesting computational properties of CA model has inspired us to investigate new application avenues. Pattern classification is an important and interdisciplinary research area spanning several disciplines such as database systems [7], machine learning [8]–[10], intelligent information systems, statistics [11]–[13], and expert systems. Many new approaches are being introduced [14], [15], as well as existing ones getting refined [16]–[21]. However, the search for new and better solutions continues, specifically to classify large volume of dataset generated in the internet-worked society of cyber-age.

In the above background scenario, design of pattern classifier based on CA has been explored in a number of papers [22]–[24]. However, the classifier proposed in [22]–[24] can handle attributes expressed as binary patterns even though real life applications demand classification of data involving real numbers. For the classifiers designed to handle binary data, an explicit or implicit discretization procedure is applied to cluster the continuous data of real numbers to a set of subintervals, - that is, to transform the continuous attributes to discrete ones. Such a discretization may degrade the quality of solution. In fact, most discretization procedures suffer from user's bias in generating the subintervals [25], [26]. Also, since discretization is performed on a finite training set, it is doubtful whether the clustered subintervals encapsulate the real distribution. Moreover, since design of information-lossless discretization procedure is not available for real life problems, some information may be lost in the transformation from continuous domain to finite subintervals and that will invariably degrade the quality of solution.

In this paper, we present the computational model of FCA to address the problem of classification of patterns of real valued data. FCA is a conventional CA with fuzzy logic applied as next state function of a cell. The concept of fuzzy logic in automata theory was introduced more than thirty years ago [27]. FCA have been considered by several researchers in both theory and applications [28]–[31]. The evaluation of the global function of such FCA involves complex computation. One of the major motivations of the present research is to reduce such computational complexity of FCA. In this context, we make the following departure from the traditional FCA model.

- The evaluation of global transition function has been realized with matrix algebraic formulation.
- Since all the FCA rules are not amenable to such matrix algebraic formulation, we consider only a subset of FCA rules.

However, we have tried to keep as close as possible to the intrinsic characteristics (neighborhood dependency and local transition function) of the FCA model [28].

The proposed pattern classifier is built around a special class of FCA termed as Fuzzy Multiple Attractor CA (FMACA). It is a natural extension of boolean Multiple Attractor CA (MACA) based pattern classifier proposed in [22]–[24] to handle real valued patterns.

In order to realize the specified objectives, the paper introduces CA preliminaries including FCA fundamentals in Sect. 2. The matrix algebraic formulation for analysis and synthesis of FCA is proposed in Sect. 3. Section 4 cov-

[†]The authors are with the Department of Computer Science and Engineering & Information Technology, Netaji Subhash Engineering College, Kolkata, West Bengal, India, 700152.

a) E-mail: pradiptamaji@hotmail.com

ers a special class of FCA termed as FMACA. Design of FMACA based classifier is presented in Sects. 4.1 and 4.2 followed by its evolution with GA formulation in Sect. 5. Finally, the performance of the FMACA based classifier is reported in Sect. 6.

2. Cellular Automata (CA)

A CA [5], [32] consists of a number of cells organized in the form of a lattice. It evolves in discrete space and time. The next state of a cell depends on its own state and the states of its neighboring cells. In a 3-neighborhood dependency, the next state $q_i(t+1)$ of a cell is assumed to be dependent only on itself and on its two neighbors (left and right), and is denoted as

$$q_i(t+1) = f(q_{i-1}(t), q_i(t), q_{i+1}(t)) \tag{1}$$

where $q_i(t)$ represents the state of the i^{th} cell at t^{th} instant of time, f is the next state function and referred to as the rule of the automata. The decimal equivalent of the next state function, as introduced by Wolfram [32], is the rule number of the CA cell. In a 2-state 3-neighborhood CA, there are total 2^{2^3} - that is, 256 distinct next state functions (rules). Out of 256 rules, two rules 85 and 238 are illustrated below:

$$\begin{aligned} \text{Rule 85}: &\quad q_i(t+1) = \overline{q}_{i+1}(t) \\ \text{Rule 238}: &\quad q_i(t+1) = q_i(t) + q_{i+1}(t) \end{aligned}$$

where + indicates OR operation. An n-cell CA is configured with the rule vector $\mathcal{R} = <\mathcal{R}_1, \cdots, \mathcal{R}_i, \cdots, \mathcal{R}_n>$ where i^{th} cell is configured with rule $\mathcal{R}_i$; each $\mathcal{R}_i$ being one of the possible 256 rules.

Fuzzy Cellular Automata (FCA)

An elementary FCA [28], [29] is a linear array of cells which evolves in time. Each cell of the array assumes a state q_i, a rational value in the interval [0, 1] (fuzzy states) and changes its state according to a local evolution function on its own state and the states of its two neighbors. The global evolution results from the synchronous application of the local rule to all cells of the array. The degree to which a cell is in fuzzy states 1 and 0 can be calculated with the membership functions

$$\mu_1(q_i) = q_i;\ \mu_0(q_i) = 1 - q_i \tag{2}$$

respectively.

Definition 1: A FCA is obtained by fuzzification of the local function of a boolean CA as defined below. In a FCA, the conventional boolean functions, as reported in [28], [29], are evaluated as noted in Table 1. Here a and b are two states having rational values in the unit interval [0, 1]. The resulting local rule of FCA is a real valued function simulating the original function.

Table 1 Evaluation of Boolean function in FCA cell.

Boolean Function	Operation	FCA Operation
OR	$a + b$	$min\{1, a + b\}$
AND	ab	$a \cdot b$
NOT	$\overline{a}$	$(1 - a)$

3. Matrix Algebraic Formulation for FCA

In this section, a matrix algebraic formulation has been proposed to completely characterize FCA. The state transition behavior of FCA has been investigated with the help of this formulation. The important aspects dealt with includes the topological characterization of FCA, its basins of attraction, etc. The approach can be found to be more general in nature than that could be achieved with the help of traditional treatment of FCA rules reported in [28], [29]. The concepts developed in this section have been used to evolve a special class of FCA in Sect. 4 to design FCA based pattern classifier handling real valued attributes.

In the present work we consider the following FCA rules, as noted in Table 2 which employ only OR and NOR logic, where q_{i+1}, q_i and q_{i-1} represent the state of $(i+1)^{th}$, i^{th} and $(i-1)^{th}$ cells at t^{th} time instance. A few definitions are introduced which are used in the rest of this paper.

Definition 2: If all the cells of a FCA obey the same rule, then the FCA is said to be a uniform FCA; otherwise, it is a hybrid/non-uniform FCA.

Definition 3: A FCA is said to be a null boundary FCA if the left (right) neighbor of the leftmost (rightmost) cell is connected to logic 0-state.

Definition 4: A FCA involving rules with NOR logic is referred to as Complemented FCA.

3.1 Dependency Matrix for FCA

The rule as defined earlier, represents the local transition function of a FCA cell. For an n-cell FCA, the global transition function is represented by an n-tuple $\mathcal{R} = <\mathcal{R}_1, \cdots, \mathcal{R}_i, \cdots, \mathcal{R}_n>$ where $\mathcal{R}_i$ is a FCA rule applied on the i^{th} cell. For the class of FCA that employs the rules noted in Table 2, the linear operator expressing the global transition function can be uniquely represented by $n \times n$ square matrix, n being the number of cells in the FCA. The next state of the FCA can be obtained by premultiplying the vector representation of the present state by this matrix. The addition

Table 2 FCA rules (complemented and non-complemented).

Non-complemented Rules		Complemented Rules	
Rule	Next State	Rule	Next State
0	0	255	1
170	q_{i+1}	85	$\overline{q}_{i+1}$
204	q_i	51	$\overline{q}_i$
238	$q_i + q_{i+1}$	17	$\overline{q_i + q_{i+1}}$
240	q_{i-1}	15	$\overline{q}_{i-1}$
250	$q_{i-1} + q_{i+1}$	5	$\overline{q_{i-1} + q_{i+1}}$
252	$q_{i-1} + q_i$	3	$\overline{q_{i-1} + q_i}$
254	$q_{i-1} + q_i + q_{i+1}$	1	$\overline{q_{i-1} + q_i + q_{i+1}}$

(OR) operation implies that $(a + b) = min\{1, (a + b)\}$[28].

Let the FCA global transition function, represented by the $n \times n$ (for n-cells) matrix, be referred to as the dependency matrix of the FCA. The i^{th} row designates a rule being applied on the i^{th} cell. For a 3-neighborhood FCA there are atmost 3 non-zero entries in any row of the matrix. The following example illustrates the above discussion.

Example 1: A 4-cell null boundary (Definition 3) hybrid FCA (Definition 2) with the following rule vector $< 238, 254, 238, 252 >$ (that is, $< (q_i + q_{i+1}), (q_{i-1} + q_i + q_{i+1}), (q_i + q_{i+1}), (q_{i-1} + q_i) >$) applied from left to right, may be characterized by the following dependency matrix

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

The first row represents the rule $(q_i + q_{i+1})$, etc. Presence (absence) of dependency is denoted as 1 (0) in the binary matrix.

The following subsection highlights the elegance of such a matrix representation.

3.2 Characterization of the State Transition Behavior

The following theorem formalizes the proof of correctness of matrix algebraic formulation for FCA configured with the rules enlisted in Table 2.

Theorem 1: If $\mathcal{P}(t)$ represents the state assignment of the cells at the t^{th} instant of time, then the state at the next instant can be represented by the state transition equation:

$$\mathcal{P}(t + 1) = T \cdot \mathcal{P}(t) \tag{3}$$

where the state of the i^{th} cell at $(t + 1)^{th}$ instant of time is given by

$$\mathcal{P}_i(t + 1) = min\{1, \sum_{j=1}^{n} T_{ij} \cdot \mathcal{P}_j(t)\}$$

and

$$T_{ij} = \begin{cases} 1, & \text{if next state of } i^{th} \text{ cell depends on} \\ & \text{present state of } j^{th} \text{ cell} \\ & i,\ j\ =\ 1,\ 2, \cdots, n \\ 0, & \text{otherwise} \end{cases}$$

Proof: In the dependency matrix T, the i^{th} row designates a rule being applied on i^{th} cell. As we consider 3-neighborhood dependency, there are atmost 3 non-zero entries in any row of the matrix. That is, T_{ij} has non-zero values only for $j = i - 1$, $j = i$ and $j = i + 1$. So, as per the evolution noted in Table 1,

$$\mathcal{P}_i(t + 1) = min\{1, (T_{i(i-1)} \cdot \mathcal{P}_{i-1}(t) + T_{ii} \cdot \mathcal{P}_i(t) + T_{i(i+1)} \cdot \mathcal{P}_{i+1}(t))\}$$

where $\mathcal{P}_i(t)$ and $\mathcal{P}_i(t + 1)$ are the states of the i^{th} cell at t^{th} and $(t + 1)^{th}$ instant of time. So,

$$\mathcal{P}_i(t + 1) = min\{1, (T_{i1} \cdot \mathcal{P}_1(t) + \cdots + T_{i(i-1)} \cdot \mathcal{P}_{i-1}(t) + T_{ii} \cdot \mathcal{P}_i(t) + T_{i(i+1)} \cdot \mathcal{P}_{i+1}(t) + \cdots + T_{in} \cdot \mathcal{P}_n(t))\}$$

Hence,

$$\mathcal{P}_i(t + 1) = min\{1, \sum_{j=1}^{n} T_{ij} \cdot \mathcal{P}_j(t)\} \tag{4}$$

Thus, we get

$$\mathcal{P}(t + 1) = T \cdot \mathcal{P}(t)$$

where

$$T_{ij} = \begin{cases} 1, & \text{if next state of } i^{th} \text{ cell depends on} \\ & j^{th} \text{ cell } i,\ j\ =\ 1,\ 2, \cdots, n \\ 0, & \text{otherwise} \end{cases}$$

Hence, the proof. □

Example 2: A 4-cell null boundary hybrid FCA with rule vector <238,254,238,252> (that is $< (q_i + q_{i+1}), (q_{i-1} + q_i + q_{i+1}), (q_i + q_{i+1}), (q_{i-1} + q_i) >$) being applied from left to right, may be represented as follows:

$$\mathcal{P}_i(t + 1) = min\{1, \sum_{j=1}^{n} T_{ij} \cdot \mathcal{P}_j(t)\}$$

where T corresponds to the FCA with rule vector <238,254,238,252> - that is $< (q_i + q_{i+1}), (q_{i-1} + q_i + q_{i+1}), (q_i + q_{i+1}), (q_{i-1} + q_i) >$ applied from left to right. So,

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

If $\mathcal{P}(0)$=(0.80 0.20 0.20 0.00) is the initial state of the FCA with rule vector $< 238, 254, 238, 252 >$, then the four states in next 4 consecutive time instants are given by

$$\mathcal{P}(1) = (1.00\ 1.00\ 0.20\ 0.20),$$
$$\mathcal{P}(2) = (1.00\ 1.00\ 0.40\ 0.40),$$
$$\mathcal{P}(3) = (1.00\ 1.00\ 0.80\ 0.80),$$
$$\mathcal{P}(4) = (1.00\ 1.00\ 1.00\ 1.00).$$

If we apply corresponding T for the above rule vector, we get similar state transition diagram.

3.3 Complemented FCA

It is interesting to note that the complemented FCA rules can also be represented conveniently in matrix notation. The rule 17, represented by $(\overline{q_i + q_{i+1}})$, is the complement of rule 238 represented as $(q_i + q_{i+1})$. However, since the NOR function cannot be represented in the multiplicative notation, let us symbolically represent it as $\overline{T}$. Thus T represents the FCA with OR rules only and $\overline{T}$ the FCA with NOR rules. In a complemented FCA the next state of a cell is obtained first by obtaining the OR output according to Eq. (4) and then by complementing this state, - that is

$$\mathcal{P}_i(t+1) = | F_i - min\{1, \sum_{j=1}^{n} T_{ij} \cdot \mathcal{P}_j(t)\} | \qquad (5)$$

where $\mathcal{P}_i(t)$ is the present state of i^{th} cell, $\mathcal{P}_i(t+1)$ is the next state of i^{th} cell, and F is an n-dimensional binary vector (n is the number of cells), responsible for complement operation after addition. F has got non-zero value (that is, 1) for the cell positions where complementation is required. The global state at the next instant be represented by the state transition equation:

$$\mathcal{P}(t+1) = | F - T \cdot \mathcal{P}(t) |$$

Example 3: A 4-cell null boundary hybrid FCA with rule vector <238,1,238,3> (that is $< (q_i + q_{i+1}), (\overline{q_{i-1} + q_i + q_{i+1}}), (q_i + q_{i+1}), (\overline{q_{i-1} + q_i}) >$) being applied from left to right, may be represented as follows:

$$\mathcal{P}_i(t+1) = | F_i - min\{1, \sum_{j=1}^{n} T_{ij} \cdot \mathcal{P}_j(t)\} |$$

where T corresponds to the FCA with rule vector $< 238, 254, 238, 252 >$ - that is $< (q_i + q_{i+1}), (q_{i-1} + q_i + q_{i+1}), (q_i + q_{i+1}), (q_{i-1} + q_i) >$ applied from left to right. So,

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

In this example, the second and fourth cells require NOR operation. Hence

$$F = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

If $\mathcal{P}(0)$=(0.80 0.20 0.20 0.00) is the initial state, then for the FCA configured with the rule vector $< 238, 1, 238, 3 >$,

$$\mathcal{P}(1) = (1.00\ 0.00\ 0.20\ 0.80),$$
$$\mathcal{P}(2) = (1.00\ 0.00\ 1.00\ 0.00).$$

If we apply corresponding T and F for above rule vector, we get similar state transition diagram.

Next section introduces a special class of FCA termed as FMACA. The pattern classifier proposed in Sect. 4.1 is built around this FMACA.

4. Fuzzy Multiple Attractor CA (FMACA)

A FMACA is a special class of FCA that can efficiently model an associative memory to perform pattern recognition/classification task [33]. Its state transition behavior consists of multiple components - each component, as noted in Fig. 1, is an inverted tree, each rooted on a cyclic state. A cycle in a component is referred to as an attractor. In the rest of the paper we consider only the FMACA having the node with self loop as an attractor state. The states in the tree rooted on an attractor form an attractor basin.

Figure 1 illustrates the state space of a 3-cell 5-state hybrid FMACA with rule vector $< 170, 238, 0 >$ - that is, rule 170 is applied on left most cell, followed by rule 238 on next one and so on. The nodes with the patterns (0.00 0.00 0.00), (0.25 0.25 0.00), (0.50 0.50 0.00), (0.75 0.75 0.00), and (1.00 1.00 0.00) are the attractors of the five components in the FMACA of Fig. 1. The state space of this FCA is divided into five attractor basins built around attractors a, b, c, d, and e. The states in a basin other than the attractor are referred to as transient states in the sense that a FMACA finally settles down in one of its attractor cycles after passing through such transient states.

Characterization of FMACA (both complemented and non-complemented), as reported in [33], establishes the fact that FMACA provides both equal and unequal size of basins. Variations in the state transition behavior make the FMACA as a potential candidate for pattern classifier. In order to establish FMACA as a pattern classifier, we present some interesting properties of FMACA.

- **Definition 5:** The depth d of a FMACA is defined as the number of clock cycles required to reach the attractor state from any nonreachable state in the state transition diagram of the FMACA.
- **Property 1:** If d is the depth of a FMACA with dependency matrix T, then

$$T^{d+1} = T^d \qquad (6)$$

Example 4: The example FMACA of Fig. 1 is used to illustrate the above results.

- It is a 3-cell 5-state FMACA having 5-attractor basins.
- The depth (d) of the FMACA is 2.
- The rule vector of the FMACA of Fig. 1 is <170, 238, 0>. So, the dependency matrix is

$$T = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad F = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

In this case,

$$T^2 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} = T^3$$

- For the rule vector $< 238, 1, 238, 3 >$,

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad F = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

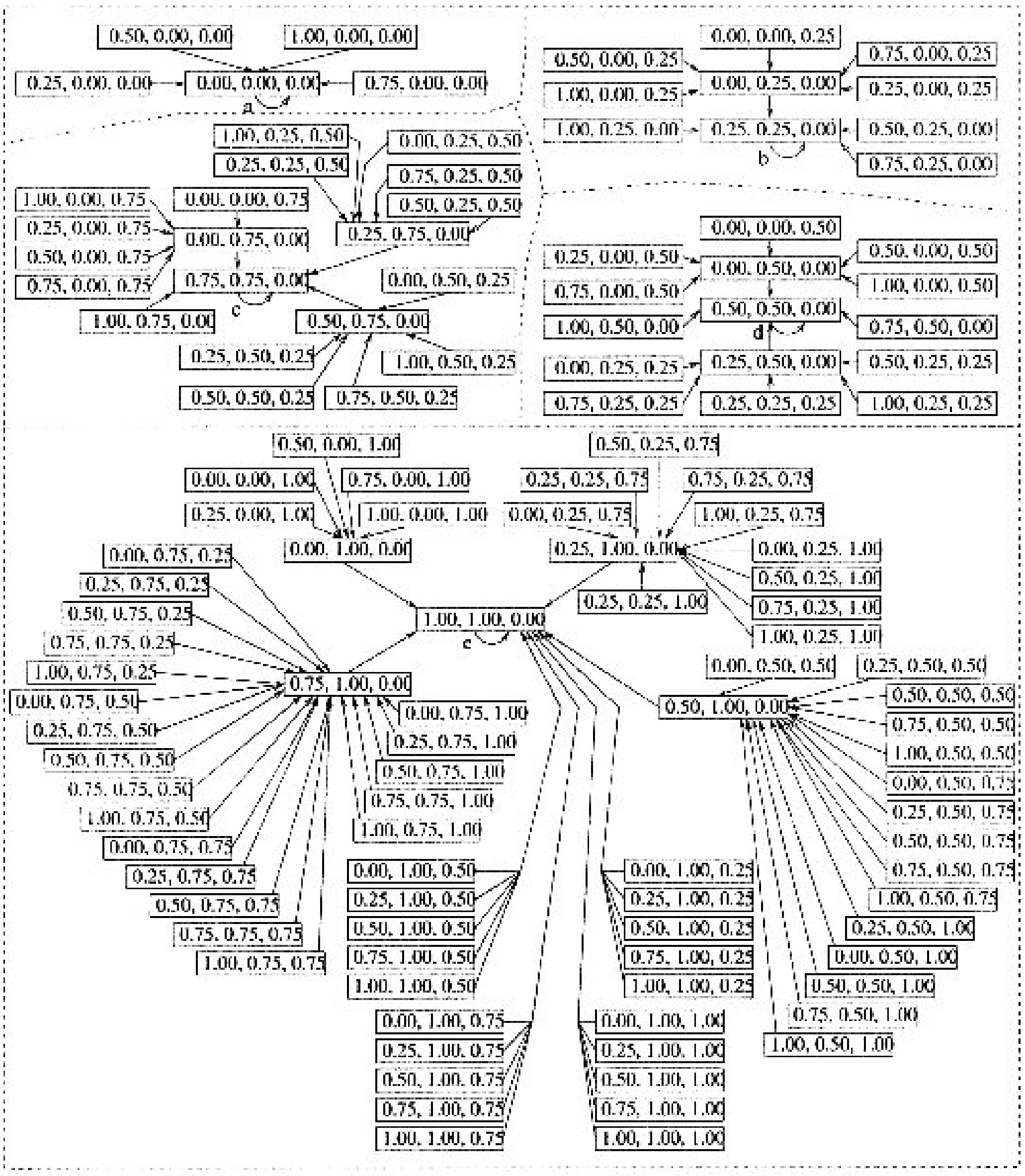


Fig. 1 State space of a 3-cell 5-state FMACA divided into five attractor basins.

In this case,

$$T^3 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} = T^4$$

4.1 FMACA Based Pattern Classifier

An n-cell FMACA with k-attractor basins can be viewed as a natural classifier. It classifies a given set of patterns into k distinct classes, each class containing the set of states in the attractor basin. The following example illustrates a FMACA based two class classifier.

Example 5: Let us have two pattern sets S_1 (Class I) and S_2 (Class II) with three attributes as noted in Table 3. In order to classify these two pattern sets into two distinct classes - Class I and II respectively, we have to design a FMACA such that the patterns of each class falls in distinct attractor basins of a FMACA.

The FMACA of Fig. 1 is able to classify them into distinct attractor basins where Class I (S_1) is represented by one set of attractor basins (say [I]= {(0.00 0.00 0.00), (0.25 0.25 0.00), (0.75 0.75 0.00) and (0.50 0.50 0.00)} in Fig. 1) while Class II (S_2) is represented by rest of the basins (say [II] = {(1.00 1.00 0.00)}). The attractor will identify the class of the patterns uniquely. The attractor yields the address of the memory that stores the class information. Therefore, Class I attractors point to the memory address {a, b, c, d} storing Class I information, while Class II information is stored in the memory address pointed by the attractor {e} (Fig. 2).

When the FMACA is loaded with an input pattern say

Table 3 Example dataset.

Attribute 1	Attribute 2	Attribute 3	Class
0.00	0.00	0.50	I
0.75	1.00	0.00	II
1.00	0.75	0.50	II
1.00	1.00	1.00	II
0.00	0.25	0.00	I
0.25	0.25	0.00	I
0.75	1.00	1.00	II
1.00	1.00	0.75	II
1.00	0.75	1.00	II
0.00	0.50	0.00	I
0.50	0.75	1.00	II
0.00	0.00	0.00	I
0.25	0.00	0.00	I
0.50	0.00	0.00	I
1.00	0.75	0.75	II
0.00	0.00	0.25	I
0.75	1.00	0.75	II
0.75	0.75	1.00	II
0.00	0.00	0.75	I
0.00	0.50	0.25	I

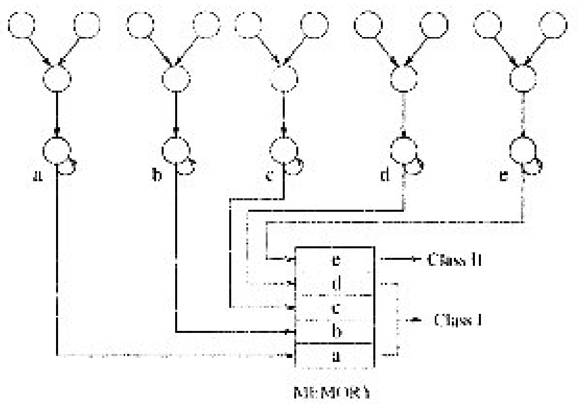


Fig. 2 FMACA based classification strategy with 5 attractor basins classifying the elements into two classes. Note: (i) An attractor basin covers the elements belonging to one class only. (ii) Each attractor points to the memory location that stores the class information.

$\mathcal{P}$ =(0.75 0.75 0.50) and is allowed to run in autonomous mode for a number of cycles equal to the depth of the FMACA, it travels through a number of states and ultimately reaches an attractor state (1.0 1.0 0.0) - the attractor representing Class II.

Based upon the above discussion, we next proceed to design a FMACA based pattern classifier classifying a given set of patterns into K classes.

4.2 Design of FMACA Based Pattern Classifier

The tree-structured pattern classifiers are used successfully in many diverse application areas. The most important feature of tree-structured classifiers is their capability to break down a complex decision-making process into a collection of simpler decisions, thus providing a solution which is often easier to interpret. This subsection presents a FMACA based tree-structured pattern classifier to classify a given set of data.

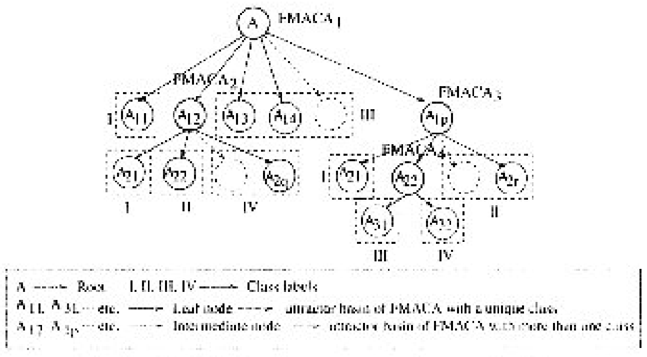

A
FMACA 1
FMACA 2
FMACA 3
FMACA 4
A11
A12
A13
A14
A1p
A21
A22
A2q
A2r
A31
A32
I
II
III
IV
A ---- Root
I, II, III, IV ---- Class labels
A11, A31 ··· etc. ---- Leaf node ---- attractor basin of FMACA with a unique class
A12, A1p ··· etc. ---- Intermediate node ---- attractor basin of FMACA with more than one class

Fig. 3 FMACA based tree-structured pattern classifier.

Like decision tree classifiers, FMACA based tree-structured classifiers recursively partition the training set to get nodes (attractor basins of a FMACA) covering patterns of only one class. Figure 3 shows a FMACA based tree-structured classifier. Each node of the tree is either a leaf or an intermediate node. A leaf node represents an attractor basin of the FMACA designated as a specific class - that is, the basin of the attractor covers the elements belonging to a single class. On the other hand, an intermediate node represents the instance to design another FMACA to classify the set of elements covered by the basin of the attractor covering elements from more than one class.

Suppose, we want to design a FMACA based classifier to classify a training set $S = \{S_1, \cdots, S_i, \cdots, S_K\}$ into K classes, where S_i is the set of elements of class i. First, a FMACA with k attractor basins is generated. The elements of the training set S get distributed into k attractor basins (nodes). Let, $\acute{S}$ be the set of elements in an attractor basin. If $\acute{S}$ belongs to only one particular class, then mark it as a leaf node and label that attractor basin as that class. Otherwise, this process is repeated recursively for each intermediate node until all the elements in each attractor basin represented by a leaf node belong to only one class. Figure 3 represents a graphical overview of the process. The above discussion is formalized in the following algorithm.

```
Algorithm 1: FMACA_Tree_Building
Input:  Training set $S = \{S_1, \cdots, S_i, \cdots, S_K\}$
Output: FMACA Tree.
        Partition ($S$, $K$);
Partition ($S$, $K$)
Step 1: Generate a FMACA with $k$ attractor basins.
Step 2: Distribute $S$ into $k$ attractor basins (nodes).
Step 3: Evaluate the distribution of patterns in
        each attractor basin (node).
Step 4: If all the patterns $\acute{S}$ covered by an attractor
        basin (node) belong to only one particular
        class, then label the attractor basin (leaf node)
        as that class.
Step 5: If $\acute{S}$ of an attractor basin belong to $\acute{K}$
        ($\acute{K} > 1$) classes, then
        Partition($\acute{S}$, $\acute{K}$).
Step 6: Stop.
```

In designing a FMACA based tree-structured classifier, it must be ensured that the tree designed is as small as possible. Because, trees with lesser number of nodes are more efficient both in terms of storage requirements and associated test time. So, the basic criteria for FMACA tree design are:

1. low error rate leading to maximum classification accuracy;
2. less number of nodes leading to low memory overhead; and
3. small tree height leading to low retrieval time.

Since, some of these requirements are conflicting in nature, we develop a heuristic solution minimizing memory overhead and retrieval time while maximizing classification accuracy. The solution mainly concentrates on following two operations at each node, other than the leaf node, for building FMACA tree.

1. Evaluation of FMACA - that is, evaluation of distribution of the elements of different classes in different attractor basins of a FMACA and the selection of the best distribution; and
2. selection of FMACA using the best distribution in the intermediate nodes.

Selection of FMACA at a node of the FMACA tree is next elaborated.

4.3 FMACA Selection to Build FMACA Tree

Splitting an intermediate node involves the design of a new FMACA to classify the subset of input elements of different classes covered by the basin of the FMACA of earlier level of the tree. An attractor basin is considered as a leaf node if all the training examples falling into the current attractor basin belong to the same class. In other words, a node (attractor basin) is split as long as there are class elements that belong to different classes. To avoid overfitting, a prepruning strategy is needed. When current node (attractor basin) is to split, its diversity is measured and compared with a threshold value. If its diversity is lesser than the threshold, then current node is split. Otherwise the learning process terminates and future class elements falling into current node (attractor basin) are classified to the most probable class of current node - that is, the class that has the maximum number of training examples in current attractor basin.

Criteria for selection of FMACA

For ease of subsequent discussions we introduce the following terminologies.

- K denotes number of classes in the dataset S.
- k denotes number of attractor basins (of a FMACA) of an intermediate node in which the dataset S is to be distributed.
- N_{ij} represents the number of elements of class j covered by i^{th} attractor basin, where $i = 1, 2, 3, 4, \cdots, k$ and $j = 1, 2, 3, 4, \cdots, K$.
- M_i indicates the distribution of class elements in the i^{th} attractor basin.

The diversity of i^{th} node (attractor basin) is given by

$$M_i = \frac{\max\{N_{ij}\}}{\sum_{j=1}^{K} N_{ij}} \tag{7}$$

If $M_i \simeq 1$, then i^{th} attractor basin indicates the class j for which N_{ij} is maximum. Otherwise, partition the examples of i^{th} attractor basins.

Suppose, after evaluating the distribution of patterns of each class, the pattern set S is partitioned into S_a and S_b, where S_a and S_b represent the pattern set belonging to leaf nodes and intermediate nodes respectively. The goodness of the splitting or partition is given by the Figure of Merit (FM), where

$$\text{FM} = \frac{|S_a|}{|S|} \tag{8}$$

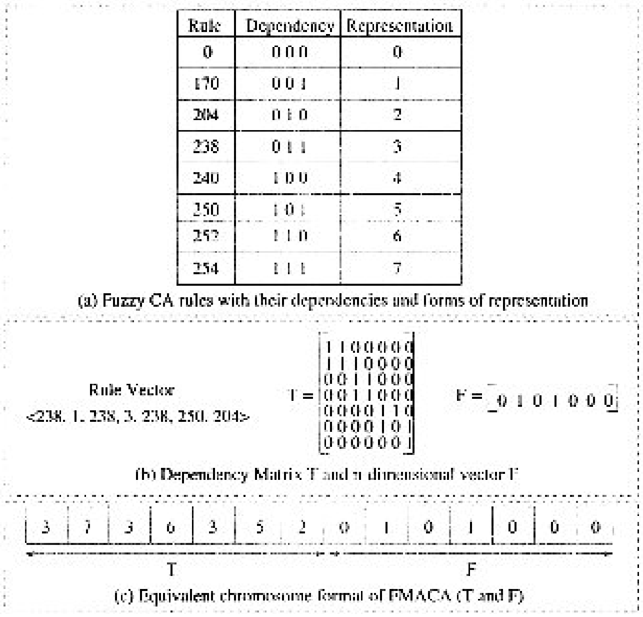
where $|S|$ represents the cardinality of the set S. The value of FM indicates the accuracy of classification or partition of a particular node. To determine the overall best distribution, classification can be done by a simple application of the FMACA to the training set. The complexity lies in determining the best distribution for each attractor basin. Two associated problems in this evaluation are next reported.

1. Suppose, the i^{th} node of the tree (an attractor basin) contains a large number of training examples and the diversity M_i of that node is less than the threshold value. In some cases, there exists a possibility where desired 3-neighborhood FMACA is not available. This is likely to occur when the training examples of different classes are highly correlated. In that case, we distribute the training examples into $\acute{k}$ attractor basins, where $\acute{k} > k$.
2. In each level of the learning process, the original training set has been partitioned into too many subsets, each belonging to an attractor basin of a FMACA. In that case, FMACA cannot generate perfect rule due to the lack of enough training examples. This may lead to lower classification accuracy. To alleviate this problem, all the training examples falling into different intermediate nodes are collected together to train a FMACA instead of training multiple individual FMACAs, each corresponding to an intermediate node. Such an approach is similar in nature to decision graph [34].

The optimal FMACA tree is constructed through the application of GA recursively at each intermediate node.

5. GA Formulation for Evolution of FMACA

The basic structure of GA revolves around the concept of



Rule	Dependency	Representation
0	0 0 0	0
170	0 0 1	1
204	0 1 0	2
238	0 1 1	3
240	1 0 0	4
250	1 0 1	5
252	1 1 0	6
254	1 1 1	7

(a) Fuzzy CA rules with their dependencies and forms of representation

Rule Vector <238, 1, 238, 3, 238, 250, 204>

$$T = \begin{bmatrix} 1&1&0&0&0&0&0 \\ 1&1&1&0&0&0&0 \\ 0&0&1&1&0&0&0 \\ 0&0&1&1&0&0&0 \\ 0&0&0&0&1&1&0 \\ 0&0&0&0&1&0&1 \\ 0&0&0&0&0&0&0 \end{bmatrix} \quad F = [\,0\ 1\ 0\ 1\ 0\ 0\ 0\,]$$

(b) Dependency Matrix T and n dimensional vector F

3	7	3	6	3	5	2	0	1	0	1	0	0	0

(c) Equivalent chromosome format of FMACA (T and F)

Fig. 4 An example chromosome for GA formulation.

encoding a solution in bit string format referred to as chromosome and evolving successive solutions according to its fitness. Three major functions of GA, random generation of initial population (IP), crossover and mutation, as developed in the current GA formulation, are next discussed.

5.1 Chromosome

Rather than the conventional bit string format, the proposed scheme employs a chromosome consisting of:

- a symbol string of numerical digits in between 0 to 7 representing $n \times n$ dependency matrix T; and
- an n-dimensional binary string representing F-vector.

So, the length of the chromosome is $2n$ where n is the number of cells in a FCA. Figure 4 represents a 14-bit chromosome corresponding to the rule vector <238, 1, 238, 3, 238, 250, 204>. While Fig. 4 (a) represents the 8 OR rules along with their dependencies and forms of representation; Fig. 4 (b) represents dependency matrix T and n-dimensional vector F corresponding to the rule vector $< 238, 1, 238, 3, 238, 250, 204 >$. Figure 4 (c) represents the chromosome format of FMACA (T and F).

5.2 Random Generation of Initial Population

To form the initial population, it must be ensured that each solution randomly generated is an n-cell FMACA with k attractor basins. The chromosomes are randomly synthesized according to the following steps.

- Randomly partition n into m integers such that $n_1 + n_2 + \cdots + n_m = n$ and $k_1 \times k_2 \times \cdots \times k_m = k$.
- For each n_i, randomly generate a valid FMACA (T_i and F_i) with k_i attractor basins.

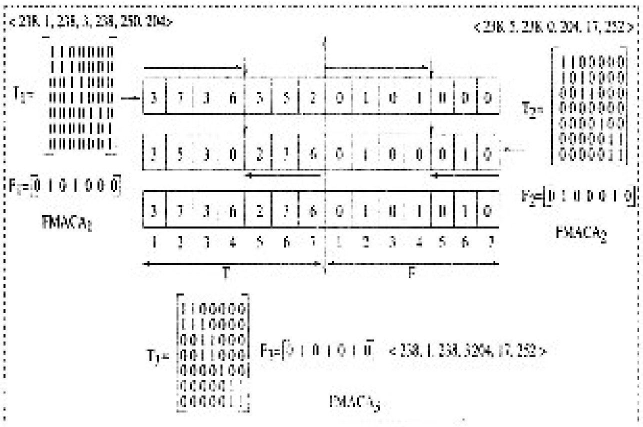


Fig. 5 An illustration of crossover technique.

- Synthesize n-cell FMACA (T and F) with k attractor basins from
    1. m dependency matrices T_is ($i = 1, 2, \cdots, m$) arranged in Block Diagonal Form [35], where each T_i represents an n_i-cell FMACA with k_i attractor basins; and
    2. concatenation of m F_is;

  where $n_1 + n_2 + \cdots + n_m = n$, and $k_1 \times \cdots \times k_m = k$.
- Synthesize an $2n$-bit chromosome corresponding to the dependency matrix T and an n-dimensional vector F through associated representation.

Figure 4 (c) represents a randomly generated 14-bit chromosome corresponding to the dependency matrix T and the complemented vector F. The 7×7 dependency matrix T is obtained from two matrices (T_1 and T_2) of length 4 and 3 respectively by Block Diagonal Form, where

$$T_1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

The 7-bit complemented vector F is produced through the concatenation of two complemented vectors (F_1 and F_2) of length 4 and 3 respectively, where

$$F_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \quad F_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

5.3 Crossover Algorithm

The crossover algorithm takes two chromosomes from the present population (PP) and forms the resultant chromosome. Like a single point crossover, it sets a crossover point and each half about the crossover point is selected from the two respective chromosomes.

Figure 5 illustrates the crossover process employed in the GA evolution with two chromosomes (FMACA_1 and FMACA_2) with T_1, F_1 and T_2, F_2 respectively. The single

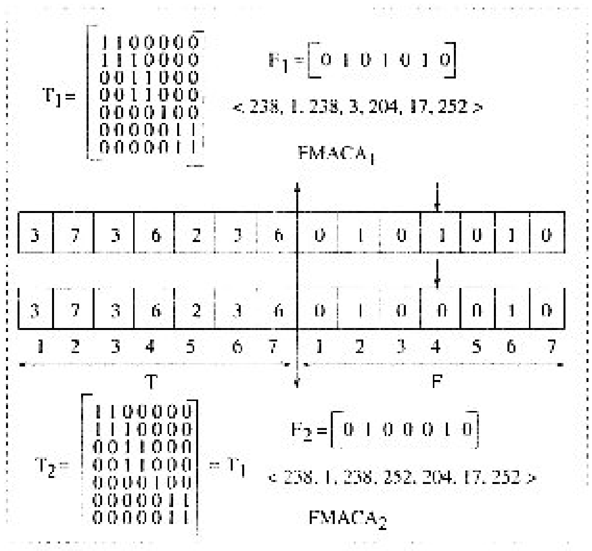


Fig. 6 An illustration of mutation technique.

crossover point is selected randomly which is 4 in this case. The first 4 symbols of first part are taken from FMACA$_1$, while the rest 3 symbols are taken from FMACA$_2$ to form dependency matrix T_3. Similarly, in the second part, first 4 symbols of FMACA$_1$ and rest 3 symbols of FMACA$_2$ are merged together to form the n-dimensional vector F_3. The resultant chromosome with T_3 and F_3 after crossover is shown in Fig. 5.

5.4 Mutation Algorithm

The mutation algorithm makes some minimal change in the existing chromosome of PP (present population) to form a new chromosome for NP (next population). Similar to conventional single point mutation, the chromosome is mutated at a single point.

Figure 6 represents an example of mutation technique on a randomly selected chromosome (FMACA$_1$). A single mutation point (4th position) is randomly selected from the second part of the chromosome. The FMACA$_2$ is the mutated version of FMACA$_1$ where T_2 is equal to T_1.

The complete algorithm for evolving FMACA through GA next follows.

Algorithm 2: Evolving FMACA

```
Input: Training set $S = \{S_1, \cdots, S_i, \cdots, S_K\}$
          Maximum Generation ($G_{max}$).
Output: Dependency matrix ($T$), $n$-dimensional vector
          $F$, and class information.
begin
    Step 1: Generate 50 new chromosomes for initial
              population (IP).
    Step 2: Initialize generation counter $GC$=zero;
              PP← IP.
    Step 3: Compute fitness value $FM$ for each
              chromosome of PP according to Eq. (8).
    Step 4: Rank chromosomes in order of fitness.
    Step 5: Store $T$, $F$, and corresponding class
              information for which fitness value $FM = 1$.
    Step 6: If $FM = 1$ for at least one chromosome
              of PP, then go to Step 12.
    Step 7: Increment generation counter ($GC$).
    Step 8: If $GC > G_{max}$ then go to Step 11.
    Step 9: Form NP by selection, crossover and
              mutation.
    Step 10: PP← NP; go to Step 3.
    Step 11: Store $T$, $F$, and corresponding class
               information for which the fitness value
               is maximum.
    Step 12: Stop.
```

The experimental results reported in Sect. 6 confirm that the GA evolution provides the desired direction to arrive at the best FMACA for classifying a given set of patterns.

6. Experimental Results

We report the experimental results to analyze the performance of the FMACA as a pattern classifier. The major metric for evaluating classifier performance is classification accuracy. We also report the generation and retrieval time, and the memory overhead required to store FMACA tree as secondary metrics.

6.1 Experimental Setup

To analyze the performance of FMACA based classifier, the experimentation has been done on randomly generated datasets. We first randomly generate K centroids which are separated by a fixed euclidean distance (say D_{min}). Then around each centroid, we randomly generate a set of patterns within d_{max} distance (maximum euclidean distance between a centroid and an element in the associated set); where $d_{max} \leq \frac{D_{min}}{2}$.

For different values of n (dimension/number of attributes) and K (number of classes), t patterns are taken for each dataset. Out of this, 50% patterns are taken from each dataset to build up the classification model. The rest 50% patterns are used to test the classification accuracy of the model. For each value of n and K, results are based on 10 different pattern sets.

All the attributes in a dataset are normalized to facilitate FMACA learning. Suppose, the possible value range of an attribute attr$_i$ is (attrval$_{i,min}$, attrval$_{i,max}$), and the real value that class element j takes at attr$_i$ is attrval$_{ij}$, then the normalized value of attrval$_{ij}$ is

$$\text{Normalize}(\text{attrval}_{ij}) = \frac{\text{attrval}_{ij} - \text{attrval}_{i,min}}{\text{attrval}_{i,max} - \text{attrval}_{i,min}}$$

All the experiments are performed in SUN with Solaris 5.6, 350 MHz clock.

6.2 GA Evolution

The time required to find out desired FMACA at each in-

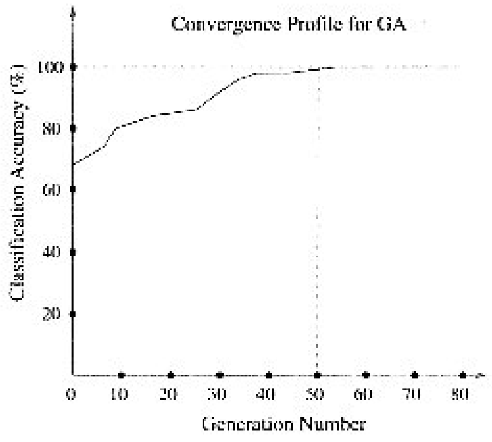


Fig. 7 Graph showing classification accuracy with generation number (G) for $n = 10$, $K = 6$ and $t = 5000$.

Table 4 Generalization of FMACA based classifier.

Dataset	Depth	Classification Accuracy		Breadth
n, K, t	of Tree	Training	Testing	of Tree
$n = 5$	1	16.9	15.6	1
$K = 2$	2	81.8	80.9	21
$t = 4000$	3	97.7	91.6	35
	4	98.4	92.4	9
$n = 10$	1	5.3	5.25	1
$K = 5$	2	24.3	23.2	13
$t = 7000$	3	65.7	63.4	53
	4	85.5	82.4	59
	5	96.6	93.9	31

Table 5 Classification accuracy of FMACA and C4.5.

No of	Size of	No of	Classification Accuracy	
Attributes	Dataset	Classes	FMACA	C4.5
5	5000	2	99.60	98.61
		4	97.80	94.20
		6	94.80	86.20
		8	95.30	88.01
		10	93.50	82.50
6	6000	2	97.90	94.30
		4	97.10	93.10
		6	94.70	85.90
		8	93.00	81.60
		10	93.30	83.40
7	10000	2	100.00	99.90
		4	95.90	88.10
		6	94.20	84.50
		8	92.89	81.00
		10	91.20	72.70
8	10000	2	97.30	91.21
		4	94.90	83.61
		6	92.80	79.60
		8	90.10	72.41
		10	90.70	73.93
9	10000	2	97.10	89.50
		4	92.70	76.87
		6	91.60	72.19
		8	91.80	74.00
		10	89.70	66.98
10	10000	2	96.60	88.00
		4	93.30	77.10
		6	91.20	69.30
		8	90.40	66.90
		10	89.10	62.70

termediate node increases with the value of G_{max} - the maximum number of generations of the GA. Our objective is to identify the optimum value of G_{max} while achieving high classification accuracy.

To identify the minimum value of G_{max}, we carry out extensive experiments to evolve desired FMACA for a particular value of n (dimension/number of attributes), K (number of classes), and t (size of dataset). For a specific value of n, K, and t, 10 different sets of data to be trained are generated randomly.

Figure 7 depicts the percentage of classification accuracy for $n = 10$, $K = 6$, and $t = 5000$ at different values of generation number (G). As the generation number (G) of the GA increases upto 50, the classification accuracy also increases. But, it improves at a very slow gradient on further increase of G. So, the GA is allowed to evolve for maximum 50 ($= G_{max}$) generations.

6.3 Generalization of FMACA Tree as a Classifier

The aim of the proposed GA evolution is to generate a FMACA tree that is good at classifying patterns similar to, but not identical to, patterns in the training set. That is, a FMACA tree that has the ability to act as a general pattern classifier.

Table 4 represents the generalization capability of the FMACA tree. While Column II depicts the depth of the tree which is the number of layers from the root to the leaf nodes, Column III represents the classification accuracy of both training and testing datasets respectively. Column IV represents the breadth of the tree which is the number of intermediate nodes in each level of the tree. The classification accuracy of training and testing confirm that the evolved FMACA tree can generalize a dataset irrespective of the number of attributes (n), classes (K) and tuples (t).

6.4 Performance Analysis

In this subsection, we compare the classification accuracy, generation and retrieval time, and memory overhead of the proposed classifier with these arrived with C4.5 [19]. The source code of C4.5 is obtained from http://www.cse.unsw.edu.au/~quinlan/.

6.4.1 Classification Accuracy

Classification accuracy is defined as the percentage of test samples that are correctly classified. Table 5 compares the classification accuracy of FMACA based tree-structured classifier with that of C4.5 [19].

Columns I and II of Table 5 represent the number of attributes/dimension (n) and size (t) of the dataset, while Column III depicts the number of classes (K) in the dataset. In Column IV, we provide the classification accuracy of FMACA based classifier and C4.5 for both training and testing datasets respectively.

Table 6 Classification efficiency and memory overhead of FMACA and C4.5.

No of Attri. (n)	No of Tuples (t)	Efficiency (ms)				Memory Overhead (byte)	
		Generation Time		Retrieval Time			
		FMACA	C4.5	FMACA	C4.5	FMACA	C4.5
5	2×10^2	291	43	71	74	1071	4335
	2×10^3	14215	273	3	306	1171	39519
	2×10^4	52557	756	80	812	1078	336815
	2×10^5	18151	2782	286	2054	1261	3200687
6	2×10^2	29276	22	3	51	1274	4839
	2×10^3	722725	162	4	259	1104	46319
	2×10^4	252458	791	35	874	1239	397727
	2×10^5	1391428	36033	363	35998	1452	3979847

All the results reported in Table 5 confirm higher classification accuracy of FMACA than that of C4.5.

6.4.2 Generation and Retrieval Time

The performance of a classifier depends on how efficient is the classifier generation process. The algorithm for generating the classification functions should be efficient. But, the generation efficiency has not been an important design consideration because usually the classifier is generated once and then is used over and over again. Thus the performance of a classifier mainly depends on how efficient is the classifier in retrieving all instances of a specified class.

Table 6 represents the generation and retrieval time of the proposed FMACA based tree classifier. Columns I and II of Table 6 represent the number of attributes (n) and number of tuples of the dataset (t) respectively. Column III depicts the generation and retrieval time. The results of C4.5 on the same dataset are provided for the sake of comparison. All the results reported in Table 6 establish that though the generation time of FMACA is higher than that of C4.5, the retrieval time is much lesser than C4.5.

The result on the first row of the table involves comparatively lesser number of training elements in a class - 100 for training and 100 for testing (total of 2×10^2 elements). Quality of the design of FMACA tree, as explained in the last section, suffers due to insufficient/inadequate number of data elements in the training set. Such a situation is likely to lead to larger height of the FMACA tree and so higher retrieval time.

6.4.3 Memory Overhead

The memory overhead of the proposed tree-structured classifier is the memory required to store the n-cell FMACA ($n \times n$ dependency matrix T and n-dimensional vector F) at each intermediate node and the corresponding attractors information. As we restrict to 3-neighborhood dependency, the dependency matrix T becomes a tri-diagonal matrix. So, the number of bits required to store each n-cell FMACA is equal to $[(3n - 2) + n]$. Thus, the memory overhead at each intermediate node of the tree, MO_{int}, is given by

$$\begin{aligned} MO_{int} = &\ \text{sizeof(FMACA)} + \\ &\ \text{sizeof(associated attractors information)} \end{aligned}$$

Hence, total memory overhead to implement FMACA based tree-structured pattern classifier, MO_{total}, is

$$\begin{aligned} MO_{total} = &\ MO_{int} \times \\ &\ \text{total\_number\_of\_intermediate\_nodes} \end{aligned}$$

Table 6 represents the memory overhead required to implement the FMACA tree. Column IV of Table 6 represents the comparison of memory overhead between FMACA and C4.5. All the results reported in Column IV of Table 6 confirm the low memory overhead of the proposed pattern classifier.

7. Conclusion

This paper deals with FCA configured with rules supporting OR and NOR function as next state logic. The characterization of FCA based on the dependency matrix and the fuzzy states of its cells have been reported. The analysis of FCA based on the matrix algebraic formulation provides the necessary foundation for analysis of 3-neighborhood FMACA as a classifier. Theoretical formulation coupled with extensive experimental results have established the scope of the FMACA in pattern recognition/classification problem.

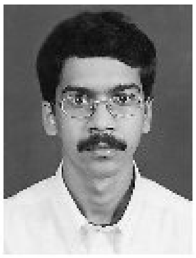
Acknowledgment

The authors thank Paola Flocchini of University of Ottawa, and the anonymous reviewers for their many helpful remarks and suggestions.

References

[1] T. Toffoli and N. Margolus, Cellular Automata Machines, The MIT Press, Cambridge, MA, 1987.
[2] C.G. Langton, "Self-reproduction in cellular automata," Physica D, vol.10, pp.135–144, 1984.
[3] T. Toffoli, "Reversible computing," in Automata, Languages and Programming, ed. J.W. De Bakker and J. Van Leeuwen, pp.632–644, 1980.
[4] G. Vichniac, "Simulating physics with cellular automata," Physica D, vol.10, pp.96–115, 1984.
[5] J.V. Neumann, The Theory of Self-Reproducing Automata, ed. A.W. Burks, University of Illinois Press, Urbana and London, 1966.
[6] S. Wolfram, A New Kind of Science, Wolfram Media, 2002.
[7] M. Stonebraker, Readings in Database Systems, 2nd ed., Morgan Kaufmann, 1993.

[8] D. Fisher, "Improving inference through conceptual clustering," Proc. 1987 AAAI Conference, Seattle, WA, 1987.
[9] R.S. Michalski, J.M. Carbonnel, and T.M. Mitchell, Machine Learning: An Artificial Intelligence Approach, Morgan Kaufmann, Los Atlos, CA, 1983.
[10] T.M. Mitchell, "Generalization as search," Artif. Intell., vol.18, pp.203–226, 1982.
[11] S. Fotheringham and E.P. Rogerson, Spatial Analysis and GIS, Taylor and Francis, 1994.
[12] L. Kaufmann and P.J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, John Wiley and Sons, 1990.
[13] G. Shaw and D. Wheeler, Statistical Techniques in Geographical Analysis, David Fulton, London, 1994.
[14] M. Mehta, R. Agrawal, and J. Rissanen, "SLIQ: A fast scalable classifier for data mining," Proc. International Conference on Extending Database Technology, Avignon, France, 1996.
[15] J. Shafer, R. Agrawal, and M. Mehta, "SPRINT: A scalable parallel classifier for data mining," 22nd VLDB Conference, 1996.
[16] P. Cheeseman and J. Stutz, "Bayesian classification (AutoClass): Theory and results," in Advances in Knowledge Discovery and Data Mining, ed. U.M. Fayyad, G. Piatetsky-Shapiro, P. Smith, and R. Uthurusamy, pp.153–180, AAAI/MIT Press, 1996.
[17] J. Hertz, A. Krogh, and R.G. Palmer, "Introduction to the theory of neural computation," Santa Fe Institute Studies in the Sciences of Complexity, Addison Wesley, 1991.
[18] R. Lippmann, "An introduction to computing with neural nets," IEEE ASSP Mag., vol.4, no.22, April 1987.
[19] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Francisco, CA, 1993.
[20] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, Classification and Regression Trees, Wadsworth, Belmont, 1984.
[21] D.E. Goldberg, Genetic Algorithms in Search, Optimizations and Machine Learning, Morgan Kaufmann, 1989.
[22] N. Ganguly, P. Maji, S. Dhar, B.K. Sikdar, and P.P. Chaudhuri, "Evolving cellular automata as pattern classifier," Proc. Fifth International Conference on Cellular Automata for Research and Industry, ACRI 2002, pp.56–68, Switzerland, Oct. 2002.
[23] S. Chattopadhyay, S. Adhikari, S. Sengupta, and M. Pal, "Highly regular, modular, and cascadable design of cellular automata-based pattern classifier," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.8, no.6, pp.724–735, Dec. 2000.
[24] P. Maji, C. Shaw, N. Ganguly, B.K. Sikdar, and P.P. Chaudhuri, "Theory and application of cellular automata for pattern classification," Fundam. Inform., vol.58, no.3–4, pp.321–354, Dec. 2003.
[25] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and unsupervised discretization of continuous features," Proc. 12th International Conference on Machine Learning, pp.194–202, Morgan Kaufmann, San Francisco, CA, 1995.
[26] R. Kohavi and M. Sahami, "Error-based and entropy-based discretization of continuous features," Proc. 2nd International Conference on Knowledge Discovery and Data Mining, pp.114–119, Portland, OR, AAAI Press, 1996.
[27] W.G. Wee and K.S. Fu, "A formulation of fuzzy automata and its application as model of learning systems," IEEE Trans. Syst., Man Cybern., vol.5, pp.215–223, 1969.
[28] G. Cattaneo, P. Flocchini, G. Mauri, and N. Santoro, "Cellular automata in fuzzy backgrounds," Physica D, vol.105, pp.105–120, 1997.
[29] P. Flocchini, F. Geurts, A. Mingarelli, and N. Santoro, "Convergence and aperiodicity in fuzzy cellular automata: Revisiting rule 90," Physica D, 2000.
[30] A. Adamatzky, Identification of Cellular Automata, 1994.
[31] A. Adamatzky, "Hierarchy of fuzzy cellular automata," Fuzzy Sets and Systems, vol.62, pp.167–174, 1994.
[32] S. Wolfram, Theory and Application of Cellular Automata, World Scientific, 1986.
[33] P. Maji and P.P. Chaudhuri, "FMACA: A fuzzy cellular automata based pattern classifier," Proc. 9th International Conference on Database Systems for Advanced Applications, pp.494–505, Korea, March 2004.
[34] J.J. Oliver, "Decision graphs — An extension of decision trees," Proc. 4th International Conference on Artificial Intelligence and Statistics, pp.343–350, Miami, FL, 1993.
[35] S. Lipschutz, Linear Algebra, 2nd ed., Schaum's Outline Series, McGraw-Hill, 1991.



Pradipta Maji received the B. Sc (Hons) degree in Physics and M. Sc degree in Electronics Science from Jadavpur University, India, in 1998 and 2000, respectively. He has submitted his Ph.D thesis entitled "Cellular Automata Evolution For Pattern Recognition" in Jadavpur University, India, 2004. Currently, he is a Lecturer of Department of Computer Science and Engineering & Information Technology, Netaji Subhash Engineering College, West Bengal, India. He has published around 25 research papers in international journals and conference proceedings. His research interests include Cellular Automata, Neural Network, Genetic Algorithm, Pattern Recognition, Data Mining, Soft Computing, Artificial Intelligence, Bioinformatics.

P. Pal Chaudhuri graduated in 1963. He was associated with IBM World Trade Corporation in various capacities until 1975. Subsequently, he switched over to academia and started his career at Indian Institute of Technology (IIT), Kharagpur. He took the leading role to initiate the Computer Science and Engineering program at IIT Kharagpur in the late 1970s. During the period of 1986 to 1988, he was the Head/Chairman of the IIT Computer Science and Engineering Department. In 1991 and 1992, he was a Visiting Professor at the University of Illinois, Urbana-Champaign, for one semester and in the subsequent semester he worked for Cadence Design Systems (India) as a Technical Advisor. During the early 1980s, he initiated a full scale research thrust and established an excellent research base at IIT Kharagpur in the field of VLSI design and testing. Since late 1980s, he has pioneered the study of the homogeneous structure of Cellular Automata (CA) and developed matrix algebraic tools in GF (2) to completely characterize the autonomous structure of CA machines. He subsequently applied this theory to develop a wide variety of applications in diverse fields. He has published more than 100 research papers in international journals and conference proceedings. He authored the books "Additive Cellular Automata Theory and Applications Volume 1" (IEEE, 1997) and "Computer Organization and Design" (Prentice Hall), a text book for undergraduate and graduate level courses. In November 1996, he was invited by Intel Corporation, USA, as a Visiting Faculty. He worked at Intel Research Labs, Portland, OR, until the end of 1997. Upon his return, he joined his Alma Mater, Bengal Engineering College. His research group there has been developing the theory of GF (2^p) CA and its applications in a wide variety of fields such as data compression, data security, pattern recognition, simulation of physical systems, VLSI design, and test etc.