5-2012

Information Flow in the Spatiotemporal Dynamics of Cellular Automata

Akshay Thakre
Utah State University

INFORMATION FLOW IN THE SPATIOTEMPORAL DYNAMICS OF CELLULAR AUTOMATA

by

Akshay Thakre

A report submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Dr. Nicholas Flann Major Professor	Dr. Curtis Dyreson Committee Member
Dr. Xiaojun Qi Committee Member	

UTAH STATE UNIVERSITY
Logan, Utah

2012

ABSTRACT

Information Flow in the Spatiotemporal Dynamics of Cellular Automata

by

Akshay Thakre, Master of Science

Utah State University, 2012

Major Professor: Dr. Nicholas Flann
Department: Computer Science

Decision making in natural systems, such as the body's immune response to a potential pathogen or a bacterial colony's initiation of fruiting due to food scarcity, is distributed over many cells that posses only local information, and not determined globally. Understanding how accurate decisions can be made in such systems where no individual decision maker has complete information has important implications in distributed software and can provide insights into the biological evolution of complexity. In this work, the process of distributed decision making is modeled using the majority problem in cellular automata, and information theoretic measures of Kolmogorov complexity are applied to quantify information flow during the decision making process. Results show that (a) when the decision making process converges the information content of the dynamics quickly reaches a peak then decays to near-zero; (b) if the process does not converge and becomes chaotic, information content oscillates over a large unstable range; (c)extensive statistically significant differences exist in information flow dynamics between convergent and chaotic outcomes; and (d) there are small, but statistically significant differences in information flow dynamics between convergence to the correct answer compared to convergence to the incorrect answer. This last result supports the hypothesis that correct decision making maximizes information flow among agents in distributed decision making.

(54 pages)

PUBLIC ABSTRACT

Decision making in natural systems, such as the body's immune response to a potential pathogen or a bacterial colony's initiation of fruiting due to food scarcity, is distributed over many cells that posses only local information, and not determined globally. Understanding how accurate decisions can be made in such systems where no individual decision maker has complete information has important implications in distributed software and can provide insights into the biological evolution of complexity. This work studies distributed decision making in solving the majority problem: given a line of text with only T and F characters, determine if there are more T characters than F characters. This problem is easily solved with a single global decision maker by counting the number of characters. However, this problem is much harder to solve with many local problem solvers that can only see one character and its three adjacent characters, and only communicate at T or an F to other local problem solvers. Local problem solvers that can only see their nearby neighborhood model the individual cell in the biological system. This work simulates the distributed local problem solvers working together to solve this problem and then studies the relationship between the accuracy of the solution found with how much information is exchanged among the problem solvers. The results show that under some circumstances, the problem solving process can become unstable and never terminate. When the process terminates, results support the hypothesis that the information exchanged is maximized when the correct solution is found.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

List of Algorithms

CHAPTER 1

OVERVIEW

1.1 Complexity of problem solving in nature

Living organisms in nature have their own simplified way of solving problems which has been evolving since the origin of life on earth. Various forms of living organisms face various complex problems or situations like adverse changes in their environment, search for food and water, deciding a mate for reproduction, defense against predators, competitors, and invaders, etc.

No matter how big and complex the problem is they evolve to solve it. They solve these problems in concert, for example, division of labor or task or skills, with synchronized actions to build colonies in ants.

Another example of problem solving in nature is multicellular living organisms developing different organ systems to perform different function of body. They develop specialized group of cells which arrange themselves to form an organ which is specialized to do a particular task, like wise they form different organs, each one is specialized to do a particular task. These different organs together form a complete organ system (for example circulatory system, nervous system, digestive system, reproductive system, urinary system, etc).

1.2 Decision-making in Nature

Living organisms decide on whether its a normal situation or a problem situation based on the external stimuli. Deciding a particular situation is a problem situation is task of decision making in concert. Living organisms respond to these stimuli in concert and solve the problem. In multi-cellular organisms, millions of cells respond to these stimuli in concert and solve the problem efficiently. For example, yeast cells changing their food from glucose to ethanol and vice versa based on the availability of glucose and ethanol; immune

system cells deciding on which type and how many cells are required, and how to attach a particular pathogen, Mimosa Pudica ("touch-me-not") plant folding its leaf in-words when even a single leaflet is touched, it does so to protect itself from getting eaten by herbivores.

These organisms do not have centralized communication system, instead they have simple system to communicate to their neighbors (a very small set of their own forms) and still they are able to come up with a global consensus and respond to the situation very efficiently and appropriately, which is known as distributed decision making.

An example of decentralized or distributed decision making is human body's Immune system's CD4+ Lymphocytes, they are immune response controllers. They decide which actions to take during an invasion, promoting or inhibiting all other immune cells via cytokines. These cells activate cells that ingest dangerous material and also produce cytokines that induce the proliferation of B and T cells [7].

Another example of distributed decision making in nature is, cells organized themselves to get a particular form, structure and shape. Cells also decide on their coloration in distributed decision making to form a particular pattern which is reasonably important for their survival. A few examples of the patterning or organization occurring in the nature are:

Tigers, as shown in figure 1.1 a), have a pattern of dark vertical stripes on reddish-orange fur with lighter underparts. The pattern of stripes camouflage the animal in the dense forest, specially tall grasses. It helps them to conceal themselves amongst the dappled shadows and long grass of their environment as they stalk their prey as shown in figure 1.1 b).

Seashells: Patterns on some seashells, like the ones in Conus and Cymbiola genus, are generated by distributed decision making. The pigment cells reside in a narrow band along the shell's lip. Each cell secretes pigments according to the activating and inhibiting activity of its neighbor pigment cells, obeying a natural version of a mathematical rule. The cell band leaves the colored pattern on the shell as it grows slowly, which is shown in figure 1.2(a).

a)  b) 

Figure 1.1: Tigers are carnivores that have stripes on their hide or skin which help them to camouflage in the dense forest, specially in tall grass. a) A Bengal tiger Panthera tigris tigris in the wild in Ranthambhore National Park, Rajasthan, India (Courtesy of Bjrn Christian Trrissen). b) Tiger Camouflaged in tall grasses (Courtesy of Steve Winter, National Geographic)

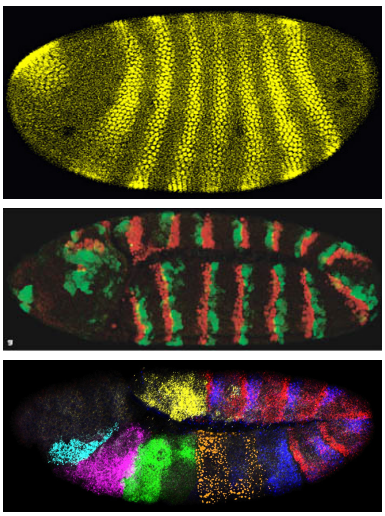
a) 

Figure 1.2: a) Conus textile, a seashell which has the patterns generated by distributed decision making of its cells (Courtesy of Richard Ling, Location:Cod Hole, Great Barrier Reef, Australia)

Drosophila: The pattern formation of the cells along the future head to tail (antero-posterior) axis of the fruit fly Drosophila melanogaster is exemplary to study how simple cells in the embryo develop themselves in to the complex patterns using the information in the genes. The development of Drosophila is particularly well studied, and it is representative of a major class of animals, the insects or insecta. Other multicellular organisms sometimes use similar mechanisms for axis formation. Figure 1.3

a) 

Figure 1.3: Male Drosophila melanogaster, also known as fruit fly, develops the patterning along its head to tail, (Courtesy Andr Karwath aka, Wikipedia)



(a) Expression of *hairy* (yellow) in the cellular blastoderm (Courtesy of Langeland, S. Paddock, and S. Carroll, HHMI)

(b) Expression of segment polarity genes, *wingless* (*wg*; green) and *engrailed* (*en*; red). Courtesy of C. Tomlin and J. D. Axelrod [21]

(c) Expression of seven *Hox* genes at the extended germ band stage. (Courtesy of Dave Kosman, UCSD)

Figure 1.4: Example of pattern formation in *Drosophila* embryos.

1.3 Models of Decision Making: Cellular Automata

Key structure of distributed decision making in living organisms:

They have very simple patterns of communication, for example, a few set of signals that progress throughout the network (of their own forms) to make decisions in unity. There is no central algorithm for making these decisions. It involves simple local agents who communicate locally which in turn communicate to their local agents and so on, finally they achieve global synchronization. Within a cell of component networks, for example

plants, have small sensor systems conceded along the leaf surface through which they sense environmental changes (stimulus) and communicate the information to the entire network using simple signals.

Cellular Automata is a discrete model studied in computability theory, mathematics, physics, complexity science, theoretical biology and microstructure modeling. It consists of a regular grid of cells. Each cell has finite number of states, such as "On" and "Off". The grid can be in any finite number of dimensions. Each cell knows about its neighborhood which is a set of cells. Most of the times the neighborhood includes the cell itself. A cells neighborhood is defined relative to itself(its position in the grid). Cellular automata can used to model any kind of distributed decision making in nature (including decision making our day to day life).

For example, the neighborhood of a cell might be defined as the set of cells a distance of 3 or less from the cell. An initial state is selected by assigning a state for each cell in the system. A new generation which reflects the new state of the cells in the system at the next time-step is created, according to some fixed rule that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. Decision making is an evolving property in time-steps.

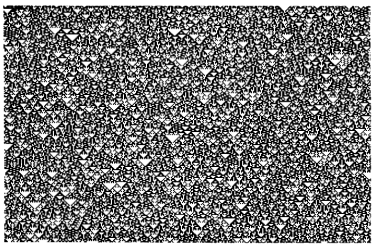
Models of Complexity:

Cellular Automata can be used to model complex systems. A cellular automata model has simple cells that have simple functions or mode of operations or states, which can generate complex systems. In nature there are millions of complex systems and structures that are generated from very simples forms of same type. These complex systems can be any thing from complex structures, complex organizations, complex functions, behaviors etc [23].

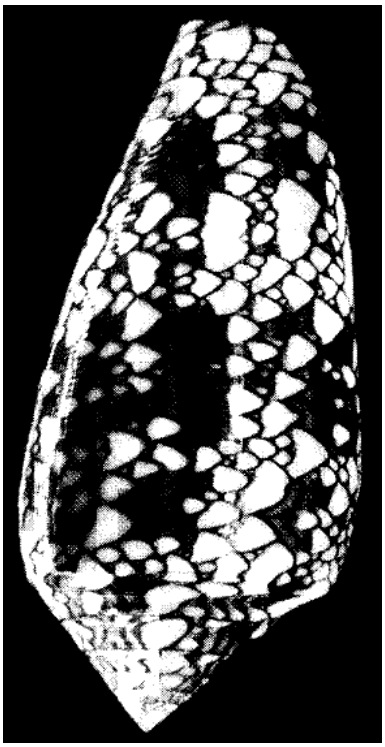
An example of Cellular Automata Models is Conway's game of life, he modeled the natural process like life, death and birth of living beings in the nature. This model needs only the initial conditions to start with, and it continues forever. The cell's states represent the life or death of the living beings and they follow certain rules to decide their fates. He

tried to represent the spontaneity of the life in the universe [2].

Seashell models of Cellular Automata are developed to synthesize realistic images and to gain better understanding of the mechanism of shell formation [12]. A sample pattern that is generated from simple initial random states On or Off of the cells with equal, independent probabilities is shown in figure 1.5 a). This pattern is reminiscent of the cone shell with a pigmentation pattern shown in figure 1.5 b).



(a) Evolution of the simple cellular automaton defined from a disordered initial state in which each site is taken to have value 0 or 1 with equal, independent probabilities. Evolution of the cellular automaton even from such a random initial state yields some simple structure (courtesy of Stephen Wolfram).



(b) A 'cone shell' with a pigmentation pattern reminiscent of the pattern generated by the cellular automaton of figure a), (Shell courtesy of P. Hut).

Figure 1.5: Seashell Model pattern formation

1.4 Majority Problem

Majority Problem is also known as the task of Density Classification. The task of density-classification [19] consists of correctly determining whether initial configurations (IC's) contain a majority of one of its discrete states or the other, by making the system converge to a configuration in which all the states are same as the state in majority at IC.

For example, suppose that the system components have two discrete states viz On and Off. Then the task of majority classification will consist of determining whether the IC's contain a majority of On's or Off's, by making the system converge, respectively, to an all On's state, or to a state of all Off's.

The density of an IC is dependent on the number of components or cells in the system. The local cells have limited information. Therefore the communication amongst the cells must be coordinated properly to classify majority of the IC's. As in [18] Density classification task is only applicable to system of odd number of cells because the outcome could be undecidable in the system with even number of cells. Also, devising CA rules that perform this task is not trivial, because cells in a CA system update their states based only on local neighborhood information.

In this report, we have referred IC's as problems in general. The problems are solved by CA rules to perform the task of density classification. The solution of a problem is determining the state in which majority of the cells are, which is determined by the DCT. The solution is correct when the DCT matches the state of the majority of the cells at the IC. The initial state of the system has random On and Off cells.

Each cell's contextual knowledge consist of its own state and the state of its nearest local six neighbors. Each column in the grid represents a time stamp or a time-step. At each time-step, each cell decides its state in the next time-step either On (black) or Off (white) based on the configuration of states of seven cells in its context (neighborhood) in that time-step, which is explained in Figure 1.6. This process continues till a fixed maximum number of time steps are completed or all the cells in the system are in the same state.

Some sample patterns generated from DCT are shown in Figure 1.7.

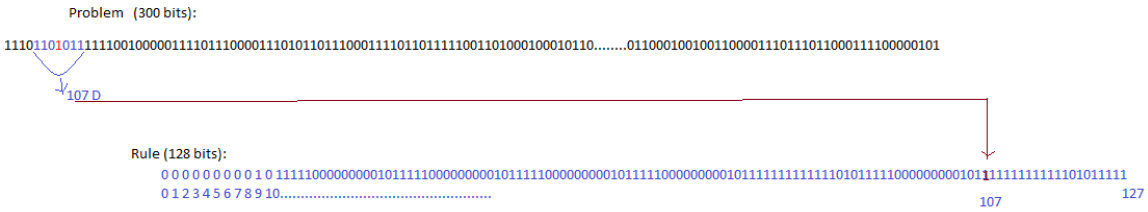


Figure 1.6: A cell (in red color) in the CA grid deciding its state in the next time-step. The cell in red color looks at the states of its six neighbors in Blue color and its own state, and encodes that configuration of seven states into a binary number and then to its integer equivalent, which is used as an index to determine its state in the next time-step from the rule's look up.

The CA model of distributed decision making is very simple to implement and understand and it demonstrates the important characteristics of the systems to be studied:

1. Simple local rules which govern the communication in a distributed network and
2. Global decision is an emergent property.

1.5 Information Flow

In the DCT, the local decision making agents (cells) determine the majority state in their neighborhood using CA rules and set their state in the next time-step to the majority state. In this way, at each time-step, this information about the majority state is conveyed to each agent from its neighborhood, which is known as *Information flow*.

The information flow in a system is high when there is less difference between the number of cells in majority state and the number of cells in minority state between two time steps. The measure of information flow can help us to determine the type of system under consideration.

This work considers the information flow within these natural decision making processes. Because the computation is local, information must be propagated among the simple decision making agents. This work applies an information theoretic measure to quantify the amount of information flowing through the system as it reaches its decision.

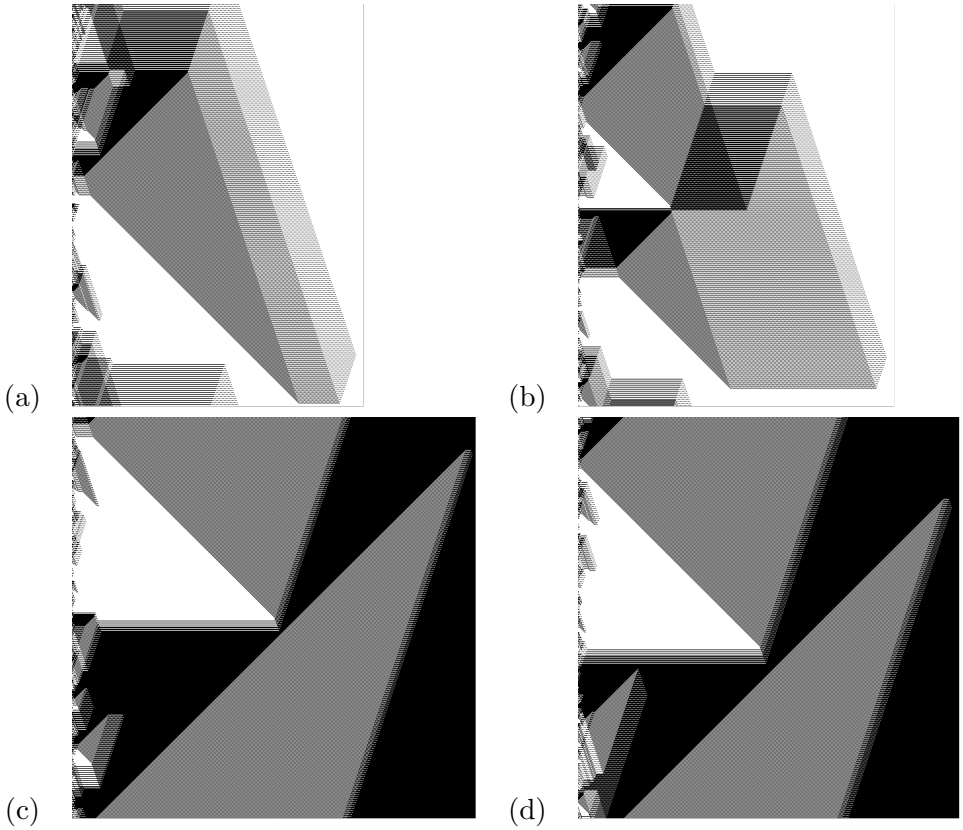


Figure 1.7: Examples of Correct and Incorrect termination of Result Patterns (all black or all white cells). a) and b) are the correct result patterns, that is the DCT has identified the majority correctly. c) and d) are incorrect result patterns, which means the DCT has failed to identify the majority.

1.6 Measuring Information in Distributed Systems

[10]. general introduction to measuring information in dynamical systems. Information flow of the dynamics during decision making process. A full review is beyond the scope of this paper, focus on Kolmogorov measures. Why useful, intrinsic information based on compression. Use normalized compression distance and set-complexity described in Section 2.2.

1.7 Hypothesis:

This work seeks to test the following hypotheses:

1. Set-complexity is a useful measure of information of distributed information processing in cellular automata.

2. Solving hard majority problems with cellular automata requires maximal information flow.

3. Therefore, if claims of 1 are true, we expect that set complexity is maximized when CA's solve hard problem correctly.

The remaining report is organized as follows. Chapter 2 describes the experiment's parameters, classification of result patterns and the use of set-complexity to measure the information flow in the system. Chapter 3 explains the five experiments to evaluate set-complexity, rule accuracy, rule robustness, set-complexity trajectory and Mann-Whitney test, we have done. Chapter 4 summarizes the key results of this study and concludes the findings from the results.

CHAPTER 2

EXPERIMENTAL SETUP

Parameters we have used to evaluate set-complexity, rule accuracy, rule robustness, set-complexity trajectory and Mann-Whitney experiments are:

Problem size, that is, the number of bits in the problem state = 599

Number of steps limit = 600

Standard 8 rules used are specified in the table 2.1

To represent the temporal-spatial state of the problem solving without loss of generality, we assign black(1) to be the majority in all problems. We used a two-dimensional grid of fixed number of rows (of length = 600, time-steps) and columns (of height 599, problem state). Each column represents the state of the cells or components of the system at a particular time-step. The state of cells in the first column represents the initial random number of 1's and 0's, problem (P_o) and the state of cells in the last column represents the result of the DCT.

For our experiments we have selected eight well known and high performing rules, which are shown in table 2.1. Each rule is a particular binary sequence of 128 On (1) and Off (0) bits representing each output of rule for all 128 possible inputs in binary counting order. These rules are used in DCT of ICs (P_o). The problems are classified based on their difficulty levels, determined by the difference between the number of cells with value 1 and the number of cells with value 0. The hardest problem will have difficulty level 1 and easiest will be n/2. The difficulty level of a problem is inversely proportional to the density of the 1 valued cells.

$$\rho = \frac{(n/2) - p}{n} \tag{2.1}$$

Table 2.1: Rules Used In Experiments

Rule#	Name	Hexadecimal	Reference	Accuracy
1	GLK	005F005F005F005F005FFF5F005FFF5F	Gacs et al. (1978) [13]	81.60%
2	Davis	002F035F001FCF1F002FFC5F001FFF1F	Davis (1991) [9]	81.80%
3	Das	000F730F001FFF0F000FFF0F001FFF1F	Das Rule [8]	82.3%
4	K96	00550055005500555F55FF5F5F55FF5F	K96 [15]	82.3%
5	Coe1	011430D7110F395705B4FF17F13DF957	Coe1 (1998) [22]	85.1%
6	BOO1	145500CC0F14021F1715FFCf0F17FF1F	Bortot et al. (2004) [20]	86.16
7	ABK	0500550505005505555FF55FF55FF55FF	Andre, Bennet, Koza [1]	82.4%
8	Unknown	0506158707041557647705017DFFB77F	Unknown	

where, p is the difficulty level of the P_o and n is the number cell in the P_o. Therefore, for the hardest problem, $\rho = \frac{(599/2)-1}{599}$, that is $\rho = 0.498$ and for the easiest problem, $\rho = 0$.

For easy problems the rules take less time-steps as compared to hard ones. For hard problems the rule some times can never find solution and also does not appear to terminate. That is, it never appears to get all bits in the same state, it takes large number of time steps. For computational limitations, we set the time-step limit to 600.

For our experiments we have determined the difficulty level of the problems on a scale 1 to 30. Problem of difficulty Level 1 is hardest one and the problem of difficulty level 30 can be solved almost perfectly.

We determined this by studying, how accuracy of a rule is affected by the difficulty level of the problem (P_o) and is discussed in detail in section 3.2. Also we studied rule sensitivity to mutations. This is discussed in detail in the in section 3.3.

2.1 Outcome Classification

A space/time pattern is the graphical representation of the state of all the cells in the system at each time-step. The '1' state is represented as a black colored cell and the '0' state is represented as a white colored cell in the pattern. A pattern corresponds to the different states through which the cells in the system go through from initial time-step to the final time-step in the DCT.

There are numerous patterns generated as a result of DCT. We classified the patterns

based on the outcome in solving the DCT. If the DCT correctly classifies the majority of a P_o, then the pattern is classified as correct-ordered. Examples of correct-ordered pattern is shown in the figure 2.1 (a) and 2.2 (a). If the DCT terminates with an incorrect majority classification then the corresponding pattern is classified as incorrect. Example of incorrect pattern is shown in figure 2.1 (d) and 2.2 (d). We have classified the patterns into two major types:

1. Correct-Ordered: A pattern is classified as Correct-Ordered if the DCT(Density Classification Task) terminates correctly, which means it has determined the correct majority in the P_o. Example, figure 2.1 (a) and 2.2 (a).

2. Incorrect: A pattern is classified as Incorrect if:

   (a) The DCT does not terminates (fails to determine the majority) in certain upper limit of time steps, or

   (b) The DCT terminates incorrectly (determines incorrect majority).

   we have classified the incorrect patterns into three sub types:

   Incorrect-Same : If the DCT is terminated incorrectly with all 0's, which means that the DCT has determined incorrect majority. Example, figure 2.1 (d) and 2.2 (d).

   Incorrect-Ordered : If the DCT does not terminate in certain upper limit of time-steps and it is repeating same configuration of states of the cells in the system after fixed number of time steps, then the result is classified as Incorrect-Ordered.

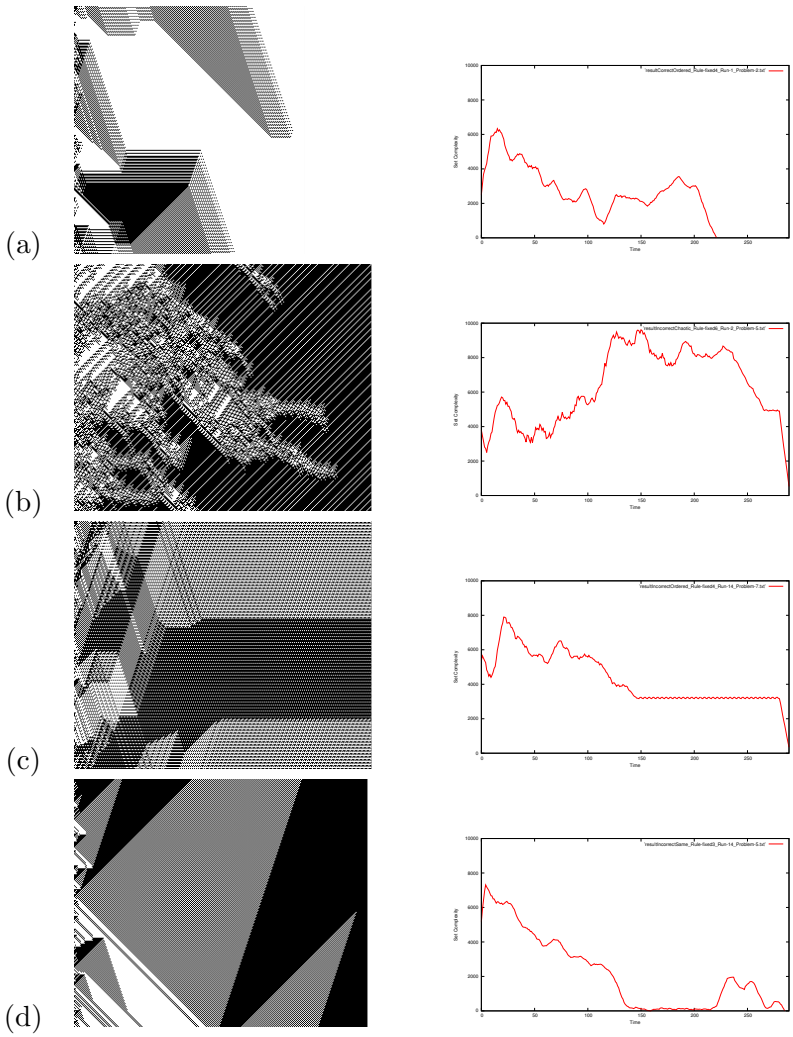


Figure 2.1: Examples of pattern types. Each row in the above figure is numbered alphabetically, represents a type of pattern. (a) Correct-Ordered , (b) Incorrect-Chaotic, (c) Incorrect-Ordered, (d) Incorrect-Same

Example, figure 2.1 (c) and 2.2 (c).

Incorrect-Chaotic : If the DCT does not terminate in certain upper limit of time-steps and it is generating a non-repeating configuration of states of the cells, then the result is classified as Incorrect-Chaotic. Example, figure 2.1 (b) and 2.2 (b).

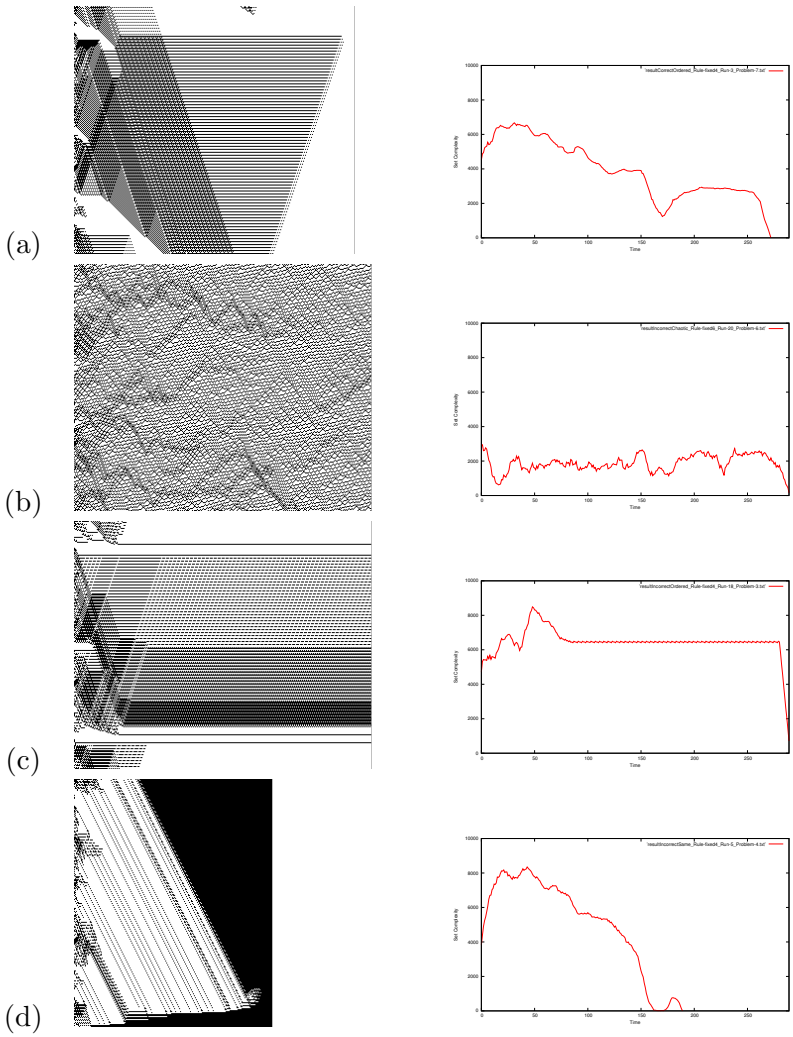


Figure 2.2: More examples of pattern types. Each row in the above figure is numbered alphabetically, represents a type of pattern. (a) Correct-Ordered , (b) Incorrect-Chaotic, (c) Incorrect-Ordered, (d) Incorrect-Same

2.2 Information measurement

To study the information flow in the system we chose set-complexity, which is based on algorithmic complexity, introduced by Kolomogorov [16]. Set complexity [14] has been used to measure the information content of regulatory networks, their temporal dynamics and the spatial patterns produced [11]. By measuring information content, set complexity can distinguish between critical systems that encode maximal information, and ordered and chaotic systems that encode low information. Set complexity (referred to as Ψ) applies

Kolmogorov's intrinsic complexity to quantify contextual information in a set of objects by discounting pairs of strings that are randomly related or redundant. Set complexity is independent of any specific application, so long as each object in the set can be encoded as a string.

In this application, set-complexity is used to compute the information content of a sequence of spatial states $P_i...P_{i+w}$, where, w is the window size [4].

2.2.1 Definition

The Kolmogorov complexity of two strings is the length of the shortest algorithm that can transform one string to the other. Exact computation is undecidable, but minimum algorithm length can be approximated by the normalized compression distance (*NCD*) [5,6]. *NCD* is defined below, where x and y are strings, xy is the concatenation of x and y and $C(x)$ is the compression size of x:

$$NCD(x,y) = \frac{C(xy) - \min(C(x), C(y))}{\max(C(x), C(y))}, 0.0 \leq NCD(x,y) \leq 1.0 \tag{2.2}$$

NCD is a measure of the similarity of the two strings and estimates the normalized size of a program that can transform x to y. [4]. Consider the following cases, where dissimilar strings could be a random string paired with a string of a single repeated character:

Similar strings: $x \simeq y$**,** $C(xy) \simeq C(x) \simeq C(y)$ then $NCD(x,y) \simeq 0$

Random strings: $x \neq y$**,** $C(xy) \simeq C(x) + C(y)$ then $NCD(x,y) \simeq 1$

Dissimilar strings: $x \neq y$**,** $C(xy) \simeq C(x), C(y) = 0$ then $NCD(x,y) \simeq 1$

To ensure accurate measurement of compression length the block size of the compressor must be greater than the string length. Here we used the bzip2 compression algorithm with a block size of 900 Kbytes [3].

Then set complexity of a set of n strings $S = \{x_1, \ldots, x_n\}$ is defined:

$$\Psi(S) = \frac{1}{n(n-1)} \sum_{x_j \in S} C(x_j) \sum_{x_j \neq x_k} d_{jk}(1 - d_{jk}) \tag{2.3}$$

where $d_{ij} = NCD(x_i, x_j)$. The distance d_{ij} measures the information between the two strings and is maximized when $NCD(x_i, x_j) = 0.5$ and minimized when $NCD(x_i, x_j) = 0.0$ or $NCD(x_i, x_j) = 1.0$. When strings in the set are similar, $\Psi(S) \simeq 0$ indicating the set belongs to the ordered domain and contains little information. Chaotic systems generate strings that appear random and then $\Psi(S)$ is minimized, but not zero because of the $C(x_j)$ multiplicative term. In [14] it is shown that $\Psi(S)$ is maximized when the set of strings describe a critical system.

2.2.2 Encoding of the states as strings

In this section we will discuss about encoding of the dynamics in the states of the cells at each time-step into strings which are used to calculate the Normalized Compression Distance(NCD) which in turn, is used to calculate Set-Complexities of the pattern at each time-step.

A cell's state which is either true (On) or false (Off) in the grid, is encoded as an integer 1 or 0 respectively. In this way, for each column in the pattern, a string of 0's and 1's is encoded. Each string directly corresponds to the state of the cells in the system at a particular time step. By comparing the strings corresponding to w consecutive time steps, where, w is the window size, we can determine the information flow content of the CA during step t to $t + w$. By graphing the set-complexity over the whole trajectory, the flow of information during problem solving can be visualized. Information flow can be used to determine the type of system viz. critical, ordered or chaotic.

Set Complexity algorithm that utilizes caching and bzip:

The encoded strings are compressed using Bzip2 compression algorithm and the compression length for each column (time-step) is stored. The pseudo code to calculate NCD

and then set-complexity using the NCD values of NCD calculations is explained in Algorithm 1. The *calculateSetComplexity* procedure call used in this algorithm is explained in Algorithm 2 and the *calculateNCD* procedure call is explained in Algorithm 3.

By caching compression values, the compressor is only called n times (number of time steps) for each individual state. By caching string concatenations the compressor is only called $n * w$ times, rather than $n * w^2$ times, where, w is the window size.

Algorithm 1 Pseudo-code to calculate set-complexity of the contextual information in the system at each time-step

input:
$problemSize \leftarrow 599$
$timeSteps \leftarrow 600$
$windowSize \leftarrow 10$
$pattern[problemSize][timeSteps]$
initialize:

$columnString$ ▷ array of strings
$colCompress$ ▷ array of longs
$pairCompress[timeSteps][windowSize]$ ▷ 2D array of longs
$NCD[problemSize][timeSteps]$ ▷ 2D array of doubles
$stComplexity[timeSteps]$ ▷ array of doubles

for $step = 0 \rightarrow (timeSteps - 1)$ **do**
  for $rowNumber = 0 \rightarrow (problemSize - 1)$ **do**
convert boolean value at $pattern[rowNumber][step]$ to int, then to string and append it to the string at $columnString[step]$
  end for
end for

for $step = 0 \rightarrow (timeSteps - 1)$ **do**
compress the string at $columnString[step]$ and store its compression length at $columnCompress[step]$
end for

for $step = 0 \rightarrow (timeSteps - windowSize)$ **do**
  for $windowNumber = 1 \rightarrow (windowSize - 1)$ **do**
append the strings at $columnString[step]$ and $columnString[step + windowNumber]$
compress the appended string and store its compression length at $pairCompress[step][windowNumber]$
  end for
end for

for $step = 0 \rightarrow (timeSteps - windowSize)$ **do**
  for $windowNumber = 1 \rightarrow (windowSize - 1)$ **do**
$pairSize \leftarrow pairCompress[step][windoNumber]$
$oneSize \leftarrow columnCompress[step]$
$twoSize \leftarrow columnCompress[step + windowNumber]$
$NCD \leftarrow$ **return**CALCULATENCD$(pairSize, oneSize, twoSize)$

  end for
end for
$setComplexity \leftarrow$ **return**CALCULATESETCOMPLEXITY$(NCD, columnCompress, windowSize)$

Algorithm 2 procedure to calculate set-complexity

procedure CALCULATESETCOMPLEXITY($NCD, columnCompress, windowSize$)
for $timestep = 0 \rightarrow (timeSteps - windowSize)$ **do**
$doubleSC \leftarrow 0.0$;
for $windowJ = 1 \rightarrow (windowSize - 1)$ **do**
$doubleinformation \leftarrow 0.0$;
for $windowK = 1 \rightarrow (windowSize - 1)$ **do**
if $windowK \neq windowJ$ **then**
$information \leftarrow information + NCD[timestep + windowJ][windowK] * (1.0 - NCD[timestep + windowJ][windowK])$;
end if
end for
$SC \leftarrow SC + columnCompress[timestep + windowJ] * information$;
end for
$setComplexity[timestep] \leftarrow (1.0/(double)(windowSize * (windowSize - 1))) * SC$;
end for
return $setComplexity$
end procedure

Algorithm 3 Procedure to calculate Normalized Compression Distance (NCD)

input:
$pairSize$ ▷ compression length of the two appended strings
$oneSize$ ▷ compression length of first string
$twoSize$ ▷ compression length of second string

procedure CALCULATENCD($pairSize, oneSize, twoSize$)
$ncd \leftarrow (pairSize - Math.min(oneSize, twoSize))/(double)Math.max(oneSize, twoSize)$;
return $(Math.max(Math.min(ncd, 1.0), 0.0))$;
end procedure

CHAPTER 3

EXPERIMENTAL STUDIES

To support our hypothesis described in section 1.7, we did five experiments:

Evaluate set-complexity: to verify the correctness of the set-complexity calculations we did in the set-complexity trajectory experiment. This experiment is discussed in detail in section 3.1.

Rule accuracy: To verify that the observed accuracy values of the rules used in our experiments are same as their known values. Therefore we can verify that DCT is done correctly in our experiments which is discussed in detail in section 3.2. Also to determine the hardest problems which lowers the accuracy of the rules.

Rule robustness: To validate that the accuracy of the rules used in our experiments have locally maximal accuracy. Section 3.3.

Set-complexity trajectory: To study the trends and to determine if there is any difference in the set-complexity curves of different class of outcomes generated from the DCT of hard problems. Section 3.4.

Mann-Whitney test: To determine, if there is significant difference the set-complexity curves of different class of outcomes and complex patterns corresponding to correct results show maximal information flow. Section 3.5.

3.1 Experiment One: Evaluating Set Complexity

We did experiments to verify that set-complexity can distinguish between random and ordered pattern trajectories, described in the Equation 2.3. We have created synthetic patterns of six possible pair wise combinations of three basic encoded string types viz. simple, random and complex which are described and named conventionally as follows:

Regular: Pattern formed by a single string repeated in all the time-steps. This string consist of a repeated substring which has fixed number of bits (0's and 1's).

Simple: Pattern formed by strings which consist of a repeated substring. This substring has fixed number of bits(0s and 1s). These fixed number of bits are chosen randomly for each time-step. Example, figure 3.1 (a1).

Random: Pattern formed by random strings of 0s and 1s, having fixed number of 1's. These fixed number of 1's are same for each time-step. Example, figure 3.1 (b1).

Complex: Pattern formed by strings that have fixed number of bits(0s and 1s) which are repeated after a fixed number of random bits. These fixed number of bits are chosen randomly for each time-step. Example, figure 3.1 (c1).

Simple-Complex: Pattern formed by repeatedly alternating simple and complex strings. Example, figure 3.1 (d1)

Simple-Random: Pattern formed by repeatedly alternating simple and random strings. Example, figure 3.1 (e1).

Random-Complex: Pattern formed by repeatedly alternating random and complex strings. Example, figure 3.1 (f1).

Simple-Random-Complex: Pattern formed by repeatedly alternating simple, random and complex strings. Example, figure 3.1 (g1).

For each synthetic pattern generated, the set complexity is calculated and plotted for each time step. Example graphs and the corresponding patterns are shown in the figure 3.1.

3.1.1 Results

Results show that the set-complexity is near zero for regular patterns, high for simple patterns. Set-complexity of complex patterns is higher than that of simple patterns. Other patterns also have high set-complexities. However, against the expectations, complexity of random patterns appears to have high value.

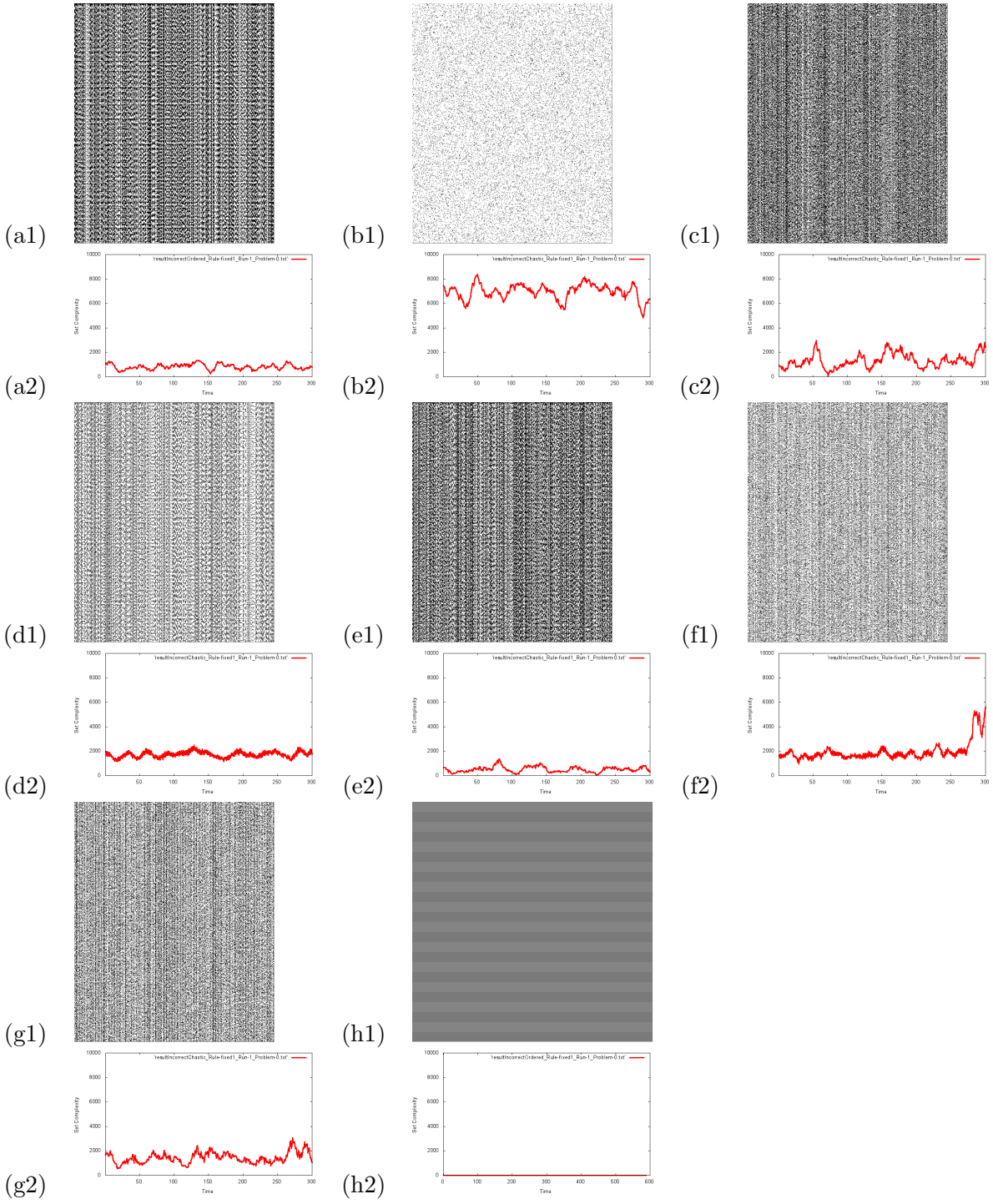


Figure 3.1: Synthetic Patterns and their Set-Complexity curves, with set-complexity(vertical axis) at each time-step(horizontal axis). (a1) Simple Pattern (a2) set-complexity curve Simple Pattern, (b1) Random Pattern, (b2) set-complexity curve, Random Pattern, (c1) Complex Pattern, (c2) set-complexity of Complex Pattern, (d1) Simple-Random Pattern, (d2) set-complexity of Simple-Random Pattern, (e1) Simple-Complex Pattern, (e2) set-complexity of Simple-Complex Pattern,(f1) Random-Complex Pattern, (f2) set-complexity of Random-Complex Pattern, (g1) Simple-Random-Complex Pattern, (g2) set-complexity of Simple-Random-Complex Pattern, (h1) Regular Pattern, (h2) set-complexity of Regular Pattern

3.2 Experiment Two: Rule Accuracy

In the two dimensional cellular automata, not all the majority problems (IC) are solved or classified correctly by any rule (existing so far) in the DCT.

For example, the hardest problem of length 599 has 300 'On' states ($\rho = 0.49$) and 299 Off states(0's or white cells) randomly arranged in the initial configuration, and the easiest problem will have all the same states (either On or Off, $\rho = 1$) in the IC.

To make sure the results of our experiments are correct, specially the problems are correctly solved by the eight rules(given in 2.1) and the results are classified correctly, we calculated the accuracy of all the rules we have used in our experiments. The accuracy of a rule is calculated to determine how well it solves a set of random problems and how its accuracy is affected by varying the difficulty level of the problem.

Measure of accuracy will help to confirm the results we have got in our experiments satisfy previous evaluation of rules. For example, if we are solving the problems of difficulty level 1 using a particular rule and its known accuracy is 90%, which means out of hundred random majority problems, a rule is expected to correctly classify approximately 90 of them. Therefore we can always check if the accuracy of rules is in the expected range, otherwise something has gone wrong in the experiments.

Accuracy of a rule for fixed number of on-bits is the percent of times a rule succeeded in the DCT. At each iteration a rule tries to determine the majority in a random majority problem (P_o). For the number of 1 bits in random P_o, the percent of times the rule succeeds to classify the density is recorded. The fixed number of 1 bits in the problem are varied from 0 till the problem length (row-length = 599). For each number of 1 bits the accuracy is calculated. The pseudo-code of the algorithm used to determine the accuracy of rules is shown in Algorithm 4.

3.2.1 Results

The plot of the accuracy against the number of on-bits (horizontal axis) in the problem gives an idea of the over all accuracy in percent (vertical axis) of the rule for number of on-bits. The graphs for accuracy of different rules are shown in Figure 3.2.

Algorithm 4 Pseudo-code to calculate accuracy of rules based on the number of On-Bits (difficulty level of the problem)

for $numberOfRules = 1 \rightarrow 8$ **do**

  for $initialNumberOfOnes = 0 \rightarrow 599$ **do**

    for $timesPerInitialNumberOfOnes = 1 \rightarrow 100$ **do**

      create a random problem with exact $initialNumberOfOnes$ (bits true).
      solve the problem
       if $resultisasexpected$ **then**
          $countOfCorrectResults \leftarrow countOfCorrectResults + 1$
       end if
       $countOfAttempts \leftarrow countOfAttempts + 1$

    end for
    $accuracy \leftarrow countOfCorrectResults/countOfAttempts * 100$
   store the $accuracy$ for the $initialNumberOfOnes$

  end for

end for

The results confirm that problems with density 50% are the hardest and most interestingly, problems with density greater than 60% or less than 40% are easily solved by the rules.

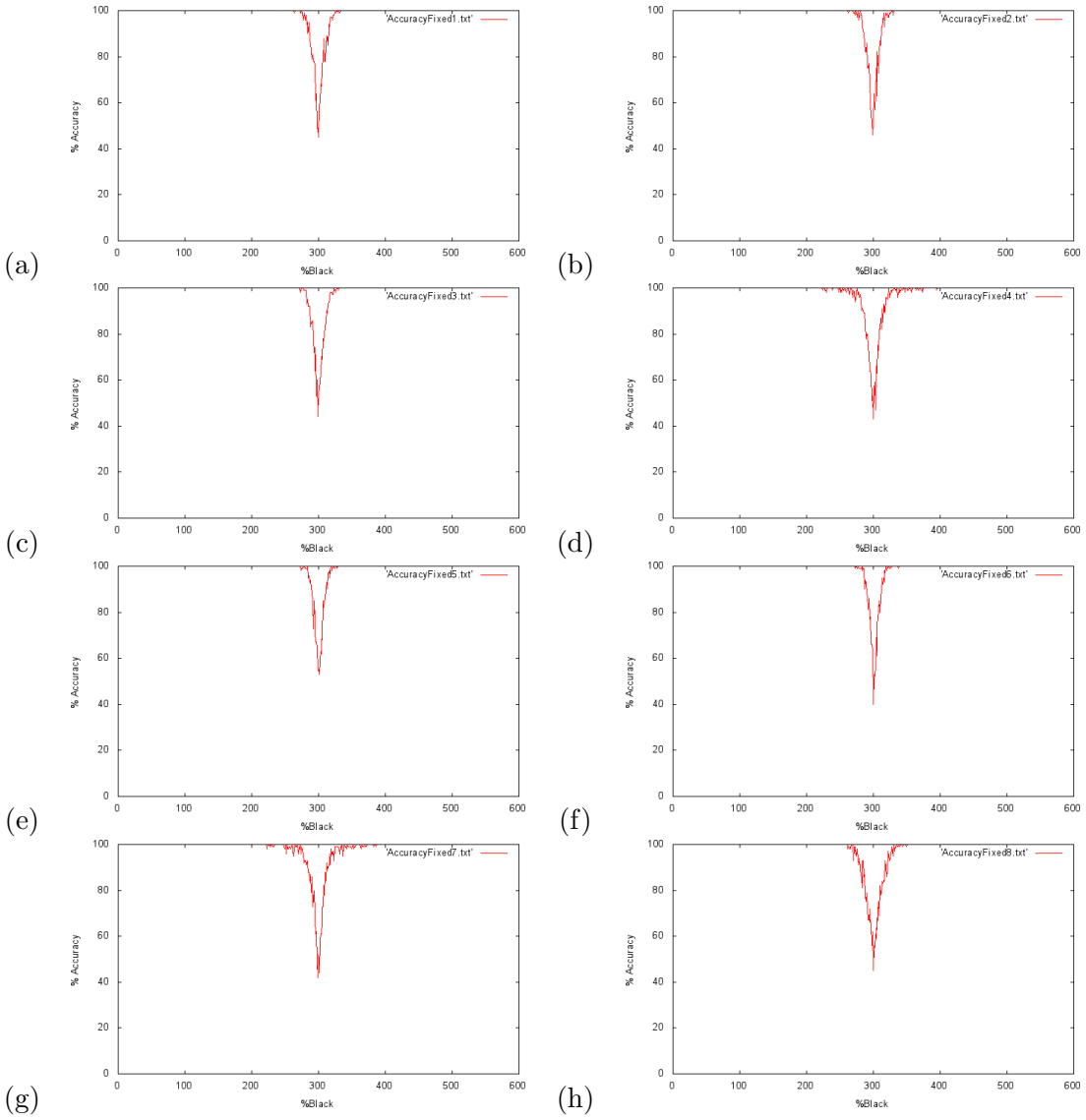


Figure 3.2: Graphs of Rule Accuracy for Number of On-bits. Each graph is the plot of accuracy(vertical axis) of the rule against number of On-Bits(horizontal axis) in the problem (IC). (a) Fixed1, (b) Fixed2, (c) Fixed3, (d) Fixed4, (e) Fixed5, (f) Fixed6, (g) Fixed7, (h) Fixed8, from the graphs its very clear that accuracy of a rule increases with increase int the density of the majority bits (state)

3.3 Experiment Three: Rule Robustness

The accuracy of the rules used in CA, can drop or increase after mutations to the rules output. A high robustness rule is one where its accuracy stays considerably high after mutations. The sensitivity of the rule is inversely proportional to its robustness. In this

experiment, we tested the sensitivity of the rules to mutations. A single mutation to a rule is a single bit flip of one of its output bits in the 128 bit word given in the table 2.1. Based on the sensitivity of a rule, we can validate that the rule's accuracy has locally maximal accuracy.

3.3.1 Results

The accuracy for number of mutation Bits of a rule is calculated using the accuracy for Number of On-Bits for the same rule as described in the previous section 3.2. A rule is mutated for fixed number of bits (0 to 7) at random locations, then accuracy for Number of 1 bits in the majority is calculated for the mutated rule. These calculated accuracies are plotted against number of on-bits in the problem. The resultant graph will be accuracy plot for fixed number of mutation bits for a rule. Similarly, accuracies can be calculated and plotted with varying number of mutation bits from 0 to 7. Hence, for each rule, eight different curves are generated. The graphs for accuracies for different number of mutation bits for different rules are shown in Figure 3.3.

The results show that each rule has locally maximal accuracy. Since accuracy falls off as bits are mutated. Interestingly, the graphs of accuracy under mutations are not symmetric. This happens because some mutations force the rule to always converge to 1 or 0.

Algorithm 5 Pseudo code to calculate accuracy of rules based on the number of mutated bits and number of on Bits.

for $numberOfRules = 1 \rightarrow 8$ **do**

for $numberOfMutationBits = 0 \rightarrow 7$ **do**

mutate the rule with $numberOfMutationBits$

for $numberOfTimesPerMutation = 1 \rightarrow 100$ **do**

for $initialNumberOfOnes = 0 \rightarrow rowLength$ **do**

for $timesPerInitialNumberOfOnes = 1 \rightarrow 100$ **do**

create a random problem with exact $initialNumberOfOnes$ (bits true).

solve the problem

if $resultisasexpected$ **then**

$countOfCorrectResults \leftarrow countOfCorrectResults + 1$

end if

$countOfAttempts \leftarrow countOfAttempts + 1$

end for

$accuracy \leftarrow countOfCorrectResults/countOfAttempts * 100$

$averageAccuaracyList[initialNumberOfOnes][numberOfTimesPerMutation] \leftarrow accuracy$

end for

end for

calculate and store $averageAccuracy$ for 100 runs of the mutated rule from $averageAccuaracyList[][]$ for each $initialNumberOfOnes$.

end for

end for

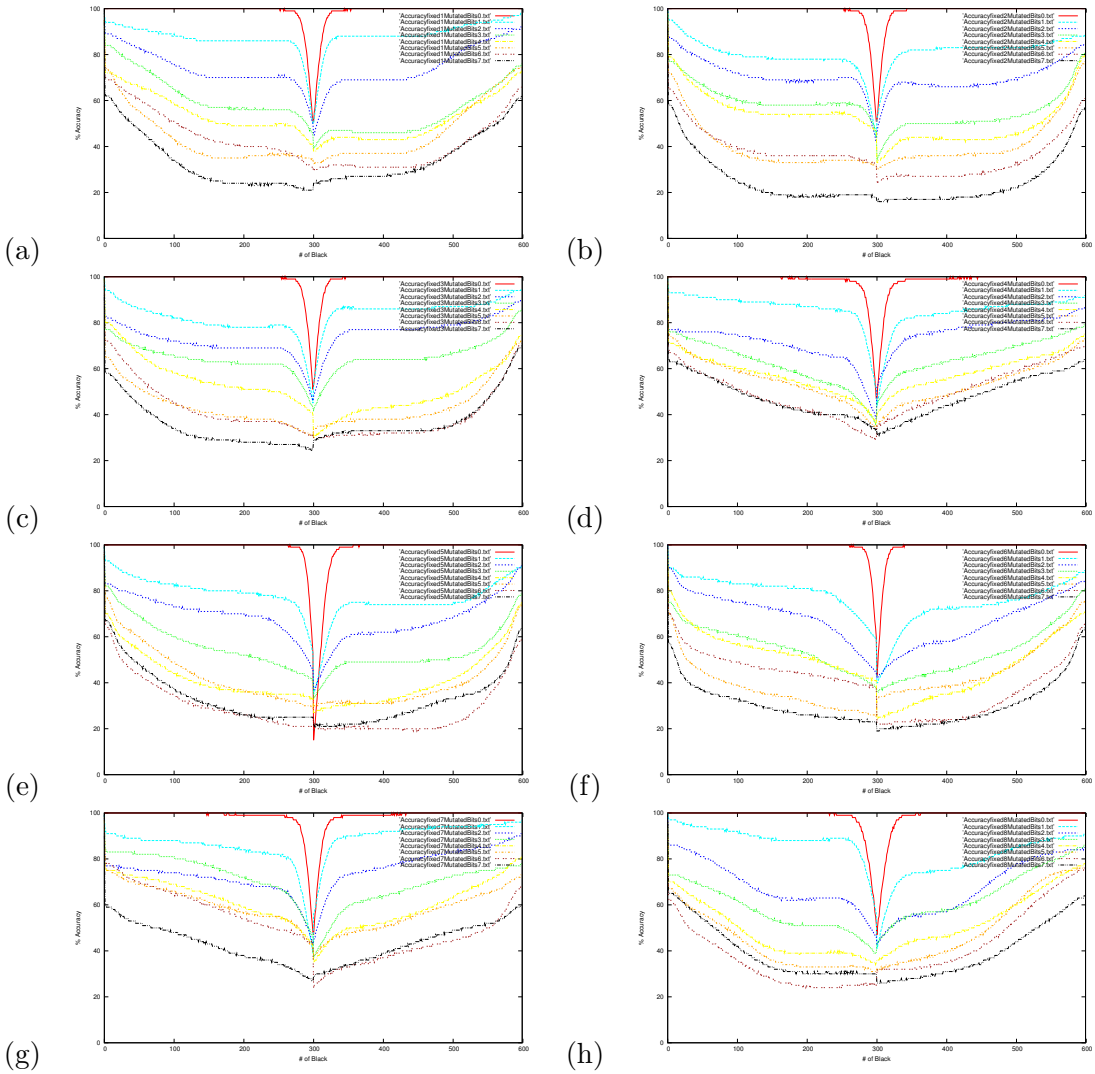


Figure 3.3: Rule Accuracy for Number of Mutation Bits, x-axis is number of 1's in the problem (of size 599), y-axis is accuracy in percent. Each graph corresponds to accuracy plot of 8 mutations (0 to 7 bit) of a rule named (a) Fixed1, (b) Fixed2, (c) Fixed3, (d) Fixed4, (e) Fixed5, (f) Fixed6, (g) Fixed7, (h) Fixed8, from the graphs its very clear as the number of mutation bits increase the accuracy drops.

3.4 Experiment Four: Set Complexity Trajectory

We calculated the set-complexity trajectory using real patterns that emerge by solving the different levels of problem difficulty for all eight of the famous rules shown in table 2.1, over a large set of random problems.

Pseudo-code used in our experiments for calculating multiple set-complexity trajectories and the median-complexity at each time-step is shown in Algorithm 6. For the DCT of a majority problem by a particular rule, set-complexity at each time-step for the window of size w, patterns is calculated. Plotting the graph of these complexities against time-steps provides a clear picture of set complexity at each time-step, precisely the measure of information flow at each time step.

Algorithm 6 Pseudo code to calculate the set-complexity

for $numberOfRules = 1 \rightarrow 8$ **do**

$resultArray[timeSteps][list < double >]$

for $numberOfTimes = 1 \rightarrow 1000$ **do**

create a random problem
solve the problem calculate the set-complexity and store it in $trajectory[]$
classify the result $Pattern[][]$ in to one of the four classes viz correct, incorrect-chaotic, incorrect-same, incorrect-ordered

for all $timeStep$ such that $0 \leq timeStep < trajectory.length$ **do**

for the current value of $timeStep$, add $complexityMeasure$ value in $trajectory[]$ to $resultClassList[]$
calculate median of $complexityMeasure$ for the current value of $timeStep$ in $resultClassList[]$ and store in the $medianResults[]$ for the corresponding $timeStep$.

end for
end for

end for

We can compare the set-complexities at each time-step of the different class of patterns generated by a particular rule. The complexity graph for each rule (each row in the figure

represents a rule) for each class of results (each column in the figure represents class of result) is shown in the Figure 3.4 and 3.5. These graphs are plotted for results-patterns generated by solving a particular class of problems, difficulty level-1, in this case.

Trajectories are classified based on the outcome (defined in section 2.1). For these graphs a log time scale is used to focus attention on the initial phases of the problem solving.

3.4.1 Results

In all trajectories, the set-complexity initially rises then either trends lower for ordered outcomes, or begins wide irregular oscillations for chaotic dynamics.

The initial rise in set-complexity is due to the organization of the original random problem state by the application of the rule, the fall from the peak is due to the introduction of the regions of regularity into the state as the system moves towards an answer. Chaotic systems replicate the random results of the synthetic data in section 3.1.

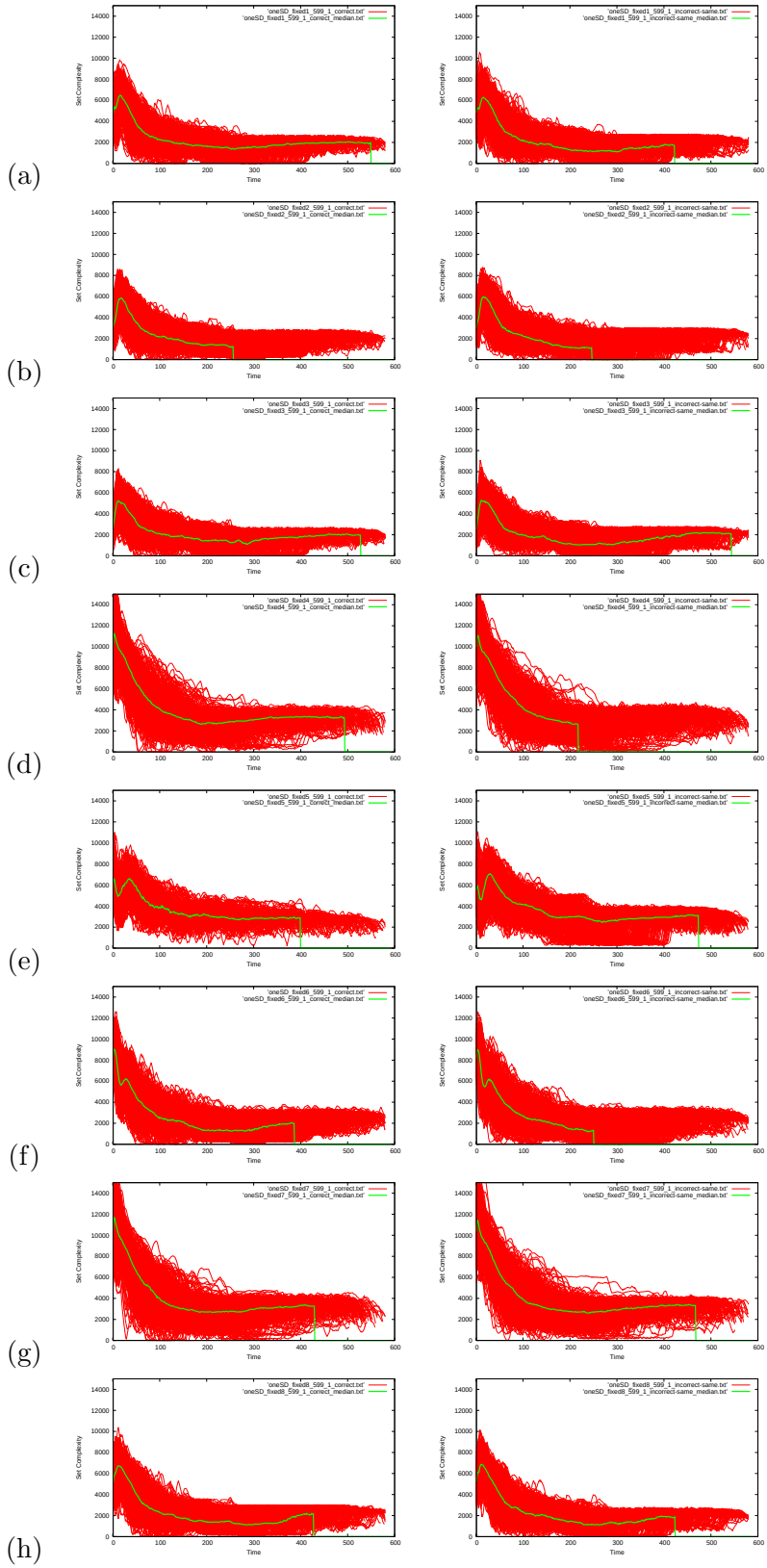
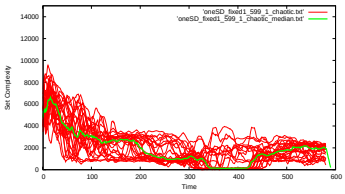


Figure 3.4: Rule Complexities. The columns represent the class of patterns. First column has Correct-Ordered, second column has Incorrect-same patterns and the rows represent the rules, (a) Fixed1, (b) Fixed2, (c) Fixed3, (d) Fixed4, (e) Fixed5, (f) Fixed6, (g) Fixed7, (h) Fixed8

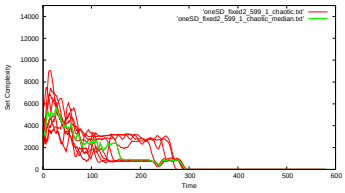
(a)

No Results are classified in this class for this Rule



(b)

No Results are classified in this class for this Rule

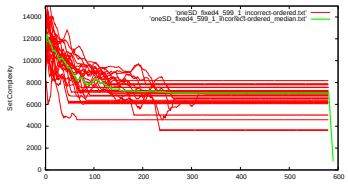


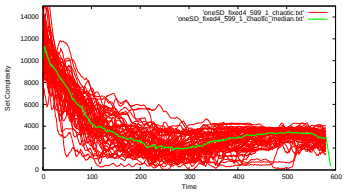
(c)

No Results are classified in this class for this Rule

No Results are classified in this class for this Rule

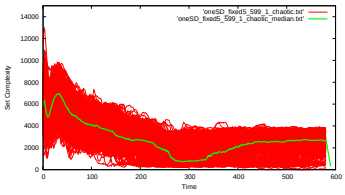
(d)





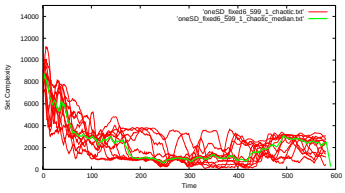
(e)

No Results are classified in this class for this Rule

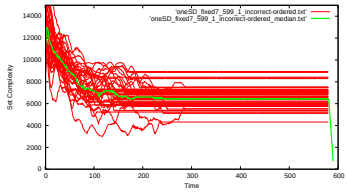


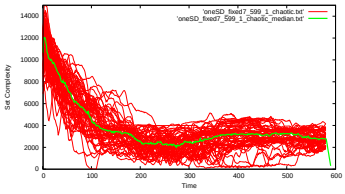
(f)

No Results are classified in this class for this Rule



(g)





(h)

No Results are classified in this class for this Rule

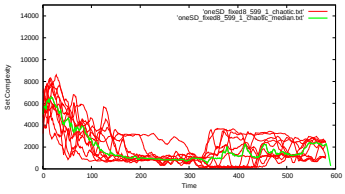


Figure 3.5: Rule Complexities. The columns represent the class of pattern. First column Incorrect-Ordered, second column Incorrect-Chaotic patterns and the rows represent the rules, (a) Fixed1, (b) Fixed2, (c) Fixed3, (d) Fixed4, (e) Fixed5, (f) Fixed6, (g) Fixed7, (h) Fixed8

3.5 Experiment Five: Mann-Whitney

The Mann-Whitney statistical test [17] is conducted to determine, if there are significant differences between the medians of all pairs of outcome classes at any time step. This test is selected because it is parameter free and works without distributional assumptions. The pseudo code used for conducting Mann-Whitney Test is shown in Algorithm 7. The set-complexities calculated at each time-step of the pattern produced as a result of DCT are used in the Mann-Whitney test.

3.5.1 Results

The p-values obtained for each time-step by the Mann-Whitney test can be plotted as a function of time-step. Thus graphs (Figure 3.7) are obtained that show at what time-steps different between the classes are significant. There was no significant difference observed between any of the classes of results. The reason might, since there are almost 3600 tests performed, that calculation came from 6 pair-wise comparisons across 600 time steps. There are likely errors in the handling of ties in the calculations as well as some unknown errors causing the p-values to be bounded below at 0.5 making it impossible to determine significance.

Algorithm 7 Pseudo code to determine measure of difference amongst the set-complexities of a given pair of results for a particular time-step. These pair of results are produced by the same rule. The inputs are two arrays of set-complexity values at a particular time-step of two set of results, it will return a p-value

initilization

$set1[]$ is a sorted the list of $setComplexties$ for a $particularTimeStep$ for a particular Rule for a type Of Result.

$set2[]$ is a sorted list of $setComplexities$ for a $particularTimeStep$ for a particular Rule for a type Of Result.

$resultSet1$ and $resultSet2$ are ranked combined

now the $set1RankSum$ and $set2RankSum$ for $set1$ and $set2$ respectively are calculated.

$UResultSet1$ and $U1ResultSet2$ is calculated using following equation

$$UResultSet1 \leftarrow set1RankSum - \left(\frac{set1Size*(set1Size+1)}{2}\right)$$

$$minU \leftarrow min(UResultSet1, UResultSet2)$$

$$mean \leftarrow (set1Size * set2Size)/2$$
$$standardDeviation \leftarrow \sqrt{set1Size * set2Size(set1Size + set2Size + 1)/12}$$

$$zValue \leftarrow (minU - mean)/standardDeviation$$

pValue is calculated from zValue

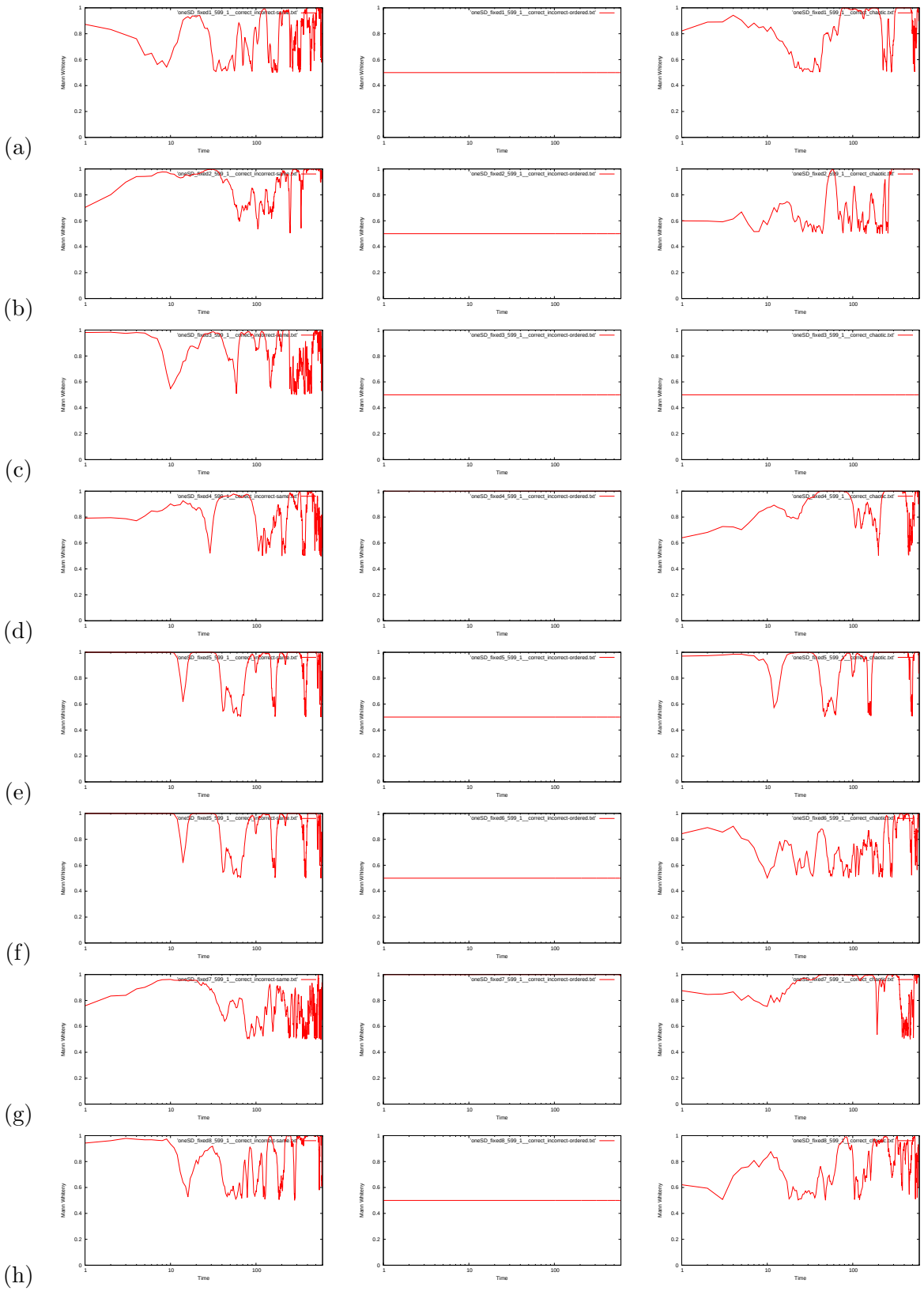


Figure 3.6: Mann-Whitney Test Graphs. Each row has graphs of a rule, (a) Fixed1, (b) Fixed2, (c) Fixed3, (d) Fixed4, (e) Fixed5, (f) Fixed6, (g) Fixed7, (h) Fixed8. Each column has graphs that compare two specific class of patterns. First column compares correct-ordered vs incorrect-same, second column compares correct-ordered vs incorrect-ordered, third column compares correct-ordered vs incorrect-chaotic patterns.

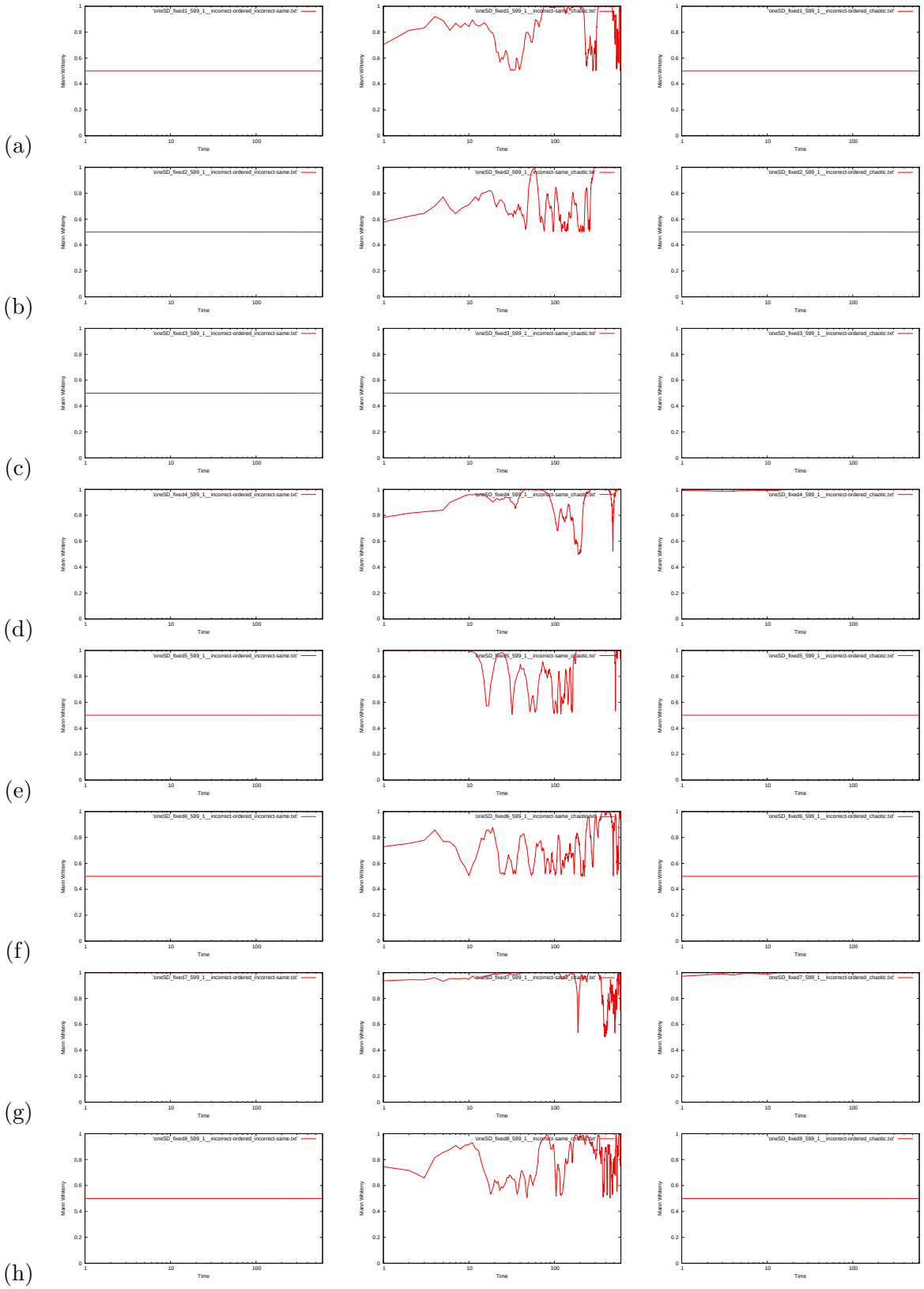


Figure 3.7: Graphs of Mann-Whitney Test. Each row corresponds to graphs of a rule (a) Fixed1, (b) Fixed2, (c) Fixed3, (d) Fixed4, (e) Fixed5, (f) Fixed6, (g) Fixed7, (h) Fixed8. Each column has graphs that compare two specific class of patterns. First column compares incorrect-same vs incorrect-ordered, second column compares incorrect-same vs incorrect-chaotic patterns, third column compares incorrect-ordered vs incorrect-chaotic patterns.

CHAPTER 4

SUMMARY AND CONCLUSIONS

We did five experiments to verify our hypothesis. The first experiment is "Evaluating Set-complexity". We created different synthetic patterns and evaluated their set-complexities. The results of this experiment have confirmed that complex patterns have maximal information flow. This result marked a mile stone in the process of verifying the first part of our hypothesis that the set-complexity measure we have used is correctly measuring the information in cellular automata.

Second experiment is "Rule Accuracy". To determine the hardest problems, we solved problems of different densities to observe their effect on the accuracy of the rules. Results of the experiment confirmed that problems with density 50% are the hardest.

Third experiment "Rule Robustness". To determine the sensitivity of the rules to mutations, we solved problems of different densities with mutated rules. The results confirmed that each rule has locally maximal accuracy. Which also means that there is no doubt on the accuracy of the rules used to solve the majority problems.

Fourth experiment "Set-complexity Trajectory". To find out if there is difference in the information flow of the systems that converge to different outcome, we calculated and observed their set-complexity trajectories. This experiment's result leads us to the conclusion that there is difference between the set-complexity trajectories of the systems with ordered outcomes and systems with chaotic outcomes. Hence, from the results of the second, third and fourth experiment, we prove the second part of our hypothesis, solving hard majority problems with cellular automata requires maximal information flow, is true.

Last experiment "Mann-Whitney". We did Mann-Whitney test, to determine if there is significant difference between median set-complexity trajectories of the systems with different outcomes at each time-step. That is, to test, if there is significant difference in the

information flow of different systems. The results showed no significant difference between the classes of results. There are some potential errors in the calculation of p-values that failed to determine significance. Fixing these problems may help in determining significance. Currently, with the results of our last experiments the last part of our hypothesis, that set complexity is maximized when CA's solve hard problem correctly, is inconclusive.

REFERENCES

[1] Andre, D., Bennett, F. H., and Koza, J. R. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In *Proceedings of the First Annual Conference on Genetic Programming* (Cambridge, MA, USA, 1996), GECCO '96, MIT Press, pp. 3–11.

[2] Berlekamp, E. R., Conway, J. H., and Guy, R. K. *Winning Ways for Your Mathematical Plays, Vol. 4*. AK Peters, Mar. 2004.

[3] Burrows, M., Wheeler, D. J., Burrows, M., and Wheeler, D. J. A block-sorting lossless data compression algorithm, 1994.

[4] Cebrián, M. C., Alfonseca, M., and Ortega, A. Common pitfalls using normalized compression distance: what to watch out for in a compressor. *Communications in Information and Systems 5* (2005), 367–384.

[5] Chen, X., Francia, B., Li, M., McKinnon, B., and Seker, A. Shared Information and Program Plagiarism Detection. *IEEE Transactions on Information Theory 50*, 7 (July 2004), 1545–1551.

[6] Cilibrasi, R., and Vitanyi, P. M. B. Clustering by Compression. *IEEE Transactions on Information Theory 51*, 4 (Apr. 2005), 1523–1545.

[7] Coico, R., Sunshine, G., and Benjamini, E. *Immunology: A Short Course*. Wiley-Liss, Oct. 2003.

[8] Das, R., Mitchell, M., and Crutchfield, J. P. A Genetic Algorithm Discovers Particle-Based Computation in Cellular Automata. In *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving*

from Nature: Parallel Problem Solving from Nature (London, UK, UK, 1994), PPSN III, Springer-Verlag, pp. 344–353.

[9] Davis, L. D., and Mitchell, M. Handbook of Genetic Algorithms. *Van Nostrand Reinhold* (1991).

[10] Dubacq, J. C., Durand, B., and Formenti, E. Kolmogorov complexity and cellular automata classification. *Theoretical Computer Science 259*, 1-2 (May 2001), 271–285.

[11] Flann, N. S., Mohamadlou, H., and Podgorski, G. J. Criticality of Spatiotemporal Dynamics in Contact Mediated Pattern Formation Information Processign in Cells and Tissues. vol. 7223 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2012, ch. 15, pp. 105–116.

[12] Fowler, D. R., Meinhardt, H., and Prusinkiewicz, P. Modeling seashells. *Computer Graphics 26*, 2 (1992), 379–387.

[13] Gach, . One-Dimensional Uniform Arrays That Wash Out Finite Islands. *Probl. Peredachi Inf. 14*, 3 (1978), 92–96.

[14] Galas, D. J., Nykter, M., Carter, G. W., Price, N. D., and Shmulevich, I. Biological Information as Set-Based Complexity. *IEEE Transactions on Information Theory 56*, 2 (Feb. 2010), 667–677.

[15] Iclănzan, D., Fülöp, P. I., Chira, C., and Gog, A. Towards the efficient evolution of particle-based computation in cellular automata. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation* (New York, NY, USA, 2011), GECCO '11, ACM, pp. 835–836.

[16] Kolmogorov, A. N. Three approaches to the quantitative definition of information. *Problems in Information Transmission 1* (1965), 1–7.

[17] Mann, H. B., and Whitney, D. R. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics 18*, 1 (1947), 50–60.

[18] Marques-pita, M., and Rocha, L. M. L.M.: Conceptual structure in cellular automata: The density classification task. In *Proceedings of the Eleventh International Conference on Artificial Life (Alife XI* (2008).

[19] Mitchell, M., Hraber, P. T., and Crutchfield, J. P. Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations. In *Complex Systems* (1993), vol. 7, pp. 89–130.

[20] Oliveira, G. M. B., Bortot, J. C., and de Oliveira, P. P. B. Multiobjective evolutionary search for one-dimensional cellular automata in the density classification task. In *Proceedings of the eighth international conference on Artificial life* (Cambridge, MA, USA, 2003), ICAL 2003, MIT Press, pp. 202–206.

[21] Tomlin, C. J., and Axelrod, J. D. Biology by numbers: mathematical modelling in developmental biology. *Nat Rev Genet 8*, 5 (May 2007), 331–340.

[22] Weinert, W. R., and Lopes, H. S. Evaluation of dynamic behavior forecasting parameters in the process of transition rule induction of unidimensional cellular automata. *Biosystems 99*, 1 (Jan. 2010), 6–16.

[23] Wolfram, S. Cellular automata as models of complexity. *Nature 311* (Oct. 1984), 419–424.

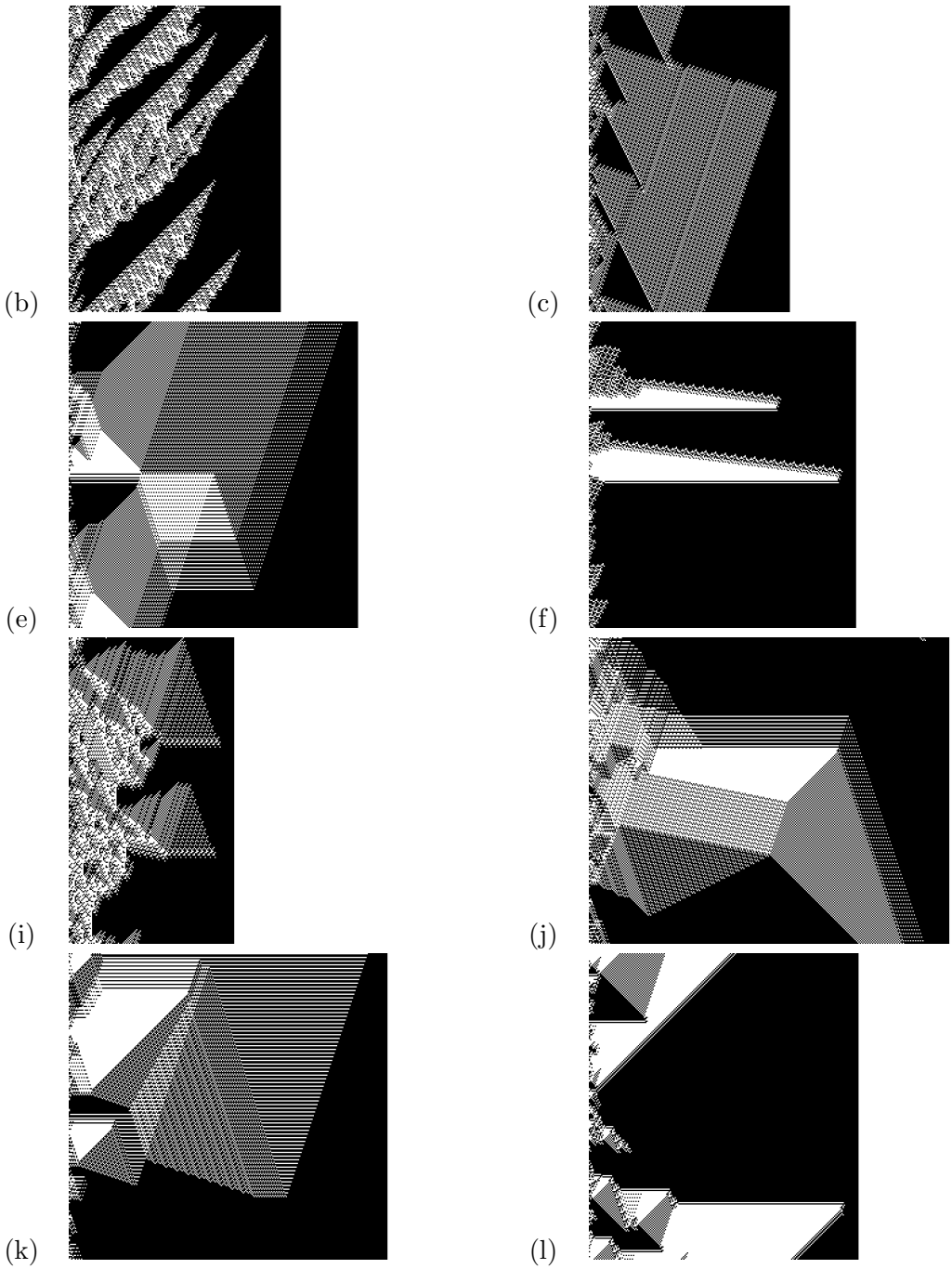


Figure 5.1: Additional Sample Correct Result Patterns