BioSystems 99 (2010) 6–16

Contents lists available at ScienceDirect

BioSystems

journal homepage: www.elsevier.com/locate/biosystems

Evaluation of dynamic behavior forecasting parameters in the process of transition rule induction of unidimensional cellular automata

Wagner Rodrigo Weinert[a], Heitor Silvério Lopes[b,*]

[a] Federal Institute of Education Science and Technology of Paraná (IFPR), R. Antônio Carlos Rodrigues 453, 83215-750 Paranaguá (PR), Brazil
[b] Bioinformatics Laboratory/CPGEI, Federal University of Technology – Paraná (UTFPR), Av. 7 de setembro 3165, 80230-901 Curitiba (PR), Brazil

ARTICLE INFO

Article history:
Received 17 January 2009
Received in revised form 3 August 2009
Accepted 6 August 2009

Keywords:
Cellular automata
Dynamic behavior forecasting parameters
Dynamic systems
Evolutionary computation

ABSTRACT

The simulation of the dynamics of a cellular systems based on cellular automata (CA) can be computationally expensive. This is particularly true when such simulation is part of a procedure of rule induction to find suitable transition rules for the CA. Several efforts have been described in the literature to make this problem more treatable. This work presents a study about the efficiency of dynamic behavior forecasting parameters (DBFPs) used for the induction of transition rules of CA for a specific problem: the classification by the majority rule. A total of 8 DBFPs were analyzed for the 31 best-performing rules found in the literature. Some of these DBFPs were highly correlated each other, meaning they yield the same information. Also, most rules presented values of the DBFPs very close each other. An evolutionary algorithm, based on gene expression programming, was developed for finding transition rules according a given preestablished behavior. The simulation of the dynamic behavior of the CA is not used to evaluate candidate transition rules. Instead, the average values for the DBFPs were used as reference. Experiments were done using the DBFPs separately and together. In both cases, the best induced transition rules were not acceptable solutions for the desired behavior of the CA. We conclude that, although the DBFPs represent interesting aspects of the dynamic behavior of CAs, the transition rule induction process still requires the simulation of the dynamics and cannot rely only on the DBFPs.

1. Introduction

Cellular automata (CAs) are discrete dynamic systems formed by simple and identical components. The dynamics of a CA is determined by its state transition rule. The application of such rule in the current configuration of the CA leads to another configuration and, thus successively changing along time. The overall dynamic behavior presents certain characteristics that can classify it. In the past, Wolfram (1984) proposed a standard classification for the dynamic behavior of CAs, qualitatively divided into four classes. Later, Li and Parckard (1990) improved Wolfram's classification to six classes.

The computational simulation of a CA system is relatively simple, provided the transition rule is known. However, finding a transition rule for a given dynamic behavior is a very difficult task, for which there is no efficient method to date. This work is focused in this gap, presenting and evaluating an approach for the induction of transition rules for CAs.

The classical problem known as classification by the majority rule (Juillé et al., 1998) (see Section 2.3) is frequently studied in the CAs theory. Basically, this problem consists in finding a transition rule that, when applied to all the initial cells of a d-dimensional lattice, will lead them either to state 0 or state 1, after n time-steps. This feature assures that all possible solutions (rules) to this problem will exhibit a behavior defined as "null", according to Li and Parckard (1990) classification. The expected final state of the lattice depends on the density of the CA in its original configuration.

It is important to point out that every transition rule that is said a solution for the problem presents a "null" behavior. However, the opposite does not hold, since a many rules that present a "null" behavior are not solution for the classification by the majority problem.

Several authors have proposed the use of computational intelligence techniques (more specifically, evolutionary computation), for determining appropriate transition rules for the classification by the majority rule problem (Richards et al., 1990; David et al., 1996; Juillé et al., 1998; Morales et al., 2001; Ferreira, 2002; Oliveira et al., 2002b). The general approach used by those works is the evolution of a population of individuals (in this case, rules) for y generations. At each generation a set of genetic operators actuate in the population generating hopefully better individuals. The quality of the individuals is systematically evaluated by a fitness function. The evolution process is ended when a satisfactory solution is found or a maximum number of generations is reached. In general, the

* Corresponding author.
E-mail addresses: wrweinert@gmail.com (W.R. Weinert), hslopes@utfpr.edu.br, hslopes@pesquisador.cnpq.br (H.S. Lopes).

0303-2647/$ – see front matter © 2009 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.biosystems.2009.08.002

problem of rule induction for CAs uses as fitness function a measure of the performance obtained by the analysis of the dynamics of the system. Every possible candidate solution (transition rule) is submitted to a simulation procedure. The rule is applied to a random initial configuration of the CA and them iterated n times, thus achieving its final configuration. Both initial and final states of the CA are compared. If both are the same, thus the rule is a solution for the problem. Otherwise, a similarity measure quantifies the quality of the rule. Using this approach, obtaining the fitness of a rule has a high computational cost, since a very large number of initial random configuration have to be simulated. For some applications, this fact makes unfeasible the evolutionary approach previously described, since it is necessary to evaluate a large number of candidate solutions. More recently, a dedicated reconfigurable hardware was proposed to compute the fitness of transition rules for the simulation of CAs (Weinert et al., 2007). Although this seems to be a viable alternative, regarding computational cost, this methodology is restricted to a single problem (classification by the majority rule), and it is quite difficult to adapt to other rule-induction problems in CAs.

Several measures that can be used to forecast the behavior of a CA can be found in the recent literature. These include for instance: sensitivity (Binder, 1993), neighborhood domain, activity propagation and absolute activity (Oliveira et al., 2001), Wuensche's Z parameter (Wuensche, 1999), activity (Langton, 1990), mean field parameters (Li, 1991) and Li's Z parameter (Li, 1991). These parameters, known as dynamic behavior forecasting parameters (DBFPs), are computed for a given transition rule without the necessity of simulating the dynamics of the CA for many iterations. The values obtained by the DBFPs are them used to classify the dynamic behavior of the CA. However, Oliveira et al. (2002a) demonstrated that parameters sensitivity, absolute activity, neighborhood domain and activity propagation can be useful for a genetic algorithm to find rules for the majority classification rule problem. The evaluation of rules is obtained as a function of two parameters with different weights. The first parameter considers the DBFPs values and causes the search to be filtered towards rules with "null" behavior. The second parameter is computed by simulating the dynamics of the CA, and verifies if the rule satisfactory solves the problem.

The objective of this work is to develop an evolutionary computation method for the induction of transition rules of CAs capable of presenting a "null" behavior, for the classification by the majority rule problem. The central idea is to find a set of DBFPs and combine them in such a way to be useful for the evolutionary process of rule induction.

This work is organized as follows. The next subsection presents some selected references about applications of CAs, with special emphasis on the simulation of biological systems. Section 2 presents a short introduction about the concepts pertaining CAs, including the features of the dynamic behavior, the DBFPs and the classification by the majority problem. A correlation analysis of the DBFPs is presented in Section 3, aiming at gathering useful information for the rule induction procedure using a gene expression programming algorithm described in Section 4. Results are shown and commented in Section 5. Finally, Section 6 concludes the work and points out future directions of research.

1.1. Related Work

Many different problems can be modeled by CAs and, thus, can be applied to many areas such as cryptography (Wolfram, 1986; Tomassini and Perrenoud, 2001; Benkiniouar and Benmohamed, 2004),text compression (Khan et al., 1999), scheduling (Swiecicka and Seredynski, 2000), task synchronization (Das et al., 1995), image processing (Rosin, 2005), data classification (David et al., 1996), stock market behavior (Wei et al., 2003) and earthquake activity (Georgoudas et al., 2007).

There has been a growing interest in using CAs for modeling and simulation of biological systems. For instance, Kansal et al. (2000) presented a CA-based model capable of simulating the growth of brain tumors. Authors used this model to simulate the Gompertzian tumor growth. The predicted composition and growth rates were in agreement with a test case from the medical literature. The dynamical simulation allowed the identification of clonal competition in the system, an important property of tumor growth.

Genetic algorithms and CAs were used by Mizas et al. (2008) for reconstructing the evolution of DNA sequences. This work is particularly interesting due to its complexity and its contribution to the biological sciences. They modeled a DNA chain using an unidimensional CA, which alphabet represents the four bases of DNA (A, C, G and T). The genetic algorithm was used to find a transition rule for the CA, capable of leading the chain to the desired dynamic behavior. By starting with two DNA sequences previously identified in different evolutionary moments, the rule found by the algorithm was capable of explaining the evolutionary changes of the sequences. This system can be particularly interesting for the identification of mutations that take place in the DNA throughout evolutionary periods.

Kiera et al. (1996) developed a CA model of an enzyme reaction with water substrate. The ease of simulation and the illustrative value of the model showed that CA models can be useful in the study of dynamic phenomena such as enzyme kinetics. This study opened up opportunities for the study of a general enzyme model and the effects on its behavior, including the interaction of ingredients with the solvent.

Spatial characteristics such as localized populations of dead cells might adversely affect the spread of an infection. Beauchemina et al. (2005) used a simple bidimensional CA model of a viral infection to investigate, through dynamic simulation, the influence of spatial heterogeneities on viral spread. The CA model was validated against clinical immunological data for uncomplicated influenza A infections.

Malleta and Pillisb (2006) presented a hybrid cellular automata-partial differential equation model to describe, through dynamic simulation, the interactions between a growing tumor next to a nutrient source and the immune system of the host organism.

Gangadhar (2005) presented an innovative method for protein sequence alignment using a bidimensional CA. Starting from an initial unaligned state, the CA evolves for a number of iterations according to a dynamic behavior defined by a set of transition rules. The emerged behavior leaded to the alignment of the sequences.

A novel way to visualize biological sequences using CAs was devised by Xiao et al. (2005). In this work, a sequence of amino acids is transformed into a digital encoding and a transition rule systematically applied over it creates a spatio-temporal diagram. Using such dynamic approach, it is possible to observe characteristics that were originally hidden in the amino acids chain.

Laurio et al. (2007) showed how to create CAs for recognizing regular patterns in protein sequences in such a way to identify families and functional sites. Starting from a regular pattern from Prosite (Hofman et al., 1999), a rule is determined using regular grammars which, in turn, is converted to a transition rule characterizing the dynamic behavior of a CA.

There are other works in the recent literature that employed CAs to model the dynamics of biological systems, for instance: epidemic propagation (Fu and Milne, 2003), simulation of cell infection caused by the HIV virus (Corne and Frisco, 2008) and pray–predator modeling (Chen and Mynett, 2003).

In the previous mentioned works, the models generated by CAs allowed a better comprehension of the dynamical behavior of specific biological systems. Usually, the computational cost of the

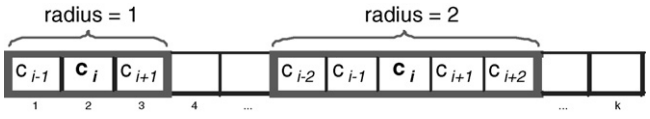

Fig. 1. Definition of an unidimensional CA.

simulation of the dynamic behavior of CA is high (see Section 2.1). Therefore, the study of methods that give some insight about the behavior of CAs, the focus of this work, may have large practical applicability in complex biological systems.

2. Cellular Automata

A CA can be represented by a d-dimensional matrix structure. All the cells of this structure are identical and they keep a neighborhood relationship determined by a predefined radius (r). The size (m) of this neighborhood is defined as a function of r, according to Eq. (1):

$$m = 2r + 1 \tag{1}$$

Boundary conditions are set so as to allow cells situated at the extremities of the matrix to be connected each other. All cells of the d-dimensional matrix are updated at the same time, in parallel, by a state transition rule. The transition rule is applied to each cell and determines the next state of the cell, considering its current state and the state of the neighboring cells. Successive iterations of the transition rule with the CA will lead to an observable dynamic behavior. Mitchell et al. (1996) defined the following formal notation for CAs:

- Σ: set of possible states of a cell;
- k: number of elements of the set Σ;
- i: index of a specific cell;
- S_i^t: state of a cell in a given time t, such that $S_i^t \in \Sigma$;
- η_i^t: neighborhood of cell i, that is, state S_i^t of cell i together with the states of cells to which cell i is connected;
- $\Phi(\eta_i)$: transition rule de that gives the next state S_i^{t+1} for each cell i, as function of η_i^t.

An unidimensional CA is defined as a matrix data structure C of size $k(C_{1\times k})$. For each cell c_i, its neighborhood is defined as the cells standing at its left $c_i(c_{i-1}, c_{i-2}, \ldots, c_{i-k})$ and those at its right $c_i(c_{i+1}, c_{i+2}, \ldots, c_{i+k})$. For the $r = 1$, the neighborhood of cell c_i is formed by its left and right neighbors, that is, (c_{i-1}) and (c_{i+1}), respectively (see Fig. 1). For $r = 2$, the neighborhood comprehends (c_{i-2} and c_{i-1}) to the left, and (c_{i+1} and c_{i+2}) de c_i to the right. Similarly, higher-order neighborhoods is straightforwardly defined.

The state of a CA is the configuration of the lattice in a given instant of time. The initial state in t_0 will take to successive states in subsequent time steps $t_1, t_2, \ldots, t_n$, by successive iteration of the transition rule of the automaton. A state of the CA has a single successor state, as a consequence of the application of the transition rule. However, it can have an arbitrary number of predecessor states, known as pre-images.

Fig. 2 exemplifies the evolution of a unidimensional CA formed by 10 cells ($C_{1\times 10}$), during four time steps. Initially, at time zero, cells are randomly initialized by elements belonging to the binary set $(\Sigma = \{0, 1\}) : c_1 = 1; c_2 = 0; c_3 = 0; c_4 = 1; c_5 = 0; c_6 = 0; c_7 = 1; c_8 = 1; c_9 = 1; c_{10} = 0$. In this example we consider $r = 1$, consequently, $m = 3$, according to Eq. (1). Since the graphical representation of this CA is linear, its first (c_1) and last (c_{10}) cells do not have neighbors to the left and to the right, respectively. Therefore, according to the boundary condition, cell c_1 becomes the neighbor of cell c_{10}. This procedure assures that the transition rule $\Phi(\eta_i)$ can be applied over all cells of the CA, without continuity failure.

Fig. 2. Evolution of a unidimensional CA.

An important issue of CAs is the number of transitions that composes the rule Φ is obtained by k^{2r+1}, and the number of rules represented by those transitions is $k^{k^{2r+1}}$. The CA presented in Fig. 2 can be evolved for 256 ($k^{k^{2r+1}} = 2^{2^{2*1+1}} = 256$) different rules. For the take of example, it is shown rule 169. This number represents a rule according to the concept of elementary automata proposed by Wolfram (1983). Therefore, the binary number 10101001, extracted from the concatenation (from right to left, that is, from transition $111 \rightarrow 1$ to transition $000 \rightarrow 1$) of the eight outputs specified by rule Φ ([$000 \rightarrow 1$; $001 \rightarrow 0$; $010 \rightarrow 0$; $011 \rightarrow 1$; $100 \rightarrow 0$; $101 \rightarrow 1$; $110 \rightarrow 0$; $111 \rightarrow 1$]) is equivalent to the number 169 in decimal.

2.1. Dynamic Behavior of a CA

The dynamic behavior of a CA can be illustrated by a spatio-temporal diagram, in which the configuration of the states in a d-dimensional lattice is plotted as function of time. Usually, cells in state 0 are represented in white, and those in state 1 are represented in black, as shown in Fig. 3.

According to Wuensche and Lesser (1992), this diagram represents the local behavior of the system, since its global behavior is independent on the initial state. The global behavior can be obtained by the analysis of a (large) set of overlapped spatio-temporal diagrams, each one constructed starting from different initial state.

Wolfram (1984) proposed a standard terminology for classifying the dynamic behavior of CAs. By studying spatio-temporal diagrams, he has demonstrated that CAs can exhibit extremely complex behaviors, and he grouped them into four qualitative classes:

- Class 1: The evolution of the CA takes to homogeneous states in which, for instance, all cells have state 1 (Fig. 3a).
- Class 2: The evolution of the CA takes to a set of simple, periodic and stable structures (Fig. 3b).
- Class 3: The evolution of the CA takes to chaotic patterns (Fig. 3c).
- Class 4: The evolution of the CA takes to complex structures and, sometimes, with long periods (Fig. 3d).

Dynamic behaviors of class 1 and class 2 can be visually identifiable. However, the distinction between behaviors of class 3 and 4 is quite difficult, since there is no clear boundary between these classes.

Li and Parckard (1990) extended the classification previously proposed by Wolfram into six classes, using a more specific definition, as follows:

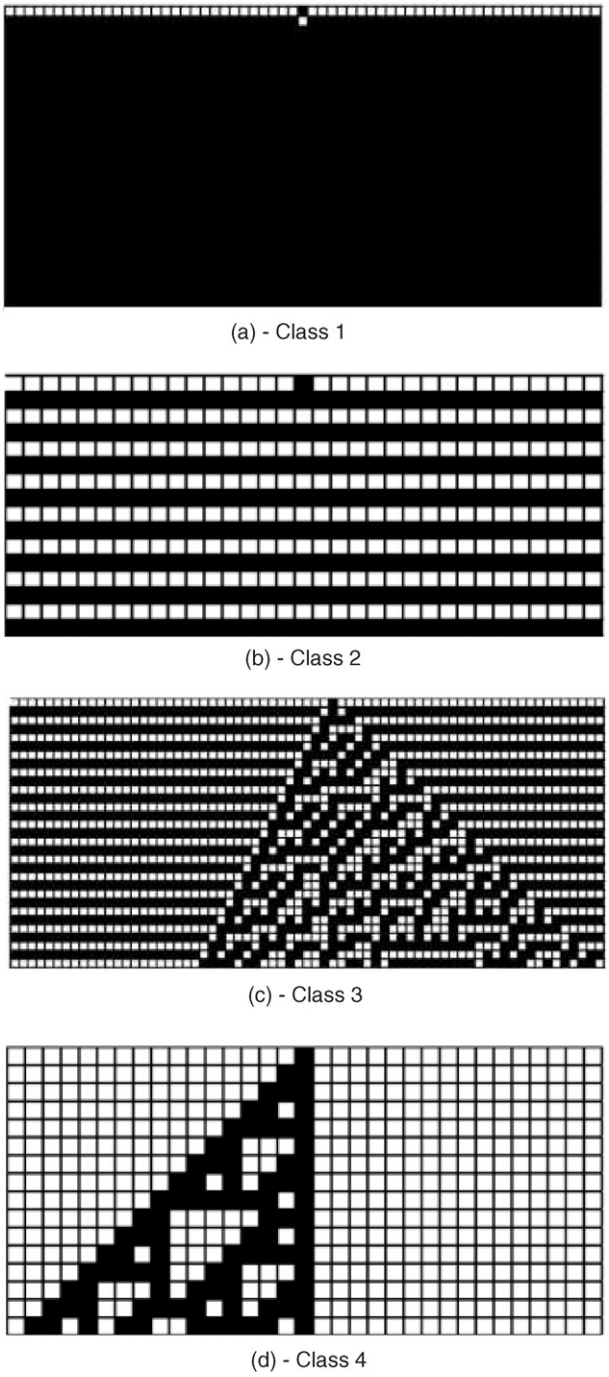

Fig. 3. Classification of the dynamic behaviors according to Wolfram (2002) (pp. 55–56). Time $t = 0$ is represented in the top of each figure and grows downwards.

- Null rules: After t iterations, all cells of the CA converge either to state 0 or state 1.
- Fixed-point rules: After t iterations, the application of a given rule maintains constant the configuration of the CA.
- Double-cycle rules: After t iterations, every two applications of the rule the CA returns to the same configuration.
- Periodic rules: After t iterations, each n repetitive applications of the rule, the CA returns to a given configuration, with $n \geq 3$.
- Chaotic rules: can produce non-periodic dynamic behaviors.
- Complex rules: can produce periodic dynamic behaviors, although they can have extremely long periods.

2.2. Parameters for Forecasting the Dynamic Behavior of CAs

The dynamic behavior forecasting parameters (DBFPs) are measures that give a quantitative evaluation of the effect of a transition rule in a CA, throughout iterations. The basic idea behind such parameters is to infer particular aspects of the dynamic behavior of a CA without the need to simulate them.

Next, a brief description of the main DBFPs found in recent literature is given, including: sensitivity (Binder, 1993), neighborhood domain, activity propagation and absolute activity (Oliveira et al., 2001), Wuensche's Z parameter (Wuensche, 1999), activity (Langton, 1990), mean field parameters (Li, 1991) and Li's Z parameter (Li, 1991). The mathematical formalization of such parameters can be found in the refereed citations.

Sensitivity measures the percent of similar transitions that are mapped to different states (Binder, 1993). A transition can be composed by n cells sequentially associated with a single output (s) in the form: $n_1, \ldots, n_n \rightarrow s$. Similar transitions to $n_1, \ldots, n_n \rightarrow s$ are all transitions that present a single neighbor cell different from the original transition. The analysis of similar transitions consists in counting the transitions that change s for a change in a single cell n, relative to the original transition.

The neighborhood domain (Oliveira et al., 2001) is a parameter used for quantifying the changes, due to the transition rule, in the central cell, taking into account the predominant state of the overall neighborhood.

The activity parameter was defined by Langton (1990), and refers to a statistical measure about the transitions of a rule. For binary CAs, the computation of this parameter consists in the sum of the output values of each transition, divided by the number of transitions that composes the rule.

The absolute activity (Oliveira et al., 2001) is based on the parameter activity, previously mentioned. It counts the number of transitions that takes to a state different from the current state of the central cell or of its neighbor cells.

The objective of the parameter Wuensche's Z parameter (Wuensche, 1999) is to quantify the propagation of perturbations in a given rule (Wuensche and Lesser, 1992; Wuensche, 1994). Such perturbation is computed as function of the pre-images in the state space. This parameter indicates whether the amount of pre-images is high (low Z) or low (high Z), for an arbitrary lattice configuration.

The concept of activity propagation is based on the two previous measures, and it is defined as the possibility of transition of a given cell, following the dominant state of the neighboring cells, and the possibility of this transition be sensitive to a minimal state change of its neighborhood.

Mean field parameters (Li, 1991) are formed by p components, such that $p = n + 1$, and n is the number of cells of the neighborhood. Each component is labeled as cm_p and stores the number of configurations (neighborhoods) that present p occurrences of state 1 in the n cells, and map this configuration to output with value 1.

The Z parameter defined by Li (1991) uses the same concept proposed by Binder (1993) for the parameter sensitivity. Although these parameters have been emerged in different works, both quantify the same information (Table 2 demonstrates this fact).

2.3. The Problem of Classification by the Majority Rule

The classical problem in CA literature known as classification by the majority rule can be modeled in different ways. In this work, we adopt a particular approach presented by Juillé et al. (1998), in which cells have binary states ($k = 2$), the unidimensional matrix structure of the CA is composed by a constant and predefined number of cells ($N = 149$), and the neighborhood radius is 3 ($r = 3$). Consequently, for this approach, the rule space of the CA is 2^{128}.

Table 1
Most efficient transition rules found in the literature, for the classification by the majority rule problem.

Rule	Hexadecimal	Reference	Efficiency
GKL	005F005F005F005F005FFF5F005FFF5F	Gacs et al. (1978)	81.60%
MHC	0504058705000F77037755837BFFB77F	Mitchell et al. (1993)	76.90%
DAV	002F035F001FCF1F002FFC5F001FFF1F	Davis (1991)	81.80%
DAS	070007FF0F000FFF0F0007FF0F310FFF	Das et al. (1995)	82.20%
ABK	0500550505005550555FF55FF55FF55FF	David et al. (1996)	82.30%
CRA	00550055005500571F55FF57FF55FF57	Cranny and Bossomaier (1999)	82.50%
JP1	011430D7110F395705B4FF17F13DF957	Juillé et al. (1998)	85.10%
JP2	1451305C0050CE5F1711FF5F0F53CF5F		86.00%
BOO1	145500CC0F14021F1715FFCF0F17FF1F		86.16%
BOO2	070017070C0057DF07BFD707CCFF559F		85.97%
BOO3	070017070C0057D707BFD707CDFF55D7	Bortot et al. (2004)	85.39%
BOO4	002F131F010FF91F00ECFF1F013DF91F		85.38%
BOO5	1071307C0000286F17313F7FF33F2B7F		85.35%
R1	015E0012005500571F5FFFFF0F55CF5F		82.70%
R2	10111000531512531F15FF5FDF5DDF5F		82.60%
R3	00010355015511571F150F77FFF5FF57		81.50%
R4	07044747070046070577475 7F777FF77		82.70%
R5	015400550050045F055FFFDF5557FF5F		81.90%
R6	0445004C37770E3F044500CDF7773FFF		81.50%
R7	150050003500770715537 75FF5F77F7F		81.70%
R8	000104171DDF555704DF441FDDDFD557		81.60%
R9	01000300113113 70DFFBFFBDDFF11FF	Oliveira et al. (2007)	81.50%
R10	0001090703030B031F1F6F37FF776F77		79.20%
R11	0100050D1D9D155F05FD555FDDFF5557		78.90%
R12	0001030211110113 17F5FFFFDDFF11FF		78.70%
R13	0001017D2113C35F4B15DF75275B9FD7		78.80%
R14	015500400054563F1057BF0FB7FFFB7F		77.80%
R15	0071023C00224D170379B53747BFFF7F		77.30%
R16	10041383005313DD3357CFED875F1FDF		78.70%
R17	040305502F06457D05013757D5F7FF7F		78.20%
R18	050470000006516D053FF5FF977FE77F		77.30%

The problem states that all cells must reach either state 0 or state 1 after M time steps, depending upon the density of the initial configuration of the CA. According to Mitchell et al. (1994), parameter c denotes a threshold for the classification. That is, if the number of 1's in the initial CA divided by the number of total cells of the same automaton if larger or equal to c, then all cells shall converge to state 1 after M time steps, otherwise, to state 0. In this work, we used $c = 0.5$, in the same way as in Juillé et al. (1998).

3. Correlation Analysis of the Dynamic Behavior Forecasting Parameters

This section aims at identifying possible statistical relationships between the DBFPs previously presented in Section 2.2. Considering that such parameters are computed as function of the transition rules of the CA, this analysis will use rules for the classification by the majority rule problem, according to the approach described in Section 2.3. This particular problem was chosen due to the availability of a large number of publications that present several different computational methods for finding interesting solutions for it. Therefore, it is possible to have a comparison of our proposed approach with another works. Another interesting feature of the cited problem is regarding the transition rules that are currently considered as good solutions for it. These rules have a behavior classified as "null", according to Li and Parckard (1990).

The rule space for the classification by the majority rule problem treated here is 2^{128} rules. This number of possibilities is too high to allow the systematic computation of the DBFPs for all rules. Therefore, after a deep search in recent literature, 31 rules were found, and all of them were submitted to the forthcoming analysis.

Table 1 presents these rules in hexadecimal format, together with the corresponding reference and their efficiency measure. Efficiency, in this context, was computed by evolving 10^4 random CAs for 200 iterations, for each rule. After, the final configuration is examined, and two particular situations are of interest:

- If the density of the initial configuration of the CA is lower than 0.5 and, after 200 iterations, all cells converge to 0.
- If the density of the initial configuration of the CA is higher or equal to 0.5 and, after 200 iterations, all cells converge to 1.

For these two situations, the score of the rule is increased, since its application to the CA really leads to the desired result. Any other situation is considered failed and the score of the rule is not increased. The efficiency of the rule is obtained by the division of the score divided by the number of trials (in our case, 10^4), therefore representing the percent of success of the rule.

Table 2 presents the computed values of the DBFPs for the reference rules presented in Table 1. The acronyms in Table 2 are: sensitivity (S), neighborhood dominance (DV), activity propagation (PA), absolute activity (AA), Wuensche's Z (Z_W), activity (A), mean field parameters (CM) and Li's Z (Z_Li). This table also shows the average values (that is, $Sm, DVm, PAm, AAm, Z\_Wm, Am, CMm, Z\_Lim$), the standard deviation, the maximum and the minimum of these parameters, computed over the 10^4 CAs. Finally, this table also shows the Euclidean distance (DE) between values of the DBFP set and the corresponding average values. DE is defined by Eq. (2) and is aimed as a measure of similarity between the values of the individual parameters of each rule (P_i) and the average values (Pm_i). In this procedure, 8 dimensions are considered, each one corresponding to a parameter. The smaller the value of DE, the more similar will be the rule parameters regarding the average values:

$$DE = \sqrt{\sum_{i=1}^{8} (P_i - Pm_i)^2} \tag{2}$$

Table 2
Computation of the dynamic behavior forecasting parameters—DBFPs.

Rule	*S*	*DV*	*PA*	*AA*	*Z_W*	*A*	*CM*	*Z_Li*	*DE*
GKL	0.23	0.91	0.07	0.10	0.25	0.50	0.50	0.23	0.27
MHC	0.37	0.91	0.08	0.18	0.54	0.49	0.49	0.37	0.11
DAV	0.30	0.88	0.09	0.16	0.24	0.50	0.50	0.30	0.23
DAS	0.25	0.87	0.10	0.22	0.38	0.50	0.50	0.25	0.13
ABK	0.23	0.88	0.09	0.20	0.50	0.50	0.50	0.23	0.13
CRA	0.25	0.88	0.09	0.21	0.47	0.50	0.50	0.25	0.10
JP1	0.40	0.85	0.11	0.25	0.48	0.51	0.51	0.40	0.13
JP2	0.33	0.84	0.11	0.26	0.48	0.50	0.50	0.33	0.08
BOO1	0.33	0.84	0.11	0.26	0.48	0.50	0.50	0.33	0.08
BOO2	0.34	0.84	0.10	0.26	0.49	0.51	0.51	0.34	0.08
BOO3	0.33	0.85	0.10	0.26	0.47	0.51	0.51	0.33	0.07
BOO4	0.34	0.85	0.12	0.22	0.34	0.50	0.50	0.34	0.14
BOO5	0.35	0.85	0.11	0.22	0.41	0.49	0.49	0.35	0.08
R1	0.29	0.87	0.11	0.19	0.50	0.50	0.50	0.29	0.06
R2	0.30	0.87	0.10	0.26	0.49	0.50	0.50	0.30	0.07
R3	0.30	0.92	0.06	0.17	0.46	0.50	0.50	0.30	0.07
R4	0.30	0.88	0.09	0.24	0.39	0.51	0.51	0.30	0.09
R5	0.28	0.91	0.08	0.15	0.45	0.49	0.49	0.28	0.08
R6	0.30	0.88	0.09	0.18	0.52	0.49	0.49	0.30	0.06
R7	0.30	0.88	0.10	0.26	0.46	0.50	0.50	0.30	0.07
R8	0.30	0.88	0.10	0.17	0.44	0.50	0.50	0.30	0.05
R9	0.30	0.89	0.08	0.16	0.56	0.49	0.49	0.30	0.11
R10	0.30	0.90	0.08	0.21	0.50	0.48	0.48	0.30	0.05
R11	0.30	0.90	0.07	0.17	0.50	0.52	0.52	0.30	0.07
R12	0.29	0.89	0.08	0.18	0.61	0.48	0.48	0.29	0.15
R13	0.40	0.89	0.07	0.19	0.48	0.49	0.49	0.40	0.12
R14	0.36	0.89	0.08	0.17	0.55	0.50	0.50	0.36	0.11
R15	0.40	0.91	0.10	0.17	0.51	0.48	0.48	0.40	0.13
R16	0.38	0.88	0.06	0.18	0.48	0.50	0.50	0.38	0.10
R17	0.36	0.90	0.06	0.20	0.47	0.49	0.49	0.36	0.07
R18	0.36	0.87	0.08	0.20	0.55	0.49	0.49	0.36	0.10

	Sm	*DVm*	*PAm*	*AAm*	*Z_Wm*	*Am*	*CMm*	*Z_Lim*
Average	0.32	0.88	0.09	0.20	0.47	0.50	0.50	0.32
Standard-deviation	0.05	0.02	0.02	0.04	0.08	0.01	0.01	0.05
Maximum	0.40	0.92	0.12	0.26	0.61	0.52	0.52	0.40
Minimum	0.23	0.84	0.06	0.10	0.24	0.48	0.48	0.23

The analysis of Table 2 shows a straight relationship between parameters *S* and *Z_Li* and parameters *A* and *CM*. The values of *S* remain identical to the values of *Z_Li*. In the same way, values of *A* are identical to *CM*, for all rules under analysis. This relationship between parameters suggest the exclusion of some parameters from further analysis. We choose to exclude *Z_Li* and *CM* because both present a high complexity level for computation, when compared with *S* and *A* respectively.

Observing the average, standard deviation, maximum and minimum values for each parameter, as well as DE in Table 2, it can be concluded that most rules present values very close each other, except for parameter *Z_W*, which has high variation.

For a deeper analysis of correlations between the values of DBFPs previously presented (excluding *Z_Li* and *CM*), Table 3 shows a correlation matrix. The computation of this matrix is based on the Pearson's correlation coefficient (*r*) (Ahlgren et al., 2003).

According to the correlation matrix shown, there is no high correlation between parameters *S*, *DV*, *PA*, *AA*, *Z_W* and *A*. It is worth to mention that Vincent (2005) defined high correlation for absolute values higher than 0.9 in the matrix. This suggests the hypothesis that all the mentioned parameters are equally important for describing the dynamic forecasting behavior of a CA.

Table 3
Correlation matrix of the DBFPs.

	S	*DV*	*PA*	*AA*	*Z_W*	*A*
S	1.00	−0.16	0.02	0.21	0.30	−0.14
DV		1.00	−0.75	−0.76	0.06	−0.39
PA			1.00	0.60	−0.14	0.18
AA				1.00	0.15	0.32
Z_W					1.00	−0.30
A						1.00

4. Transition Rule Induction Using DBFPs

In this section two basic issues are addressed: (a) what is the real usefulness of the DBFPs for identifying specific dynamic behaviors? and (b) Is it possible to emerge different behaviors for rules with the same DBFPs values?

The objective of these questions is to evaluate how the DBFPs can be used in the process of rule induction. That is, if it is possible to use previous knowledge about the desired dynamic behavior to devise transition rules, without the need of exhaustive simulation of the CA.

To answer the questions, an evolutionary rule induction system was developed to find transition rules that present "null" behaviors. The proposed system is described in details in the next subsection.

4.1. Rule Induction with Evolutionary Computation

In mathematics, the term "induction" is understood as a reasoning by which a given property of one element is extended to all elements of a set. In this context, it is possible to generalize conclusions obtained from the analysis of a set of premises. According to Quinlan (1986), the rule induction problem consists in finding classification rules capable of determining the class of objects by analyzing the values of their attributes. The set of attributes that describe an object usually measure or identify a relevant feature of it. Many algorithms were proposed for rule induction, including evolutionary computation (EC) methods.

EC is a growing field of research in computer science, with many successful applications to real-world problems. Most of paradigms in EC borrow inspiration from natural process of evolution. Amongst those paradigms, the genetic algorithm (GA) is, possibly, the most widely known (Holland, 1975; Goldberg, 1989), since it has been successfully applied to many optimization problems. A direct descendent of GAs is genetic programming (GP), devised by Koza (1992, 1994). The idea behind GP is induce computers to find solutions, in the form of programs, to problems for which they were not previously programmed to. That is, based on a set of pairs *inputs*, *output*, induce a program capable of incorporating a model of the problem and generating outputs for other, previously unseen inputs. To each program a fitness value is associated, based on its ability to provide good solutions to the problem in hand. Due to its distinguishing characteristics, genetic programming have been used for complex rule-induction problems in several domains (see, for instance, Bojarczuk et al., 2004), thus suggesting its applicability in finding transition rules for CAs.

An extension of GP, known as gene-expression programming (GEP) was proposed by Ferreira (2001), uses concepts from both GP and GA. The fundamental difference between these three approaches is how individuals are represented. In a GA, individuals are usually represented as a linear fixed-size string of bits (although other representations can also be found). In GP, individuals are nonlinear structures of different sizes and shapes, in the form of a tree. In GEP, individuals are first encoded as a linear string of fixed-size (genome or chromosome), which is later expressed as non-linear structures of different sizes and shapes (expression trees).

GEPCLASS (Weinert and Lopes, 2006) is an implementation of GEP specifically designed for finding rules for classification problems based on supervised learning. In classification problems it is aimed to find a rule or a set of them capable of model a given domain of known samples, and then classify unseen sets of data of the same nature.

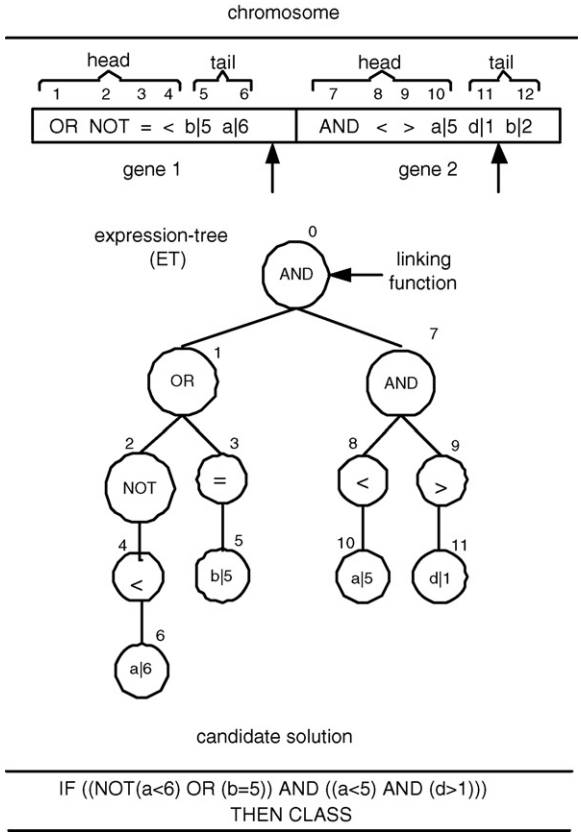

Fig. 4. Example of the structure of a chromosome, the expression tree and its corresponding rule in GEPCLASS.

GEPCLASS implements a structure where a population of individuals is evolved for a number of generations. At each generation several genetic operators is applied to selected individuals of the population, generating diversity and, hopefully, improving the quality of the overall population, regarding a given problem. Following Ferreira (2001), an individual is composed of a single chromosome that, in turn, is composed of n genes. Each gene is divided into two parts: head and tail. The head of the gene can have elements belonging only to the set of functions (operations), such as: *AND*, *OR*, *NOT*, $=$, $\neq$, $>$ and $<$. The tail, in turn, can have elements either from the set of functions or from the set of terminals. The set of terminals includes the attributes that describe the problem and particular values (such as constants).

GEP uses the biological concept of open read frame ORF as encoding principle. That is, the sequence of elements at the genotypical level has the potential to express phenotypically, but not necessarily all them will be used. Therefore, the encoding region of a gene (ORF) can "activate" or "deactivate" portions of the genetic material during evolution.

The mapping between the genotype to the phenotype is carried out as follows. The whole chromosome is transcribed into a variable-size expression tree. The genetic material used in this process is that one defined by the ORF in each gene. The transcription rules follow the standard of the Karva language (Ferreira, 2001), where each gene is transcribed to a separated sub-tree. Next, all sub-trees are joined together by a linking function (usually, *AND* or *OR*), thus composing the expression tree. This tree represents a candidate solution for the problem in hand and, when evaluated by a fitness function, the quality of this solution will be objectively estimated. The transcription process in GEP is illustrated in Fig. 4. For gene 1 of this figure the ORF ends at position 6, and for gene 2, it ends at position 11. The ORF of a gene is determined by the functions and their respective arities (number of arguments) encoded in the head of the gene. The elements of a gene are read from leftmost position towards right. For instance, in gene 1 of the figure there is a logical operator (*OR*), which arity is 2, meaning that two arguments are necessary for evaluating this function. Therefore, the next two operators after *OR* (that is, *NOT* at position 2 and $=$ at position 3) belong their arguments. Recursively, one can determine the arguments of *NOT* and $=$, and so on. This procedure is repeated until there are no more functions without arguments. The size of the tail of the gene is determined according to the size of the head (Weinert and Lopes, 2006). Therefore, it is warranted that there will not be functions with missing attributes encoded in the head of the gene. It is important to noted that a precedence relationship must be defined for logical operators, relational operators and terminals.

The genetic operators implemented in GEPCLASS are: mutation, recombination and transposition. All of them comply with the closure property (Koza, 1992), thus finding valid rules. That is, the hierarchical structure of the expression trees is always maintained after the application of the operators (Weinert and Lopes, 2006). The fitness function implemented in GEPCLASS was the same used for other rule induction problems, and is shown in Eq. (3):

$$\text{fitness} = \frac{tp}{tp+fn} * \frac{tn}{tn+fp} \tag{3}$$

where

- *tp* (true positive): the rule predicts that the instance belongs to a class, and it indeed belongs;
- *fp* (false positive): the rule predicts that the instance belongs to a class, but in fact it does not;
- *tn* (true negative): the rule predicts that the instance does not belong to a class, and indeed it does not;
- *fn* (false negative): the rule predicts that the instance does not belong to a class, but in fact it does.

Further details of the genetic operators of GEPCLASS and their closure property, as well as the fitness function implemented for classification problems can be found in Weinert and Lopes (2006).

4.2. Implementation

GEPCLASS was modified to be applied to the classification by the majority rule problem. The instance of this problem used in our work was the same suggested by Juillé et al. (1998): a unidimensional CA with 149 cells and radius 3. The running parameters of GEPCLASS were:

- Number of individuals: 50.
- Number of generations: 50.
- Linking function: *AND*.
- Function set: *AND*, *OR*, *NOT*, $=$ and $\neq$.
- Terminal set: *a*, *b*, *c*, *d*, *e*, *f*, *g*.
- Number of genes per chromosome: 3.
- Size of head: 6–15.
- Selection method: stochastic tournament.
- Genetic operators: the same defined in Ferreira (2001), with the same probabilities.

The terminal set (*a*, *b*, *c*, *d*, *e*, *f*, *g*) are binary and represent all the possible combinations of neighborhood of the problem, thus mapping all transitions of a given rule. The number of combinations is 128, obtained as shown in Section 2. The definition of these 128 binary combinations, each one with 7 bits (one bit for each terminal), can be easily generated converting from binary to decimal base. That is, the conversion of the decimal numbers present in the interval 0 . . . 127 to their corresponding binary numbers, define

Table 4
State transition table.

#	Transitions							Rule
	a	*b*	*c*	*d*	*e*	*f*	*g*	
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	1
⋮								⋮
127	1	1	1	1	1	1	0	0
128	1	1	1	1	1	1	1	0

the 128 possible neighborhood combinations, according to Table 4. For instance, a possible rule encoded in an individual of GEPCLASS would be:

if $((a = 0)$ *AND* $(b = 1))$ *then*
Rule:=1; *else Rule*:=0;

In fact, there is no restriction regarding the number of terminals that can appear in the rule. In this simple example only the terminals *a* and *b* (see Table 4) were considered. Any terminal that do not appear in a rule can effectively assume any value, since they do not influence the final outcome of the rule.

The concatenation of all 128 output (the last column of Table 4), that is, 00 . . . 10 forms the rule that is said a candidate solution to the problem in hand (see Section 2.3). Next, the candidate solution is submitted to the fitness computation procedure.

The fitness function is computed using the following DBFPs *S*, *DV*, *PA*, *AA*, *Z_W* and *A*. Since there is no known optimal set of values for these parameters, the average values were adopted as reference, as shown in Table 2. Eq. (4) presents the fitness function used in this work.

$$\text{fitness} = Sn * DVn * PAn * AAn * Z\_Wn * An \quad (4)$$

Acronyms in this equation are those defined in Section 3 and the subscript *n* indicates that values are normalized in the range [0 . . . 1] according to Table 5 and the corresponding average values. The fitness function has the property of returning 1 only when all parameters are 1. The basic idea in this normalization is to find linear equations that return a high fitness for rules that have the values of the DBFPs around to the average values, and low fitness when the DBFPs are far from average. Fig. 5 presents plots illustrating how the linear adjustment of the DBFPs are done.

5. Results and Discussion

In the analysis of the DBFPs in Section 3 it was observed that parameters *S* and Z_L_i, and *A* and *CM* are numerically equivalent, for the same rule. This finding allowed the exclusion of parameters Z_L_i and *CM* from the remaining analysis.

Once that all the 31 rules analyzed exhibit "null" behavior and there are no reference values for the DBFPs computed with these rules, the average values were adopted as reference. It is fair to use average values since the Euclidean distance between the parameters of the 31 rules and the average values of the parameters is small for most cases.

The analysis of correlation between parameters revealed that parameters *DV*, *PA*, *AA*, *Z_W* and *A* really provide distinct information in the process of forecasting the dynamic behavior or CAs.

A GEP-based system (GEPCLASS) was developed for transition rule induction, thus finding rules with "optimized" DBFPs. The fitness function of GEPCLASS was changed according to the specific forecasting parameter to be optimized.

Initially, GEPCLASS was run with the parameters mentioned in Section 4.2. We obtained 6 rules for the classification by the majority rule using a unidimensional CA with 149 cells and radius 3. These rules are presented in Table 6 in hexadecimal format with the respective values of efficiency. Efficiency was computed by means of the simulation of 10^4 CAs during 200 iterations, starting with random configurations.

Each one of the rules presented in Table 6 is regarding the optimization of a given DBFP. For instance, for Rule_S GEPCLASS found a rule whose *S* value is as near as possible to the value of *Sm*. Since that other parameters were not considered in this experiment, the fitness function shown in Eq. (4) was reduced to: *fitness* = *Sn*. Similar procedure was adopted for all the other DBFPs.

Table 5
Linear adjustment of the DBFPs.

Normalized parameter	Pseudocode
Sn	*If* $(S <= Sm)$ *Then* $Sn := 3.125 * S$; *Else* $Sn := -1.470 * S + 1.470$;
DVn	*If* $(DV <= DVm)$ *Then* $DVn := 1.136 * DV$; *Else* $DVn := -8.333 * DV + 8.333$;
PAn	*If* $(PA <= PAm)$ *Then* $PAn := 11.112 * PA$; *Else* $PAn := -1.098 * PA + 1.098$;
AAn	*If* $(AA <= AAm)$ *Then* $AAn := 5 * AA$; *Else* $AAn := -1.250 * AA + 1.250$;
Z_Wn	*If* $(Z\_W <= Z\_Wm)$ *Then* $Z\_Wn := 2.127 * Z\_W$; *Else* $Z\_Wn := -1.886 * Z\_W + 1.886$;
An	*If* $(A <= Am)$ *Then* $An := 2 * A$; *Else* $An := -2 * A + 2$

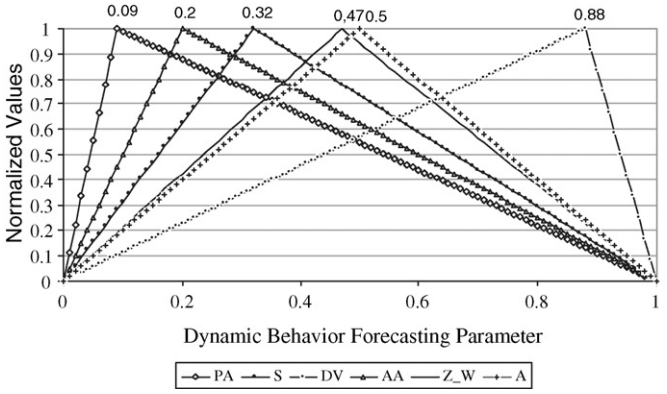

Fig. 5. Linear adjustment of the DBFPs.

Table 6
Transition rules found by GEPCLASS considering parameters individually.

Rule	Hexadecimal	Efficiency
Rule_S	0505A5A50F0FF0F00505A5A50F0FF0F0	0.00%
Rule_DV	00110033555555550515053755555555	50.30%
Rule_PA	0000000007770FFF0000000007770FFF	50.33%
Rule_AA	0CFF0CFF0CFF0CFF3FFF3FFF3FFF3FFF	11.08%
Rule_ZW	FF33FF73FF33FF50FF33FF73FF33FF50	0.00%
Rule_A	AABB0031BBBB3333AABA0030BBBB3333	0.00%

Table 7
Values for the parameters of rules presented in Table 6.

Rule	*S*	*DV*	*PA*	*AA*	*Z_W*	*A*	*DE*
Rule_S	**0.32**	0.62	0.19	0.40	0.69	0.44	0.41
Rule_DV	0.22	**0.88**	0.08	0.18	0.69	0.39	0.27
Rule_PA	0.20	0.87	**0.09**	0.18	0.66	0.33	0.28
Rule_AA	0.18	0.79	0.16	**0.20**	0.44	0.75	0.31
Rule_Z_W	0.19	0.38	0.33	0.67	**0.47**	0.73	0.77
Rule_A	0.24	0.52	0.27	0.45	0.55	**0.50**	0.49

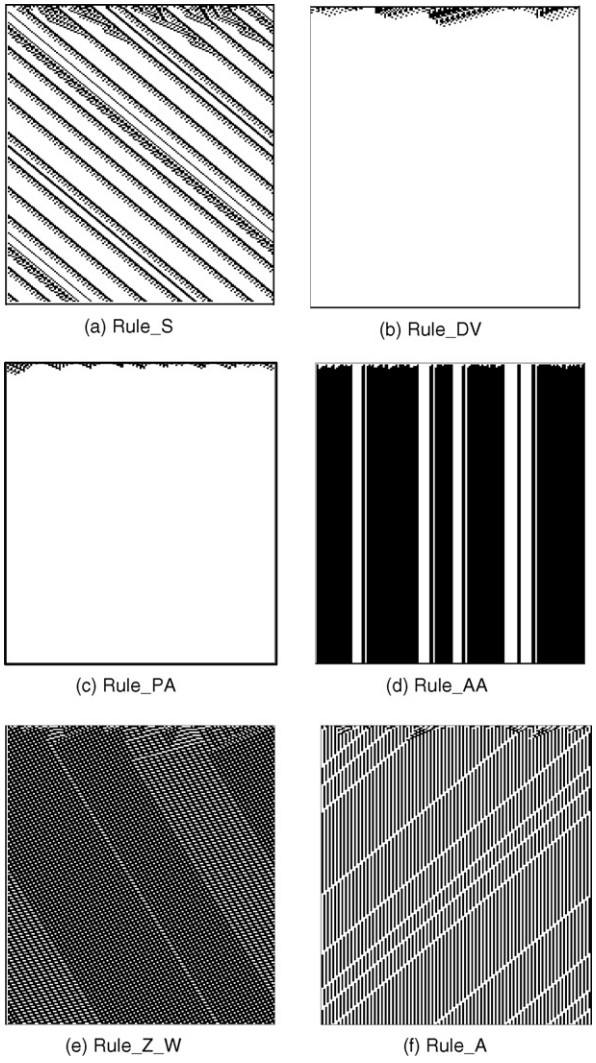

Fig. 6. Simulation of the dynamic behavior of rules shown in Table 6.

Table 7 shows the values of the DBFPs regarding to the rules previously shown in Table 6.

Fig. 6 presents an example of the simulation of the dynamic behavior for each one of the rules previously shown in Table 6.

Although GEPCLASS did not have difficulty in finding the 6 rules, the efficiency of them, as well as their dynamic behavior, indicate that they are not solutions for the classification by the majority rule problem. Also, not all rules present a "null" behavior. Therefore, these results indicate that the use of DBFPs individually, as the only source of information for the rule induction procedure, is inefficient.

The efficiency rate for the rules found were far bellow the expected values. For instance, rules Rule_DV and Rule_PA had efficiency rate around 50%. However, observing the simulations of these rules in Fig. 6 and the value of parameter "activity" in Table 7, it becomes clear that such rules can take any initial configuration of the CA to a state where all cells are 0, after 200 iterations. This do not solve the problem, although a "null" behavior can be identified. Rule_S, Rule_AA, Rule_Z_W and Rule_A besides presenting very low values for efficiency, they lead the CA to periodic behavior, as shown in Fig. 6.

Another experiment was done using GEPCLASS to find rules for the same problem. However, differently from those rules previously shown in Table 6, now all parameters were jointly considered by using the fitness function of Eq. (4). The top 10 rules found by GEPCLASS are shown in Table 8 and the corresponding DBFPs computed for these rules are shown in Table 9. Examples of the dynamic behavior of CAs for these rules 2, 3, 4, 6, 8 and 10 are shown in Fig. 7.

Table 8
Transition rules found by GEPCLASS considering all parameters together.

Rule	Hexadecimal	Efficiency
Rule_1	111110101111F3101F1F1F1F1F1FFF1F	2.83%
Rule_2	000F000F555F555FCCCFCCCFDDDFDDDF	0.00%
Rule_3	37370113377F015F37370113377F015F	49.96%
Rule_4	0AAA0FFF011105550AAA0FFF0BBB0FFF	0.00%
Rule_5	00000F0FFFFF0F1F00110F1FFFFF0F1F	1.83%
Rule_6	3333000000000F0F3F3F3F3F3F3F3F	49.33%
Rule_7	0A5F0A5F0A5FFFFF005500550A553B77	0.00%
Rule_8	0033F0330033F0330033FFFF0033FFFF	49.73%
Rule_9	0A0F0A0F000F000F3F3F3F3F3F3F3F	1.62%
Rule_10	331133113311331111FF11FF11FF11FF	49.33%

The difficulty for GEPCLASS to find acceptable solutions was significantly higher than in the previous experiment, since more degrees of freedom were added to the problem. Only Rule_8 and Rule_10 were capable of maintaining values within the [maximum, minimum] range established as reference (see Table 2). Although these rules presented a "null" behavior in the dynamic simulation, they achieved an efficiency rate around 50%. Observing the simulation of these rules (Fig. 7d–f) it is clear that these rules can take any initial configuration of a CA to a state in which all cells are 0, after 200 iterations.

For all the rules found in this experiment, the parameter "sensitivity" was quite different from the reference value. However, observing Table 2 we note that some rules, such as GKL, DAS, ABK and CRA have values of the DBFPs very close to those of the 10 rules found by GEPCLASS (Table 8). Therefore, the use of an "optimized" set of DBFPs in the rule induction procedure do not take important benefits to the quality of solutions.

6. Conclusions and Future Work

A total of 31 rules were used in the analysis of eight DBFPs. The main objective of this analysis was to verify the capacity of each parameter individually and the group of parameters, for forecasting the dynamic behavior of CAs. A classical problem in CA literature was selected for this study: the classification by the majority rule. This problem is interesting because the solutions for it are known as "null" behavior rules, according to Li and Parckard (1990). That is, independently of the initial configuration of the system, it will converge to a state where all cells of the CA will take the same value (either 0 or 1) after a transient period of successive iterations.

According to the results obtained, it can be concluded that the DBFPs studied in this work, when considered individually or together, do not provide enough information for classifying efficiently the dynamic behavior of a CA. Although this conclusion is

Table 9
Values of the DBFPs for rules presented in Table 8.

Rule	*S*	*DV*	*PA*	*AA*	*Z_W*	*A*	*DE*
Rule_1	0.26	0.86	0.09	0.26	0.46	0.47	0.09
Rule_2	0.24	0.82	0.13	0.22	0.49	0.58	0.14
Rule_3	0.26	0.82	0.13	0.22	0.46	0.52	0.10
Rule_4	0.27	0.80	0.13	0.22	0.49	0.52	0.11
Rule_5	0.24	0.84	0.11	0.24	0.46	0.54	0.11
Rule_6	0.25	0.85	0.10	0.25	0.50	0.50	0.10
Rule_7	0.27	0.81	0.13	0.21	0.47	0.51	0.10
Rule_8	0.23	0.85	0.10	0.20	0.45	0.50	0.10
Rule_9	0.22	0.86	0.09	0.20	0.47	0.53	0.11
Rule_10	0.25	0.85	0.10	0.20	0.50	0.50	0.08

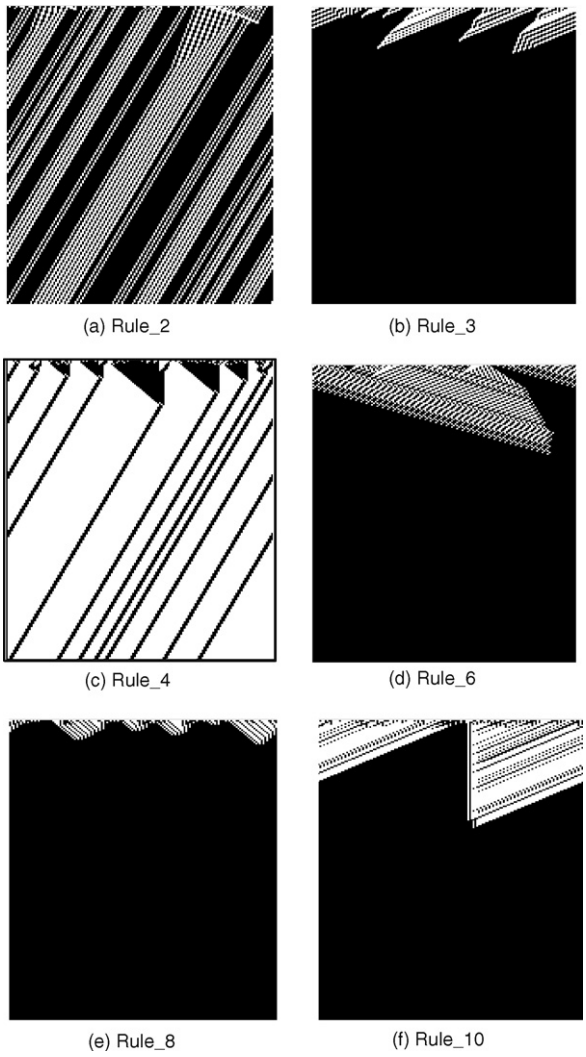

Fig. 7. Simulation of the dynamic behavior of rules 2, 3, 4, 6, 8 and 10, show in Table 8.

specific for the problem of classification by the majority rule, it is fair to expect that it can be extended to other classes of problems dealt by CAs.

It was observed that different transition rules having the values of the DBFPs very close each other can display distinct dynamic behavior in the simulation. These results found suggest that it is not possible to identify a clear boundary of values for the DBFPs studied for identifying transition rules with "null" behavior.

Consequently, this fact may have a relevant impact in approaches that use DBFPs in the process of rule-induction for CAs, such as the evolutionary computation approaches previously mentioned. The way the DBFPs were used in this work requests a set of reference values for them. Possibly, this reference set does not exist for the problem treated here, and shall be addressed in future work. Results obtained in this work suggest the need of further exploration in DBFPs. More expressive parameters, regarding the full dynamic behavior of CAs, are still needed.

A possible line of investigation is devising parameters not completely dependent on the transition rules, but that take into account features captured by sampling a simulation of the CA. Since simulations are computationally expensive, further development will have to address this issue, either using reconfigurable hardware for accelerating simulations (such as in Weinert et al., 2007) or using parallel computing.

Acknowledgements

This work was partially supported by the Brazilian National Research Council under grant no. 309262/2007-0 to H.S. Lopes.

References

Ahlgren, P., Jarneving, B., Rousseau, R., 2003. Requirements for a cocitation similarity measure, with special reference to pearson's correlation coefficient. Journal of the American Society for Information Science and Technology 54 (6), 550–560.
Beauchemina, C., Samuelb, J., Tuszynskia, J., 2005. A simple cellular automaton model for influenza a viral infections. Journal of Theoretical Biology 232, 223–234.
Benkiniouar, M., Benmohamed, M., 2004. Cellular automata for cryptograpy. In: Proceedings of the International Conference on Information and Communication Technologies: From Theory to Applications, Damascus, Syria, pp. 423–424.
Binder, P., 1993. A phase diagram for elementary cellular automata. Complex Systems 7, 241–247.
Bojarczuk, J., Lopes, H., Freitas, A., 2004. A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets. Artificial Intelligence in Medicine 30, 27–48.
Bortot, J., de Oliveira, P., Oliveira, G., 2004. Multiobjective, heuristic evolutionary search in a cooperative environment leads to the best cellular automaton rule in the density classification task. In: Proceedings of the VIIIth Brazilian Symposium on Neural Networks. IEEE Press/SBC, São Luís (CD-ROM: Paper 3565).
Chen, Q., Mynett, A., 2003. Effects off cell size and configuration in cellular automata based pray–predator modelling. Simulation Modelling Practice and Theory 11, 609–625.
Corne, D., Frisco, P., 2008. Dynamics of HIV infection studied with cellular automata and conformon-p system. BioSystems (91), 531–544.
Cranny, T., Bossomaier, T., 1999. The density classification problem for cellular automata: searching within structure. Tech. rep., Charles Sturt University.
Das, R., Crutchfield, J., Mitchell, M., Hanson, J., 1995. Evolving globally synchronized cellular automata. In: Proceedings of the Sixth International Conference on Genetic Algorithms, pp. 336–343.
David, A., Forrest, B., Koza, H., 1996. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In: Proceedings of the First Annual Conference the Genetic Programming 1996. MIT Press, Cambridge, MA, pp. 3–11.
Davis, L., 1991. Handbook of Genetic Algorithms, 1st edition. Van Nostrand Reinhold.
Ferreira, C., 2001. Gene expression programming: a new adaptative algorithm for solving problems. Complex Systems 13, 87–129.
Ferreira, C., 2002. Discovery of the Boolean functions to the best density-classification rules using gene expression programming. Lecture Notes in Computer Science 2278, 51–60.
Fu, S., Milne, G., 2003. Epidemic modeling using cellular automata. In: Proceedings of the 1st Australian Conference on Artificial Life, pp. 43–57.
Gacs, P., Kurdyumov, G., Levin, L., 1978. One dimensional uniform arrays that wash out finite islands. Problemy Peredachi Informatsii 12, 92–98.
Gangadhar, D., 2005. Pelican - protein-structure alignment using cellular automata models. In: Proceedings of the International Conference in Adaptive and Natural Computing Algorithms, Coimbra, Portugal, pp. 308–311.
Georgoudas, I., Sirakoulis, G., Andreadis, I., 2007. Modelling earthquake activity features using cellular automata. Mathematical and Computer Modelling 46, 124–137.
Goldberg, D., 1989. Genetic Algorithms in Search, Optimization, and Machine Learning, 1st edition. Addison-Wesley Professional.
Hofman, K., Bucher, P., Falquet, L., Bairoch, A., 1999. The prosite database, its status in 1999. Nucleic Acids Research 27, 215–219.
Holland, J., 1975. Adaptation in Natural and Artificial Systems, 1st edition. The University of Michigan Press.
Juillé, H., Pollack, J., 1998. Coevolving the ideal trainer: application to the discovery of cellular automata rules. In: Koza, J. (Ed.), Proceedings of the Third Annual Conference in Genetic Programming. Morgan Kaufmann, San Francisco, pp. 519–527.
Kansal, A.R., Torquato, S., Harsh, G.R., IV, Chiocca, E.A., Deisboek, T.S., 2000. Cellular automaton of idealized brain tumor growth dynamics. BioSystems 55, 119–127.
Khan, A., Choudhury, P., Dihidar, K., Verma, R., 1999. Text compression using two-dimensional cellular automata. Computers and Mathematics with Applications 37 (6), 115–127.
Kiera, L., Chenga, C., Testab, B., Carruptb, P., 1996. A cellular automata model of enzyme kinetics. Journal of Molecular Graphics 14, 227–231.
Koza, J., 1992. Genetic Programming: on the Programming of Computers by Means of Natural Selection, 1st edition. MIT Press.
Koza, J., 1994. Genetic Programming. II. Automatic Discovery of Reusable Programs, 1st edition. MIT Press.
Langton, C., 1990. Computation at the edge of chaos: phase transitions and emergent computation. Physica D 42, 12–37.
Laurio, K., Linaker, F., Narayanana, A., 2007. Regular biosequence pattern matching with cellular automata. Information Sciences 146, 89–101.
Li, W., 1991. Parameterizations of cellular automata rule space. Tech. rep., Santa Fe Institute Tech.
Li, W., Parckard, N., 1990. The structure of elementary cellular automata rule space. Complex Systems 4 (3), 281–297.
Malleta, D., Pillisb, L.D., 2006. A cellular automata model of tumor–immune system interactions. Journal of Theoretical Biology 239, 234–350.

Mitchell, M., 1996. Computation in cellular automata: a select review. In: Gramss, T. (Ed.), Nonstandard Computation. VHC Verlagsgesellschaft, Weinheim, pp. 95–140.
Mitchell, M., Crutchfield, J., Hraber, P., 1994. Evolving cellular automata to perform computations: mechanisms and impediments. Physica D 75, 361–391.
Mitchell, M., Hraber, P., Crutchfield, J., 1993. Revisiting the edge of chaos: evolving cellular automata to perform computations. Complex Systems 7, 89–130.
Mizas, C., Sirakoulis, G.C., Mardires, V., Karafyllidis, I., Glykos, N., Sandaltzopoulos, R., 2008. Reconstruction of DNA sequence using genetic algorithms and cellular automata: towards mutation prediction? BioSystems 92, 61–68.
Morales, F., Crutchfield, J., Mitchell, M., 2001. Evolving two-dimensional cellular automata to perform density classification: a report on work in progress. Parallel Computing 27, 571–585.
Oliveira, G., Asakura, O., de Oliveira, P., 2002a. Dynamic behaviour forecast as a driving force in the coevolution of one-dimensional cellular automata. In: Proceedings of the VIIth Brazilian Symposium on Neural Networks, vol. 1. IEEE Computer Society, Los Alamitos, CA, USA, pp. 98–103.
Oliveira, G., Bortot, J., de Oliveira, P., 2002b. Multiobjective evolutionary search for one-dimensional cellular automata in the density classification task. In: Proceedings of the 8th International Conference on Artificial Life. MIT Press, Cambridge, MA, USA, pp. 202–206.
Oliveira, G., Bortot, J., de Oliveira, P., 2007. Heuristic search for cellular automata density classifiers with a multiobjective evolutionary algorithm. In: Proceedings of the VIth Congress of Logic Applied to Technology LAPTEC 2007, vol. 1, Santos, SP, pp. 1–12.
Oliveira, G., Omar, N., de Oliveira, P., 2001. Definition and application of a five-parameter characterization of one-dimensional cellular automata rule space. Artificial Life 7, 277–301.
Quinlan, J., 1986. Induction of decision tree. Machine Learning 1, 81–106.
Richards, F., Meyer, T., Packard, N., 1990. Extracting cellular automaton rules directly from experimental data. Physica D 45, 189–202.
Rosin, P., 2005. Training cellular automata for image processing. Lecture Notes in Computer Science 3540, 195–204.
Swiecicka, A., Seredynski, F., 2000. Cellular automata approach to scheduling problem. In: Proceedings of the International Conference on Parallel Computing in Electrical Engineering. IEEE Computer Society, Washington, DC, USA, pp. 29–33.
Tomassini, M., Perrenoud, M., 2001. Cryptography with cellular automata. Applied Soft Computing 1, 151–160.
Vincent, W., 2005. Statistics in Kinesiology. Human Kinetics, Champaign.
Wei, Y., Ying, S., Fan, Y., Wang, B., 2003. The cellular automata model of investment behavior in the stock market. Physica A 325, 507–516.
Weinert, W., Benitez, C., Lopes, H., Lima, C., 2007. Simulation of the dynamic behavior of one-dimensional cellular automata using reconfigurable computing. Lecture Notes in Computer Science 4419, 385–390.
Weinert, W., Lopes, H., 2006. GEPCLASS: a classification rule discovery tool using gene expression programming. Lecture Notes in Computer Science 4093, 871–880.
Wolfram, S., 1983. Cellular automata. Los Alamos Science 9, 2–21.
Wolfram, S., 1984. Universality and complexity in cellular automata. Physica D 10, 1–35.
Wolfram, S., 1986. Advances in cryptology. Lecture Notes in Computer Science 218, 429–432.
Wolfram, S., 2002. A New Kind of Science, 1st edition. Champaign, Wolfram Media.
Wuensche, A., 1994. Complexity in one-D cellular automata: gliders, basins of attraction and the *Z* parameter. Tech. Re 94-04-025, Santa Fe Institute.
Wuensche, A., 1999. Classifying cellular automata automatically: finding gliders, filtering, and relating space–time patterns, attractor basins and the *Z* parameter. Complexity 3, 47–66.
Wuensche, A., Lesser, M., 1992. The Global Dynamics of Cellular Automata. Addison-Wesley, Reading, USA.
Xiao, X., Shao, S., Ding, Y., Huang, A., Chen, X., Chou, K.-C., 2005. Using cellular automata to generate image representation for biological sequences. Amino Acids 28, 29–35.