Automatic Classification of One-Dimensional Cellular Automata

Rochester Institute of Technology
Computer Science Department
Master of Science Thesis

Daniel R. Kunkle

July 17, 2003

Advisor: Roger S. Gaborski Date

Reader: Peter G. Anderson Date

Observer: Julie A. Adams Date

Abstract

Cellular automata, a class of discrete dynamical systems, show a wide range of dynamic behavior, some very complex despite the simplicity of the system's definition. This range of behavior can be organized into six classes, according to the Li-Packard system: null, fixed point, two-cycle, periodic, complex, and chaotic. An advanced method for automatically classifying cellular automata into these six classes is presented. Seven parameters were used for automatic classification, six from existing literature and one newly presented. These seven parameters were used in conjunction with neural networks to automatically classify an average of 98.3% of elementary cellular automata and 93.9% of totalistic $k = 2$ $r = 3$ cellular automata. In addition, the seven parameters were ranked based on their effectiveness in classifying cellular automata into the six Li-Packard classes.

Acknowledgments

Thanks to Gina M.B. Oliveira for providing a table of parameter values for elementary CA, which was valuable in double-checking my parameter calculations.

Thanks to Stephen Guerin for allowances and support during my absence from RedfishGroup while finishing this work.

Thanks to Samuel Inverso for discussions, suggestions and distractions, from which this work has benefited tremendously.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A cellular automaton (CA) consists of a regular lattice of cells, possibly of infinite size in theory but finite in practical simulation. This regular lattice can be of any dimension. Each cell can take on one of a finite number of values. The values of the cells are updated synchronously, in discrete time steps, according to a local rule, which is identical for all cells. This update rule takes into account the value of the cell itself and the values of neighboring cells within a certain radius.

One-dimensional CA, which are the focus of this thesis, are traditionally represented visually as a space-time diagram. This diagram presents the initial configuration of the CA as a horizontal line of cells, colored according to their state. For binary, or two-state, CA the state 0 is represented by white and the state 1 is represented by black. Each subsequent configuration of the CA is presented below the previous one, creating a two-dimensional picture of the system's evolution. Figure 1.1 shows a space-time diagram for a one-dimensional CA with a rule specifying a cell take on the state 1 at time $t + 1$ if either of its two closest neighbors are in state 1 at time t and take on state 0 otherwise. The boundary cells in this CA, and all others presented later, have "wrap-around" connections to cells on the opposite side. That is, the left-most and right-most cells are considered neighbors, creating a circular lattice.

1.1 Definition of Cellular Automata

A CA has three main properties, dimension d, states per cell k, and radius r. The dimension specifies the arrangement of cells, a one dimensional line, two dimensional plane, etc. The states per cell is the number of different values any one cell can have and $k \geq 2$. The radius defines the number of cells in each direction that will have an effect on the update of a cell. For one-dimensional CA, a radius of r results in a neighborhood of size $m = 2r + 1$. For CA of higher dimension it must be specified whether the radius refers only to directly adjacent cells or includes diagonally adjacent cells as well. For example, a two-

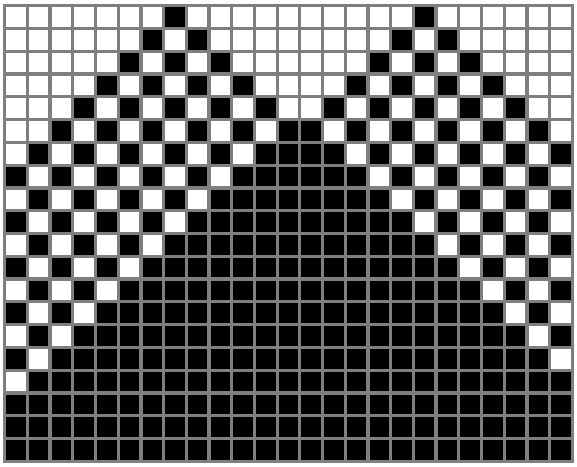


Figure 1.1: Space-time diagram of a one-dimensional CA. Each cell takes on state 1 at $t+1$ if either neighbor is in state 1 at time t, and takes on state 0 otherwise.

dimensional CA of radius 1 will result in a neighborhood of either size 5 or 9. This thesis will deal exclusively with one-dimensional CA.

Each cell has an index i, the state of a cell at time t is given by S_i^t. The state of cell i along with the state of each cell in the neighborhood of i is defined as η_i^t.

The local rule used to update each cell is often referred to as a *rule table*. This table specifies what value the cell should take on for every possible set of states the neighborhood can have. The number of possible sets of states the neighborhood can have is k^m, resulting in k^{k^m} possible rule tables. The application of the rule to one cell for one time step is defined by the function $\Phi(\eta_i^t)$ yielding S_i^{t+1}.

Boundary condition for the lattice are most often taken into account by wrapping the space into a finite, unbounded topology: a circle for one-dimensional CA, a torus for two-dimensional CA, and hyper-tori for higher dimensions.

1.2 A Brief History of Cellular Automata

Cellular automata have shown up in a large number of diverse scientific fields since their introduction by John von Neumann in the 1950's. A recent history of this body of work is given by Sarkar in [28]. Sarkar splits the field into three main categories: classical, games, and modern. These same three categories have been identified by others, including McIntosh in [21], where he presents a chronological history of CA. McIntosh labels these three categories by their defining works, namely von Neumann's self-reproducing machines for classical, Conway's Game of Life for games, and Wolfram's classification scheme for modern.

1.2.1 von Neumann's Self-Reproducing Machines

Classical research is based on von Neumann's original use of CA as a tool for modeling biological self-reproduction, which is collected by Burks in [30]. Von Neumann's self-reproducing machine was a two-dimensional cellular automaton with 29 states and a five cell neighborhood. This extremely complex CA was in fact a universal computer and a universal constructor that when given a description of any machine could construct that machine. So, when given its own description it would self-reproduce. E. F. Codd later constructed a variant of von Neumann's self-reproducing machine requiring only 8 states [4]. More recently, Langton constructed much less complicated CA with 8 states that is capable of self-reproduction without requiring a universal computer/constructor.

1.2.2 Conway's Game of Life

Conway's Game of Life is the most prominent example in the CA games category and is the most well known CA in general. The Game of Life was first popularized in 1970 by Gardner in his Scientific American column *Mathematical Games* [10, 11].

The Game of Life is a two-dimensional CA with two states and an update rule considering a cell's eight neighbors as follows: if two neighbors are black, then the cell stays the same color; if three neighbors are black the cell becomes black; if any other number of neighbors is black the cell becomes white. Much of the popularity of the Game of Life comes from the ecological vocabulary used to describe it. Cells are said to be *alive* or *dead* if they are black or white, respectively. The logic of the update rule is described in terms of *overcrowding* and *isolation*, implying that two or three alive neighbors is good for a cell. The "players" of the Game of Life were most interested in finding stable and locomotive structures, or *life forms*, that could survive in their environment. A whole zoo of life forms has been cataloged, with creative names like *gliders*, *puffers*, and *spaceships*.

Along with the game's popularity in recreational computing it is also the subject of substantial research, including the proof that the Game of Life is a universal computer [2].

1.2.3 Wolfram's Classification

The most recent era of research has its roots in the work of Wolfram, involving the study of a large set of one-dimensional CA. Much of the foundation of this area was laid in the 1980's and is collected in [32]. More recently, Wolfram has released his work as a large volume entitled *A New Kind of Science* [33].

This work marks a shift away from studying specific, complicated CA toward the empirical study of a large set of very simple CA. Wolfram noticed very different dynamical behaviors in simple CA and classified them into four categories, showing a range of simple, complex, and chaotic behavior. This and other classification schemes are detailed in Chapter 3.

1.3 Goals and Methods

The goals of this thesis are:

1. A comprehensive study of methods for classifying cellular automata based on dynamical behavior. Included are a number of classification systems (Chapter 3), quantifications (Chapter 4), and parameterizations (Chapter 5). This study will be restricted to one-dimensional, two-state CA, but the methods presented here can be extended to CA of more complicated structure.

2. A new classification parameter based on the incompressibility of a CA rule table, υ, is presented in Chapter 5.8.

3. Chapter 7 uses both qualitative and quantitative measures to compare the effectiveness of each parameter in classifying CA.

4. The parameters are used in conjunction with neural networks to automatically classify CA. These methods and their results are detailed in Chapter 6.

All code required to accomplish these tasks was written in MATLAB and is included in Appendix D. MATLAB was chosen mainly for it extensive Neural Network Toolbox, which allowed efforts to focus primarily on implementing and studying cellular automata instead of neural networks (though the neural network used for CA classification is presented in Chapter 6). Further, the matrix-centric nature of MATLAB is often useful when dealing with CA, which themselves are matrices. One downside of MATLAB is that it is an expensive commercial product. However, Octave (octave.org) is an open source alternative that is "mostly" compatible with MATLAB, though it can sometimes be difficult to run MATLAB code in Octave due to subtle differences. Also, Octave does not have a neural network package.

Also included in the appendices are data tables listing the calculated parameter values for the CA used in training and testing the neural network. These tables are useful for verifying the results of parameter calculations from other implementations. Sample space-time diagrams of each of the CA used in training and testing are also given, providing a reference for verifying classification accuracy.

It is the aim that work presented here not only explore new methods of classifying CA but also provide a comprehensive starting point for further research in this and related areas.

Chapter 2

Rule Spaces

A few rules spaces have attracted most of the attention in efforts to classify cellular automata (CA). These spaces are usually relatively small and contain simple rules, allowing a complete and in depth study. Two such sets of rules are defined here: the elementary rule space and totalistic rule spaces. These two spaces make up the training and testing sets of the neural network designed to classify CA based on the parameterizations presented in Chapter 5.

2.1 Elementary Rule Space

The elementary one-dimensional CA are those with $k = 2$, and $r = 1$. This yields a rule table of size 8, and 256 possible different rule tables. The numbering scheme used here for these elementary rules is that described by Wolfram [32]. Rule tables for elementary CA are of the form $(t_7t_6t_5t_4t_3t_2t_1t_0)$, where the neighborhood (111) corresponds to t_7, (110) to t_6, ..., and (000) to t_0. The values t_7 through t_0 can be taken to be a binary number, which provides each elementary CA with a unique identifier, in the decimal range 0 to 255.

Through reflection and white-black symmetry the elementary rule space is reduced to 88 rule groups [32, 19]. These rule groups each have 1, 2 or 4 rules that are behaviorally equivalent to each other, the only difference being either a mirror reflection, a white-black negation, or both. A rule $(t_7t_6t_5t_4t_3t_2t_1t_0)$ is equivalent to rule $(t_7t_3t_5t_1t_6t_2t_4t_0)$ by reflection, to rule $(\bar{t_0}\bar{t_1}\bar{t_2}\bar{t_3}\bar{t_4}\bar{t_5}\bar{t_6}\bar{t_7})$ by negation, and to $(\bar{t_0}\bar{t_4}\bar{t_2}\bar{t_6}\bar{t_1}\bar{t_5}\bar{t_3}t_7)$ by both reflection and negation. Table 2.1 shows the 88 behaviorally distinct rule groups. The rule with the smallest decimal representation is taken as the representative rule in each group. The column on "dynamics" will be explained in section 3.

2.2 Totalistic Rule Space

Totalistic rules are a subset of normal CA rules where the update rule depends on the sum of the states of a cell's neighborhood instead of the specific pattern

Table 2.1: Elementary rule groups and dynamics (reproduced from [24]).

Group		Dynamics	Group		Dynamics	Group		Dynamics
0	255	Null	**35**	49,59,115	Two-Cycle	**108**	201	Two-Cycle
1	127	Two-Cycle	**36**	219	Fixed Point	**110**	124,137,193	Complex
2	16,191,247	Fixed Point	**37**	91	Two-Cycle	**122**	161	Chaotic
3	17,63,119	Two-Cycle	**38**	52,155,211	Two-Cycle	**126**	129	Chaotic
4	223	Fixed Point	**40**	96,235,249	Null	**128**	254	Null
5	95	Two-Cycle	**41**	97,107,121	Periodic	**130**	144,190,246	Fixed Point
6	20,159,215	Two-Cycle	**42**	112,171,241	Fixed Point	**132**	222	Fixed Point
7	21,31,87	Two-Cycle	**43**	113	Two-Cycle	**134**	148,158,214	Two-Cycle
8	64,239,253	Null	**44**	100,203,217	Fixed Point	**136**	192,238,252	Null
9	65,111,125	Two-Cycle	**45**	75,89,101	Chaotic	**138**	174,208,244	Fixed Point
10	80,175,245	Fixed Point	**46**	116,139,209	Fixed Point	**140**	196,206,220	Fixed Point
11	47,81,117	Two-Cycle	**50**	179	Two-Cycle	**142**	212	Two-Cycle
12	68,207,221	Fixed Point	**51**		Two-Cycle	**146**	182	Chaotic
13	69,79,93	Fixed Point	**54**	147	Complex	**150**		Chaotic
14	84,143,213	Two-Cycle	**56**	98,185,227	Fixed Point	**152**	188,194,230	Fixed Point
15	85	Two-Cycle	**57**	99	Fixed Point	**154**	166,180,210	Periodic
18	183	Chaotic	**58**	114,163,177	Fixed Point	**156**	198	Two-Cycle
19	55	Two-Cycle	**60**	102,153,195	Chaotic	**160**	250	Null
22	151	Chaotic	**62**	118,131,145	Periodic	**162**	176,186,242	Fixed Point
23		Two-Cycle	**72**	237	Fixed Point	**164**	218	Fixed Point
24	66,189,231	Fixed Point	**73**	109	Chaotic	**168**	224,234,248	Null
25	61,67,103	Two-Cycle	**74**	88,173,229	Two-Cycle	**170**	240	Fixed Point
26	82,167,181	Periodic	**76**	205	Fixed Point	**172**	202,216,228	Fixed Point
27	39,53,83	Two-Cycle	**77**		Fixed Point	**178**		Two-Cycle
28	70,157,199	Two-Cycle	**78**	92,141,197	Fixed Point	**184**	226	Fixed Point
29	71	Two-Cycle	**90**	165	Chaotic	**200**	236	Fixed Point
30	86,135,149	Chaotic	**94**	133	Periodic	**204**		Fixed Point
32	251	Null	**104**	233	Fixed Point	**232**		Fixed Point
33	123	Two-Cycle	**105**		Chaotic			
34	48,187,243	Fixed Point	**106**	120,169,225	Chaotic			

of states. A totalistic CA rule can be specified by $(t_m t_{m-1} \ldots t_1 t_0)$, where m is the size of the neighborhood and each t_i specifies what state a cell will take on when the sum of the states of its neighborhood is i. The same numbering system used earlier for the full set of CA rules is also used here for totalistic rules. The rule $(t_m t_{m-1} \ldots t_1 t_0)$, with each t_i having a value in the range $[0, k)$, is seen as a base-k number. Most totalistic rules considered here are binary, $k = 2$.

Any totalistic rule can be converted easily into the normal rule format. Every position in the normal rule with a neighborhood sum of i is given the value t_i. The table below shows the same rule in both totalistic and normal form.

Form	Rule	Index
Totalistic	0 1 0 1	5
Normal	0 1 1 0 1 0 0 1	105

All totalistic rules remain unchanged under reflection because of their symmetry. A totalistic rule $(t_m t_{m-1} \ldots t_1 t_0)$ with $k = 2$ is behaviorally equivalent to $(\bar{t_0}\bar{t_1} \ldots \bar{t_{m-1}}\bar{t_m})$ under negation.

The number of totalistic rules with k states and neighborhood size m is k^{m+1}, much less than the k^{k^m} normal rules for the same k and m. Despite this much smaller set of rules totalistic CA have shown to represent all classes of behavior [31]. This can be seen in Appendix C.2 where typical patterns for all of the totalistic rules with $k = 2$ and $r = 3$ are shown. Further, Table 2.2 lists the dynamics of each of the 136 behaviorally distinct rule groups for totalistic $k = 2$ $r = 3$ rules. These dynamics were determined through manual inspection of space-time diagrams of each CA, similar to those shown in Appendix C.2.

The combined qualities of a reduced space and full behavior representation make totalistic rules a good test bed for CA classification systems in addition to elementary CA, which have traditionally been the focus.

Table 2.2: Totalistic k=2 r=3 rule groups and dynamics.

Group		Dynamics	Group		Dynamics	Group		Dynamics
0	255	Null	**49**	115	Chaotic	**113**		Chaotic
1	127	Two-Cycle	**50**	179	Chaotic	**114**	177	Chaotic
2	191	Periodic	**51**		Chaotic	**116**	209	Chaotic
3	63	Two-Cycle	**52**	211	Chaotic	**118**	145	Chaotic
4	223	Null	**53**	83	Chaotic	**120**	225	Chaotic
5	95	Periodic	**54**	147	Chaotic	**122**	161	Chaotic
6	159	Periodic	**56**	227	Chaotic	**124**	193	Null
7	31	Two-Cycle	**57**	99	Chaotic	**126**	129	Periodic
8	239	Fixed Point	**58**	163	Chaotic	**128**	254	Null
9	111	Chaotic	**60**	195	Chaotic	**130**	190	Chaotic
10	175	Chaotic	**61**	67	Two-Cycle	**132**	222	Null
11	47	Two-Cycle	**62**	131	Periodic	**134**	158	Chaotic
12	207	Chaotic	**64**	253	Null	**136**	238	Null
13	79	Periodic	**65**	125	Chaotic	**138**	174	Chaotic
14	143	Chaotic	**66**	189	Chaotic	**140**	206	Chaotic
15		Two-Cycle	**68**	221	Null	**142**		Chaotic
16	247	Fixed Point	**69**	93	Chaotic	**144**	246	Null
17	119	Chaotic	**70**	157	Chaotic	**146**	182	Chaotic
18	183	Chaotic	**72**	237	Null	**148**	214	Chaotic
19	55	Periodic	**73**	109	Chaotic	**150**		Chaotic
20	215	Chaotic	**74**	173	Chaotic	**152**	230	Periodic
21	87	Chaotic	**76**	205	Chaotic	**154**	166	Chaotic
22	151	Chaotic	**77**		Chaotic	**156**	198	Chaotic
23		Two-Cycle	**78**	141	Chaotic	**160**	250	Null
24	231	Periodic	**80**	245	Null	**162**	186	Chaotic
25	103	Chaotic	**81**	117	Chaotic	**164**	218	Null
26	167	Chaotic	**82**	181	Chaotic	**168**	234	Fixed Point
27	39	Two-Cycle	**84**	213	Chaotic	**170**		Chaotic
28	199	Chaotic	**85**		Chaotic	**172**	202	Chaotic
29	71	Periodic	**86**	149	Chaotic	**176**	242	Fixed Point
30	135	Chaotic	**88**	229	Complex	**178**		Chaotic
32	251	Null	**89**	101	Chaotic	**180**	210	Chaotic
33	123	Chaotic	**90**	165	Chaotic	**184**	226	Chaotic
34	187	Chaotic	**92**	197	Chaotic	**188**	194	Chaotic
35	59	Two-Cycle	**94**	133	Chaotic	**192**	252	Null
36	219	Null	**96**	249	Null	**196**	220	Null
37	91	Chaotic	**97**	121	Chaotic	**200**	236	Null
38	155	Chaotic	**98**	185	Chaotic	**204**		Chaotic
40	235	Fixed Point	**100**	217	Null	**208**	244	Null
41	107	Chaotic	**102**	153	Chaotic	**212**		Chaotic
42	171	Chaotic	**104**	233	Fixed Point	**216**	228	Fixed Point
43		Chaotic	**105**		Chaotic	**224**	248	Null
44	203	Chaotic	**106**	169	Chaotic	**232**		Fixed Point
45	75	Chaotic	**108**	201	Chaotic	**240**		Fixed Point
46	139	Chaotic	**110**	137	Chaotic			
48	243	Fixed Point	**112**	241	Fixed Point			

Chapter 3

Classifications

There have been a number of schemes proposed to classify cellular automata (CA) based on their dynamics and behavior. Classification is based on the "average" behavior of the CA over all possible starting states. Many CA will seem to be in a number of different classes for certain special starting states, but for most normal initial conditions will be consistent.

3.1 Wolfram

One of the first and most well known classification systems was proposed by Wolfram [32]. The Wolfram classification scheme includes four qualitative classes which are primarily based on a visual examination of the evolution of one-dimensional CA.

- **Class I:** evolution leads to a homogeneous state in which all cells have the same value
- **Class II:** evolution leads to a set of stable or periodic structures that are separated and simple
- **Class III:** evolution leads to chaotic patterns
- **Class IV:** evolution leads to complex patterns, sometimes long-lived

The qualitative nature of these definitions leads to classes with fuzzy boundaries. Some CA, especially more complex CA with larger neighborhoods, will show properties belonging to more than one class. Classes III and IV are particularly difficult to discern between.

3.2 Li-Packard

The *limiting configuration* is the final state, or cycle of states, after a sufficient number of steps. The cycle length of the limiting configurations and the time

it takes to reach the limiting configuration are primary determinants of which class a CA belongs to. This idea is implied in the Wolfram classification, and is more explicitly presented in the Li-Packard classification.

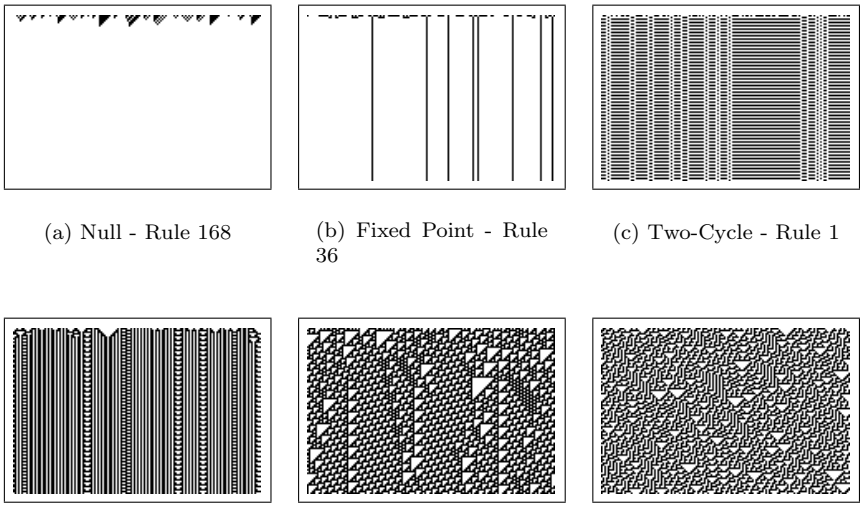
Li and Packard have iteratively developed a classification system based on Wolfram's scheme, the latest version of which has six classes [18]. It is this Li-Packard system that is adopted for classification of CA here.

- **Null:** the limiting configuration is homogeneous, with all cells having the same value.
- **Fixed point:** the limiting configuration is invariant after applying the update rule once. This includes rules that simply spatially shift the pattern and excludes rules that lead to homogeneous states.
- **Two-cycle:** the limiting configuration is invariant after applying the update rule twice, including rules that simply spatially shift the pattern.
- **Periodic:** the limiting configuration is invariant by applying the update rule L times, with the cycle length L either independent or weakly dependent on the number of cells.
- **Complex:** may have periodic limiting configurations but the time required to reach the limiting condition can be extremely long. This transient time will typically increase at least linearly with the number of cells.
- **Chaotic:** non-periodic dynamics, characterized by an exponential divergence of the cycle length with number of cells and an instability with respect to perturbations to initial conditions.

The Li-Packard classification system basically breaks Wolfram's Class II into three new classes: fixed point, two-cycle, and periodic. Examples of elementary CA in each of these six classes are provided in Figure 3.1. These six classes describe the dynamics of the 88 elementary rule groups in Table 2.1 and the 136 totalistic $k = 2$ $r = 3$ rule groups in 2.2. Table 3.1 shows the number of elementary and totalistic rule groups and rules in each class of the Li-Packard system. As the rule table grows in size the frequency of chaotic rules also increases because as soon as any subset of the rule introduces chaotic patterns those patterns dominate the overall behavior [33]. This explains the larger proportion of chaotic rules in totalistic k=2, r=3 rules over the elementary k=2, r=1 rules, which come from rules spaces of size 2^{128} and 256 respectively.

3.3 Undecidability and Fuzziness of Classifications

Culik and Yu, in [6], present a formal definition of four classes of CA that attempt to match the informal qualities of Wolfram's four classes. The Culik-Yu classification is defined as a hierarchy where each subsequent class contains



(a) Null - Rule 168

(b) Fixed Point - Rule 36

(c) Two-Cycle - Rule 1

(d) Periodic - Rule 94

(e) Complex - Rule 110

(f) Chaotic - Rule 30

Figure 3.1: Examples of elementary CA in each Li-Packard class.

Table 3.1: Number of equivalent rule groups and rules in each dynamic class of the Li-Packard system.

Li-Packard Class	Elementary Groups	Elementary Rules	Totalistic Groups	Totalistic Rules
Null	8	24	22	44
Fixed Point	32	97	11	20
Two-Cycle	28	79	9	16
Periodic	6	18	10	20
Complex	2	6	1	2
Chaotic	12	32	83	154
TOTAL	88	256	136	256

all of the previous classes. However, these classes can be easily modified to be mutually exclusive. The four classes are described in [5] as follow:

- **Class I:** CA that evolve into a quiescent (homogeneous) configuration.
- **Class II:** CA that have an ultimately periodic evolution.
- **Class III:** CA for which is is decidable whether α ever evolves to β for any two configurations α and β.
- **Class IV:** All CA.

Using this classification, Culik and Yu show that it is in general undecidable which class a CA belongs to. This is true even when choosing only between class I and class II, as presented above. Because the Culik-Yu classification is a formalization of Wolfram's four classes this undecidability can be informally seen as extending to the Wolfram classification and other derivatives of that classification, including the Li-Packard classification that is used extensively here.

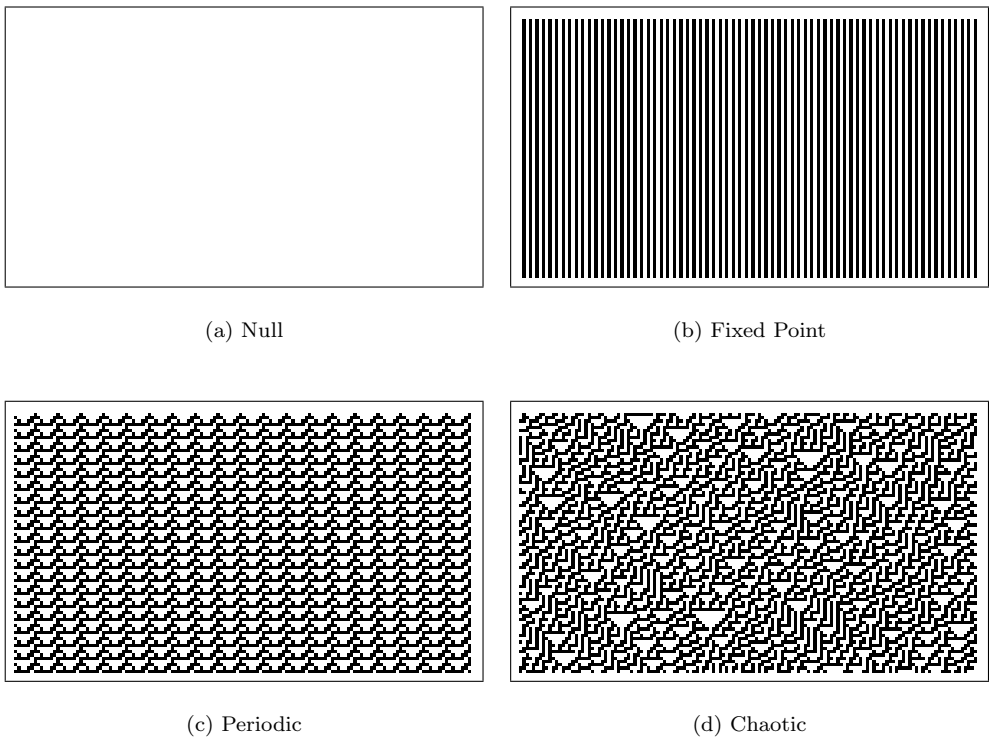
The formal undecidability of the Culik-Yu classification is also related to the informal observation of fuzziness in the classifications of Wolfram and others. The main source of this fuzziness is from the variation in CA behavior with different initial conditions. For example, the elementary rule 30, which usually exhibits chaotic behavior can also show null, fixed point, or periodic behavior depending on initial conditions (see Figure 3.2).

Another source of fuzziness is from rules that exhibit multiple classes of behavior for a single initial condition. These CA consistently show several classes of behavior and are fundamentally difficult to place in a single class. Figure 3.3 shows several examples of such borderline CA.

Both of the sources of fuzziness are addressed in this thesis. First, parameterizations of the CA rule table are used instead of quantifications of the space-time diagram. By predicting the behavior of CA directly from their rule tables the fuzziness arising from different initial conditions is avoided. A comparison of parameterizations and quantifications is given in Chapter 3.4. Second, a classification system that can handle borderline cases is needed, which is one of the primary motivations for using neural networks. The neural network presented in Chapter 6 has six outputs, one for each Li-Packard class, which are in the range $[0, 1]$. Because this output is a range instead of a binary value the neural network can specify to what degree a given CA is a member of each class. For example, the best output of the neural network when given the parameter values for the CA shown in Figure 3.3(b) might be [0 0 0 0 0 1 0.5], where each output corresponds to a class. The last two values, 1 and 0.5, specify the CA is a member of both the complex and chaotic classes to different degrees.

3.4 Quantification vs. Parameterization

The two main tools for automatically classifying CA are the quantification of space-time diagrams and the parameterization of rule tables. Quantification



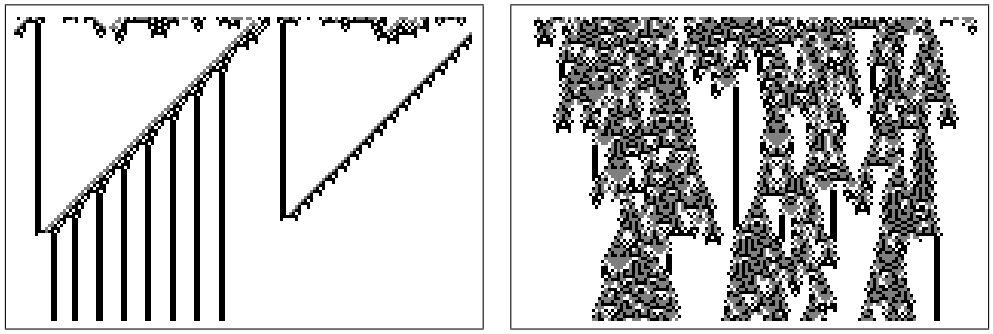
(a) Null

(b) Fixed Point

(c) Periodic

(d) Chaotic

Figure 3.2: Elementary rule 30 exhibiting many different classes of behavior (adapted from [33], page 268).



(a) Rule 219 - Fixed Point and Complex

(b) Rule 438 - Chaotic and Complex

(c) Rule 1380 - Periodic and Chaotic

(d) Rule 1632 - Null, Periodic and Chaotic

Figure 3.3: Totalistic $r = 1$ $k = 3$ rules with behavior on the borderline between several classes (adapted from [33], page 240).

is an obvious choice, as they are based directly upon observed behavior. The original classifications by Wolfram were created after manually observing a large number of space-time diagrams and noting differences in their behavior. He later quantified this behavior as a means to support this classification and provide a means for automatic classification [31].

Parameterizations are based on the rule tables of CA, instead of space-time diagrams. They, in a sense, *predict* the behavior of a CA. Actually, they predict the average behavior of a CA over a sufficiently large set of initial conditions. This means that unlike quantifiers, parameters will always have the same value for a given CA. Quantifiers must select some subset of initial conditions and the choice of those initial conditions can effect the values obtained. Accurate results for quantifiers may require calculating the evolution of CA for a large number of initial configurations over a large number of time steps. Because of this, parameters will very often require less computational effort than quantifiers.

Along with classifying CA, parameters can be used to create CA that are expected to fall withing a certain class. Langton used his λ parameter in this way to study the structure of various CA rule spaces [17].

This work focuses on the use of parameters for classifying CA. The set of parameters used is defined in Chapter 5. Quantifiers are covered more briefly in Chapter 4.

Chapter 4

Behavior Quantification

As mentioned in Chapter 3.4, behavior quantifications are measures based on the execution of a CA and on the resulting space-time diagram. This chapter presents two quantifications from the literature, input-entropy and difference pattern spreading rate. These quantifications are useful in classifying CA into systems such as those presented in Chapter 3. Though the main focus of this work is the classification of CA by parameterizations of their rules, a brief discussion of quantifications is useful in understanding classification of CA in general.

4.1 Input-entropy

Input-entropy, introduced by Wuensche [35], is based on the frequency with which each rule table position is used over a period of execution, also known as the *look-up frequency*. The look-up frequency can be represented by a histogram where each column is a position in the rule table and the height of the column is the number of times that position was used (Figure 4.1). The input-entropy at time t is defined as the Shannon entropy of the look-up frequency distribution.

$$S^t = -\sum_{i=1}^{k^m} \frac{Q_i^t}{n} log\left(\frac{Q_i^t}{n}\right) \tag{4.1}$$

where Q_i^t is the look-up frequency of rule table position i at time t.

Figure 4.1 shows three example CA with different classes of dynamic behavior: ordered, complex, and chaotic. The ordered class encompasses the null, fixed point, two-cycle, and period classes of the Li-Packard system (Chapter 3.2). The figure shows a space-time diagram for each rule along with the corresponding input-entropy and look-up frequency histogram.

Because ordered dynamics quickly settle down to a stable configuration, or set of periodic configurations, they tend to use very few positions in the rule table, resulting in a low input-entropy. Complex dynamics display a wide range

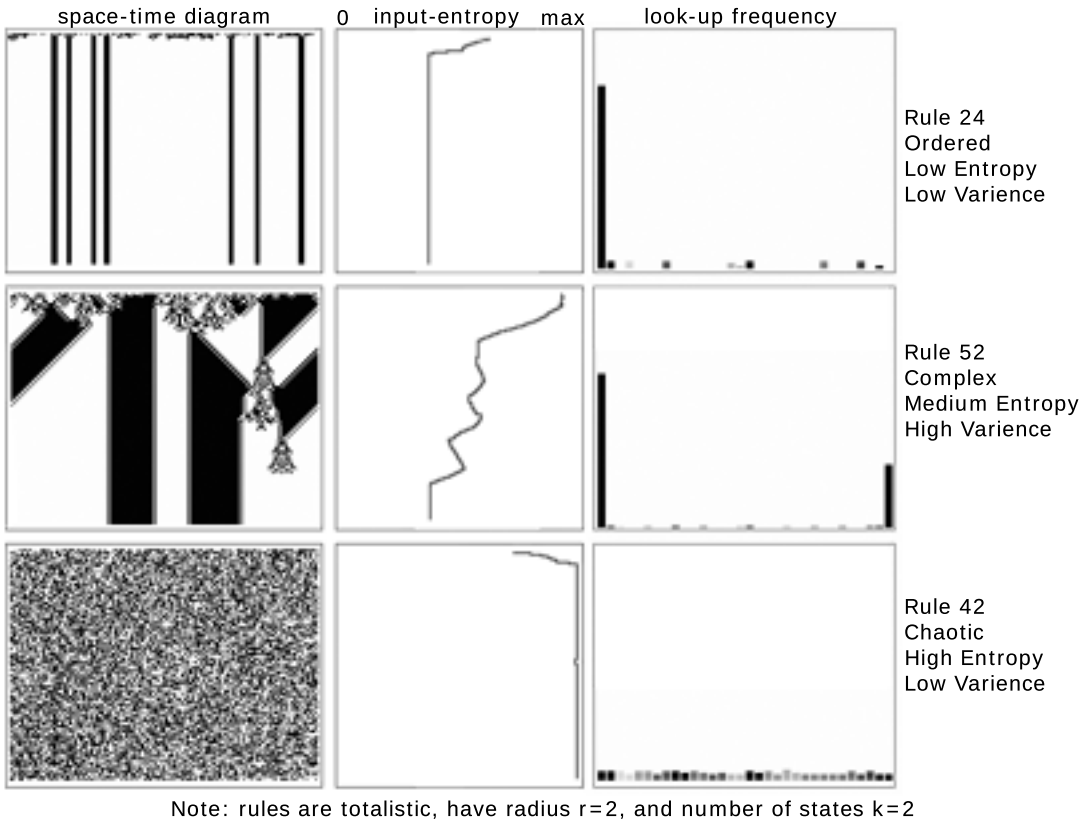


Figure 4.1: Representative ordered, complex, and chaotic rules showing space-time diagrams, input-entropy over time, and a histogram of the look-up frequency (adapted from [35], page 15) .

of input-entropies, constantly changing the set of utilized rule table positions over the course of execution. Chaotic dynamics have a high input-entropy over the entire execution. So, using the average input-entropy and the variance of the input-entropy CA can be automatically classified into ordered, complex, and chaotic classes.

A more general entropy measure, *set entropy*, was introduced earlier by Wolfram [31]. Set entropy considers the frequency of all blocks of states in the space-time diagram, not only blocks of size m. For a block size of X there are k^X possible block configurations. Over a period of execution each block configuration will have a frequency of occurrence $p_i^{(X)}$. The spatial set entropy is defined as

$$S^{(X)} = -\frac{1}{X}\sum_{j=1}^{k^X} p_j^{(X)} log p_j^{(X)} \tag{4.2}$$

The mean and variance of set entropy can be used in exactly the same manner as input-entropy, to automatically classify CA as ordered, complex, or chaotic.

4.2 Difference Pattern Spreading Rate

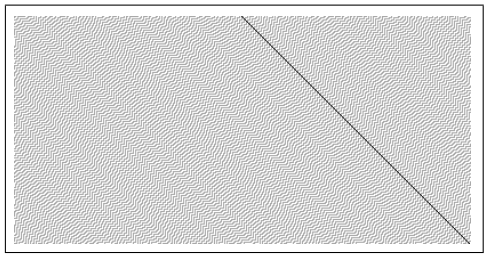
A difference pattern is a space-time diagram based on two separate executions of a CA. The CA is executed once with a random initial condition and then executed again with the state of a single cell changed. The cells of the difference pattern are on (state 1) when that cells is different in the two executions and off (state 0) otherwise. Figure 4.2 shows difference patterns for representative CA from each of the six Li-Packard classes. The first execution is shown in gray and white, the difference pattern is overlaid in black.

The *difference pattern spreading rate*, γ, is the sum of the speeds with which the left and right edges of the difference pattern move away from the center [20, 31]. A left edge moving to the right, and vice-versa, results in a negative speed for that edge.

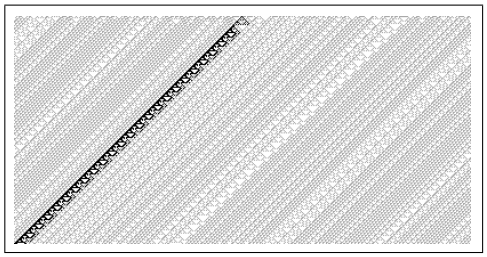
As seen in Figure 4.2, ordered dynamics (null, fixed point, two-cycle, and periodic) result in a spreading rate $\gamma = 0$, chaotic dynamic yield a high spreading rate near the maximum possible $\gamma = m$, where m is the size of the neighborhood, and complex dynamics yield highly variable spreading rates with average values somewhere between those found in ordered and chaotic CA. γ therefore provides a means of classifying CA similar to that of input-entropy.

(a) Null - Rule 40

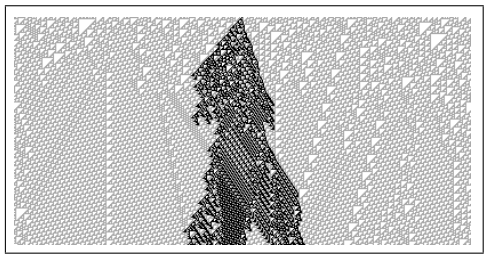
(b) Fixed Point - Rule 13



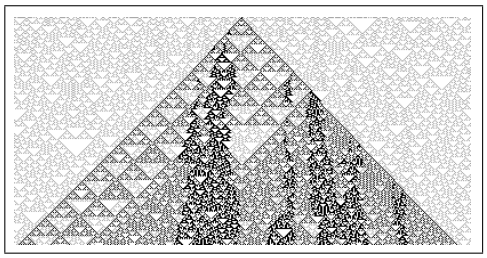
(c) Two-Cycle - Rule 15



(d) Periodic - Rule 26



(e) Complex - Rule 110



(f) Chaotic - Rule 18

Figure 4.2: Difference patterns for representative rules from each of the six Li-Packard classes.

Chapter 5

Rule Parameterization

As mentioned in Chapter 3.4, parameterization are measures based on the rule tables of cellular automata (CA). This chapter presents eight parameterization, seven from the literature and one original. Seven of these parameters, λ, Z, μ, AA, ND, AP and υ, are used later in conjunction with neural networks (NN) in automatically classifying CA. The eighth, mean field, is not used because it has a variable number values based on the size of the CA neighborhood.

5.1 λ - Activity

λ, proposed by Langton [17], is one of the simplest and most well known parameterizations of CA. The calculation of λ for a given CA rule table is as follows. One of the k states is taken to be a "quiescent" state. λ is the number of transitions in the rule table that yield non-quiescent states. For the binary CA being used here λ is simply the sum of the rule table, with 0 being the quiescent state and 1 being the only non-quiescent state. λ is also referred to as *activity* [3] because, in general, the more non-quiescent state transitions in the rule table the more active the CA will be.

A normalized form of λ is the ratio of non-quiescent transitions to the total number of transitions in the rule table. This yields a measure in the range $[0, 1]$. Because of white-black symmetry, λ is symmetric about the value 0.5 for $k = 2$ rules. So, a value of $\lambda = 0.75$ is the same as $\lambda = 0.25$. For the experiments given here, where the number of states $k = 2$, a normalized λ is calculated as

$$\lambda = 1 - \left| \frac{2 \sum_{i=1}^{n} t_i}{n} - 1 \right| \tag{5.1}$$

where n is the size of the rule table, t_i is the output of entry i in the rule table, and $|x|$ represents the absolute value of x. λ is still in the range $[0, 1]$

but rule tables equivalent by a white-black negation will yield the same value. That is, the non-normalized values λ and $1 - \lambda$ both map to the same value by Equation (5.1).

Langton observed that λ has a correlation with the qualitative behavior of the CA for which it was calculated. In particular, as λ increases from 0 to 1 the average CA behavior passes through the following classes:

$$\text{fixed point} \Rightarrow \text{periodic} \Rightarrow \text{complex} \Rightarrow \text{chaotic}$$

This corresponds to the Wolfram classification in the order:

$$\text{class I} \Rightarrow \text{class II} \Rightarrow \text{class IV} \Rightarrow \text{class III}$$

This transition from highly order to highly disordered behavior is compared by Langton to physical phase transitions through solid $\Rightarrow$ liquid $\Rightarrow$ gas. Complex dynamics can be said to be on "the edge of chaos" (or similarly, on the edge of order) because the behavior is much more difficult to predict than ordered dynamics but much less random than chaotic dynamics.

Langton admits in [17] that λ may not be the most accurate parameterization of CA, but that because of its simplicity it has merit as a coarse approximation of dynamic behavior. In Chapter 5.2 below the mean field parameterization of CA, a generalization of λ, is examined.

5.2 Mean Field

Mean field theory can provide a set of parameters for CA, similar to the λ parameter described above [14, 18, 32]. These parameters, like λ, deal with sums of the states in the CA rule table. However, instead of summing the states for all positions of the rule table, a set of mean field parameters are calculated for subsets of positions in the rule table. These rule entry subsets are chosen based on similarities in the neighborhoods of those entries.

The mean field parameters for a CA, labeled n_i, are defined by Gutowitz in [14] as

> n_i are integer coefficients counting the number of neighborhood blocks which lead to a 1, and contain themselves i 1's.

For binary CA with neighborhood size m there are $m + 1$ mean field parameters, $n_0, n_1, \ldots, n_m$. Each parameter, n_i, has a range from 0 to $\binom{m}{i}$. For elementary CA this results in the following four mean field parameters and ranges: n_0 in range $[0, 1]$, n_1 in range $[0, 3]$, n_2 in range $[0, 3]$, and n_3 in range $[0, 1]$. Each of these mean field parameters together yields a mean field cluster, denoted as $\{n_0, n_1, n_2, n_3\}$.

Although there are multiple parameters given by mean field theory, instead of the single λ parameter, there is still a large reduction in the amount of data in the mean field cluster over the full rule table. The full rule table of binary

CA grows in size as 2^{2^m} while the mean field cluster of binary CA grow in size as $m + 1$.

The normalized mean field parameters used here are given by

$$n_i = \frac{\text{number of neighborhoods with } i \text{ 1's that lead to a 1}}{\binom{m}{i}} \tag{5.2}$$

where m is the size of the neighborhood, i ranges from 0 to m, and $\binom{m}{i}$ is the number of neighborhoods in the CA rule with i 1's.

One negative aspect of the mean field parameters is that rules that are equivalent by negation, and therefore have the same dynamic behavior, will have different mean field values. For example, rule 11 (00001011) has mean field parameters (1, 1/3, 1/3, 0) and rule 47 (00101111) has mean field parameters (1, 2/3, 2/3, 0). Because of this, and because the number of parameters is not constant for rules with different radii, the mean field parameters are not used as a part of the neural network classification system presented in Chapter 6.

5.3 Z - reverse determinism

The Z parameter is defined by Wuensche and Lesser in [36] and is explored further in [34, 35]. Z is based on a reverse algorithm for determining all of the possible previous states, preimages, of a CA from a given state. Specifically, the reverse algorithm will attempt to fill the preimage from left to right (or right to left). There are three possibilities when attempting to fill a bit in the preimage:

1. The bit is deterministic (determined uniquely): there is only one valid neighborhood.
2. The bit is ambiguous and can be either 0 or 1 (for binary CA)
3. The bit is forbidden and has no valid solution

The algorithm continues sequentially for deterministic bits, will recursively follow both paths for ambiguous bits, and halt for forbidden bits. This is done until all possible preimages of the given state are found.

The Z parameter is defined as the probability of the next unknown bit being deterministic. Z is in the range $[0, 1]$, 0 representing no chance of being deterministic and 1 representing full determinism. Equivalently, low Z corresponds to a large number of preimages and high Z corresponds to a small number of preimages (for an arbitrary state).

The Z parameter, however, does not need to be calculated using the reverse algorithm from any particular state, it can be calculate directly from the rule table. Two version of the probability can be calculated from the rule table, Z_{LEFT} corresponding to the reverse algorithm executing from left to right, and Z_{RIGHT} corresponding to execution from right to left. Z is defined as the maximum of Z_{LEFT} and Z_{RIGHT}.

The following is a description of the calculation of Z according to [35].

Let n_m, where m is the size of the neighborhood, be the count of the neighborhoods, or rule table entries, belonging to deterministic pairs such that $t_{m-1}t_{m-2}\ldots t_1 0 \rightarrow T$ and $t_{m-1}t_{m-2}\ldots t_1 1 \rightarrow \bar{T}$ (not T).

Because there are 2^m neighborhoods that may belong to such deterministic pairs, the probability that the next bit is uniquely determined by a deterministic pair is $R_m = n_m/2^m$.

Further, let n_{m-1} be the count of rule table entries belonging to deterministic quadruples such that $t_{m-1}t_{m-2}\ldots t_2 0\star \rightarrow T$ and $t_{m-1}t_{m-2}\ldots t_2 1\star \rightarrow \bar{T}$, where $\star$ represents a "don't care" bit that can be either 0 or 1.

The probability that the next bit is uniquely determined by a deterministic quadruple is $R_{m-1} = n_{m-1}/2^m$.

m such probabilities are calculated for each deterministic tuple, 2-tuple, 2^2-tuple, up to a 2^m-tuple that covers the entire rule table. The probability that the next bit will be uniquely determined by at least one m-tuple is given as the union $Z_{LEFT} = R_m \cup R_{m-1} \cup \ldots \cup R_1$, which can be expressed as

$$Z_{LEFT} = R_m + \sum_{i=1}^{m-1} R_{m-1} \left(\prod_{j=m-i+1}^{m} (1 - R_j) \right) \tag{5.3}$$

where $R_i = n_i/2^k$, and $n_i =$ the count of rule table entries belonging to deterministic 2^{m-i}-tuples.

When performed conversely the above procedure yields Z_{RIGHT}. One simple implementation of Z is the maximum of performing the Z_{LEFT} procedure on the rule table and performing the Z_{LEFT} procedure again on the reflected rule table. The reflected rule table is the original rule table with bits from mirror image neighborhoods swapped. For example, the reflection of an elementary rule $t_7t_6t_5t_4t_3t_2t_1t_0$ is $t_7t_3t_5t_1t_6t_2t_4t_0$. Performing Z_{LEFT} on the reflected rule is equivalent to performing Z_{RIGHT} on the original rule.

Z is related to λ (see Chapter 5.1) in that both are classifying parameters varying from 0 (ordered dynamics) to 1 (chaotic dynamics). Further, λ, as calculated by Equation (5.1), must be $\geq Z$, as calculated by Equation (5.3). Z, however, has been found to be a better parameterization than λ in a number of respects [26, 35].

5.4 μ - Sensitivity

The sensitivity parameter μ was proposed by Binder [3], and as pointed out in [26] was earlier proposed by Pedro P. B. de Oliveira under the name "context dependence". μ is "the number of changes in the outputs of the transition rule, caused by changing the state of each cell of the neighborhood, one cell at a time, over all possible neighborhoods of the rule being considered [24]." Typically this measure is given as an average

$$\mu = \frac{1}{nm} \sum_{(v_1 v_2 \cdots v_m)} \sum_{q=1}^{m} \frac{\delta \Phi}{\delta v_q} \tag{5.4}$$

where m is the neighborhood size, $(v_1 v_2 \cdots v_m)$ represent all possible neighborhoods, and n is the number of possible neighborhoods ($n = 2^m$). $\frac{\delta \Phi}{\delta v_q}$ is the CA Boolean derivative. If $\Phi(v_1 \cdots v_q \cdots v_m) \neq \Phi(v_1 \cdots \bar{v_q} \cdots v_m)$ then $\frac{\delta \Phi}{\delta v_q} = 1$, meaning the output is sensitive to the value of v_q, otherwise $\frac{\delta \Phi}{\delta v_q} = 0$.

μ, as calculated above, will yield values in the range $[0, 1/2]$. For uniformity, a normalized version of μ in the range $[0, 1]$ will be used for all purposes here.

Similar to each of the parameters presented here so far, μ, in general, corresponds to a transition from order to chaos in CA, rules with low μ most likely being ordered and rules with high μ most likely being chaotic. This holds with intuition, as ordered systems are less sensitive to changes than chaotic systems.

5.5 AA - Absolute Activity

Absolute activity (AA), neighborhood dominance (ND), and activity propagation (AP) are three parameters proposed by Oliveira et al. [26, 24] to aid in the classification of binary CA. These parameters were built to follow a series of eight guidelines that were proposed in [26] after a study of existing parameters from the literature, including λ, mean field, μ, and Z.

Absolute activity is described here, neighborhood dominance in Chapter 5.6, and activity propagation in Chapter 5.7.

Absolute activity was proposed as an improvement on Langton's λ activity parameter (see Chapter 5.1). Specifically, λ disregards information about neighborhood structure and looks only at overall activity in the rule table, where absolute activity quantifies activity relative to the cell states of each neighborhood.

Absolute activity is defined for elementary CA in [24] as:

> the number of state transitions that change the state of the central cell of the neighborhood + number of state transitions that map the state of the central cell onto a state that is either different from the left-hand cell or from the right-hand cell of the neighborhood - 6

The subtraction of six at the end of the above description is due to the six heterogeneous neighborhoods in elementary rules (110, 101, 100, 011, 010, and 001), which will always result in at least one difference between the cells in the neighborhood and the value of the target cell. The range of this parameter for elementary rules is $[0, 8]$ and is normalized to the range $[0, 1]$ for all uses here.

Equations (5.5), (5.6), (5.7), (5.8), (5.9), and (5.10), reproduced from [24], define the absolute activity parameter for the generalized case of binary one-dimensional CA with arbitrary radius.

The non-normalized, generalized absolute activity parameter is given by:

$$A = \sum_{(v_1 v_2 \cdots v_m)} \Bigg([\Phi(v_1 \cdots v_c \cdots v_m) \neq v_c] + \sum_{q=1}^{c-1} [\Phi(v_1 \cdots v_q \cdots v_m) \neq v_q \vee \Phi(v_1 \cdots v_{m-q+1} \cdots v_m) \neq v_{m-q+1}] \Bigg) \quad (5.5)$$

where Φ is the application of the rule to a neighborhood, m is the size of the neighborhood and $c = \frac{(m+1)}{2}$ is the position of the neighborhood's center cell. The $\vee$ symbol represents the logical OR operator and *[expression]* acts as an "if" statement, returning 1 if *expression* is true and 0 if *expression* is false.

The normalized version of absolute activity is given as

$$a = \frac{A - MIN}{MAX - MIN} \quad (5.6)$$

where:

$$MIN = \sum_{(v_1 v_2 \cdots v_m)} (\min(m_0, m_1)) \quad (5.7)$$

$$MAX = \sum_{(v_1 v_2 \cdots v_m)} (\max(m_0, m_1)) \quad (5.8)$$

specifying that m_0 and m_1, defined below, be calculated for each possible neighborhood $(v_1 v_2 \cdots v_m)$

$$m_0 = \sum_{q=1}^{c} ([v_q = 0] \vee [v_{m-q+1} = 0]) \quad (5.9)$$

$$m_1 = \sum_{q=1}^{c} ([v_q = 1] \vee [v_{m-q+1} = 1]) \quad (5.10)$$

MIN and MAX, as calculated in Equations (5.7) and (5.8), are the minimum and maximum possible values of of A, as calculated in Equation (5.5)

5.6 ND - Neighborhood Dominance

Neighborhood dominance (ND) is similar to absolute activity in that it measures activity relative to neighborhood states. However, neighborhood dominance does not differentiate between the center cell of the neighborhood and perimeter cells of increasing distance from the center cell. Instead, the state of the new cell defined by a neighborhood is compared to the *dominant* state of that neighborhood. Neighborhood dominance is a count of the number of transitions that have a target state matching the dominant state of the neighborhood. The definition of neighborhood dominance for the elementary rule

space is given in [24] as:

> 3× (number of homogeneous rule transitions that establish the next state of the central cell as the state that appears the most in the neighborhood) + (number of heterogeneous rule transitions that establish the next state of the central cell as the state that appears the most in the neighborhood)

The factor of three applied to the first term compensates for the fact that there are only two homogeneous neighborhoods, (111) and (000), and six heterogeneous neighborhood containing two cells in one state and one cell in the other state. This factor also makes sense in light of findings by Li and Packard [19], which show that the neighborhoods (111) and (000) play a crucial role in determining the dynamic behavior of the CA. So much so that they termed these neighborhoods *hot bits*. Li and Packard focused on the importance of these bits using mean field parameters, presented in Chapter 5.2, where the first and last mean field parameters correspond to the hot bits.

For rules with larger neighborhoods a factor is applied to each set of neighborhoods that have the same level of homogeneity, the size of the factor increasing with the homogeneity of the neighborhoods. This ensures that sets of neighborhoods with few representative bits in the rule receive a compensating weight in the calculation of neighborhood dominance. This is shown in the generalized, non-normalized, definition of neighborhood dominance as defined in [24]

$$D = \sum_{v_1 v_2 \cdots v_m} \binom{m}{V+c} [V < c \wedge \Phi(v_1 v_2 \cdots v_m) = 0] + \sum_{v_1 v_2 \cdots v_m} \binom{m}{V-c} [V \geq c \wedge \Phi(v_1 v_2 \cdots v_m) = 1] \tag{5.11}$$

where $V = \sum_{q=1}^{m} v_q$ and $c = \frac{m+1}{2}$ is the index of the center cell in the neighborhood. The $\wedge$ symbol represents the logical AND operator. Note also that as used here $\binom{n}{k} = 0$ if $k < 0$ or $k > n$. As defined previously, *[expression]* acts as an "if" statement, returning 1 if *expression* is true and 0 if *expression* is false.

The normalized version of neighborhood dominance is

$$d = \frac{D}{2 \times \sum_{q=0}^{c-1} \binom{m}{q} \binom{m}{c+q}} \tag{5.12}$$

the denominator yielding the maximum possible value of Equation (5.11) for a rule with a neighborhood of size m.

5.7 AP - Activity Propagation

Activity propagation (AP) is the third parameter defined by Oliveira at al. in [24]. It combines the ideas of neighborhood dominance (Chapter 5.6) and sensitivity (Chapter 5.4).

Each neighborhood of size m has m corresponding neighborhoods with a hamming distance of 1. That is, m other neighborhoods can be generated by flipping each bit in a neighborhood, one at a time. For elementary CA rules, with $m = 3$, there will be three neighborhoods with hamming distance 1 for each neighborhood. In [24] these three neighborhoods are labeled the Left Complemented Neighborhood (LCN), the Right Complemented Neighborhood (RCN), and the Central Complemented Neighborhood (CCN). Activity propagation is defined for elementary rules as the sum of the following three counts:

1. Number of neighborhoods who's target state is different from the dominant state AND the target state of the LCN is different from the dominant state of the LCN.

2. Number of neighborhoods who's target state is different from the dominant state AND the target state of the RCN is different from the dominant state of the RCN.

3. Number of neighborhoods who's target state is different from the dominant state AND the target state of the CCN is different from the dominant state of the CCN.

The sum of these three counts is divided by 2 to compensate for counting each neighborhood twice.

The generalized, normalized activity propagation parameter, as given in [24], is

$$p = \frac{1}{nm} \sum_{(v_1 v_2 \cdots v_m)} \sum_{q=1}^{m} \left[\Big(\big(V < c \wedge \Phi(\cdots v_q \cdots) = 1\big) \vee \big(V \geq c \wedge \Phi(\cdots v_q \cdots) = 0\big) \Big) \wedge \Big(\big(\bar{V_q} < c \wedge \Phi(\cdots \bar{v_q} \cdots) = 1\big) \vee \big(\bar{V_q} \geq c \wedge \Phi(\cdots \bar{v_q} \cdots) = 0\big) \Big) \right] \quad (5.13)$$

where $V = \sum_{q=1}^{m} v_q$, $\bar{v_q}$ is the complement of v_q, $\bar{V_q} = V - v_q + \bar{v_q}$, m is the size of the neighborhood, $c = \frac{m+1}{2}$ is the index of the center cell of the neighborhood, and n is the number of possible neighborhood $(v_1 v_2 \cdots v_m)$. As defined previously, *[expression]* acts as an "if" statement, returning 1 if *expression* is true and 0 if *expression* is false.

5.8 υ - Incompressibility

Dubacq, Durand, and Formenti, in [9], utilize algorithmic complexity, specifically Kolmogorov complexity, to define a CA classification parameter κ. They prove that the set of all possible CA parameterizations is enumerable, that there exists at least one "optimal" parameter, and that $\kappa(x)$ is one such optimal parameter

$$\kappa(x) = \frac{K(x|l(x)) + 1}{l(x)} \tag{5.14}$$

where x is is the CA rule, $l(x)$ is the length of x, and $K(x|y)$ represents the Kolmogorov complexity of x given y. $K(x|y)$ therefore yields the length of the shortest program that will produce x given y.

However, κ is not computable, due to the fact that $K(x|y)$ is not computable. Instead, an approximation of κ can be used as a classification parameter. It is suggested in [9] to approximate κ with the compression ratio of the rule table by using any practically efficient compression algorithm.

I will define here the *incompressibility* parameter, υ, based on a run length encoding (RLE) [12] of the CA rule table. This will serve as a simple approximation of the algorithmic complexity of a given CA rule.

When attempting to compress a CA rule table it is important to consider the ordering of the bits. Normally, the bits are ordered lexicographically according to the binary representation of their neighborhoods, as demonstrated for elementary CA in Chapter 2.1 and as shown in Table 5.2(a). However, the lexicographic ordering doesn't fully take into account the similarity of the neighborhoods. Ideal for the purpose of determining incompressibility is a rule table ordered such that similar neighborhoods are proximate.

Neighborhoods that have small hamming distances can be considered similar, or related. A Gray code [15] can be used to order the neighborhoods, represented by integers from 0 to 2^m, such that all adjacent neighborhoods have a hamming distance of one. There are a number of Gray codes that can be used, in this case the *binary-reflected Gray code* will be used. One of the simplest ways to create a binary-reflected Gray code is to start with all bits zero and iteratively flip the right-most bit that produces a new number. The following is a simple algorithm to convert a standard binary number into a binary-reflected Gray code: the most significant bit of of the Gray code is equal to the most significant bit of the binary code; for each bit i, where smaller values of i correspond to less significant bits, $G_i = \text{xor}(B_{i+1}, B_i)$. Converting back from Gray code to binary is simply $B_i = \text{xor}(B_{i+1}, G_i)$.

Converting each neighborhood into the corresponding binary-reflected Gray code using the above method and rearranging the bits of the rule to match their original neighborhood yields the rule table ordering shown in Table 5.2(b).

A second way to order the neighborhoods of a rule table by similarity is by the sum of the bits in the rule table. This measure of neighborhood similarity has proven to be successful in other parameters, such as the mean field

parameters, defined in Chapter 5.2. Table 5.2(c) shows an elementary rule table ordered primarily by the sum of the bits in the neighborhoods and secondarily by lexicographic order.

Table 5.1: Four elementary rule orderings.

	1	1	1	1	0	0	0	0
neighborhood	1	1	0	0	1	1	0	0
	1	0	1	0	1	0	1	0
rule	t_7	t_6	t_5	t_4	t_3	t_2	t_1	t_0

(a) Lexicographic

	1	1	1	1	0	0	0	0
neighborhood	0	0	1	1	1	1	0	0
	0	1	1	0	0	1	1	0
rule	t_4	t_5	t_7	t_6	t_2	t_3	t_1	t_0

(b) Gray Code

	1	1	1	0	1	0	0	0
neighborhood	1	1	0	1	0	1	0	0
	1	0	1	1	0	0	1	0
rule	t_7	t_6	t_5	t_3	t_4	t_2	t_1	t_0

(c) Sum

	symmetric				negation reversible				reflection reversible			
	1	1	0	0	1	1	0	0	1	1	0	0
neighborhood	1	0	1	0	1	0	1	0	1	0	0	1
	1	1	0	0	0	0	1	1	0	0	1	1
rule	t_7	t_5	t_2	t_0	t_6	t_4	t_3	t_1	t_6	t_4	t_1	t_3

(d) Symmetric Neighborhood

A simple function based on RLE, Υ, is defined below, which returns the number of adjacent bits in a binary string that are not equal. This is equivalent to the number of contiguous blocks of ones and zeros in the binary string minus one

$$\Upsilon(s) = \sum_{i=1}^{n-1} [s_i \neq s_{i+1}] \tag{5.15}$$

where s is a string of bits $s_1 s_2 \ldots s_n$ and $[expression]$ returns 1 if the expression is true and 0 if the expression is false.

Let r_l be a CA rule table in lexicographic ordering, r_g be the binary-reflected Gray code ordering of r_l, and r_s be the sum ordering of r_l. Three corresponding versions of the incompressibility parameter can be defined as

$$\upsilon_l = \frac{1}{n-1}\Upsilon(r_l) \tag{5.16}$$

$$\upsilon_g = \frac{1}{n-1}\Upsilon(r_g) \tag{5.17}$$

$$\upsilon_s = \frac{1}{n-1}\Upsilon(r_s) \tag{5.18}$$

where n is the size of the rule table. This yields the fewest number of contiguous blocks in the rule table in either lexicographic ordering, Gray code ordering, or sum ordering, normalized to the range $[0, 1]$.

The main problem with these definitions of υ is that two CA with equivalent behavior, a rule and its equivalent reflected and/or negated rule, will often have different υ values. This leads to difficulty in using υ as a classifier and is contrary to the first of eight guidelines presented by Oliveira et al. in [26]. To attempt to minimize the problem of equivalent rules having different parameter values a new ordering is defined.

The *symmetric neighborhood ordering* is defined as follows:

1. Define three separate *rule parts*, a *symmetric part*, a *negation reversible part*, and a *reflection reversible part*

2. Traverse the lexicographic rule ordering one bit at a time.

   If the bit is from a symmetric neighborhood (is equivalent under reflection) place the bit in the symmetric neighborhood rule part.

   If the neighborhood is non-symmetric place the bit into both the negation reversible part and the reflection reversible part. Then, place the bits corresponding to the negation and reflection of that neighborhood into the negation reversible part and the reflection reversible part such that it is the same distance from the end of the rule part that the original bit is from the start of the rule part.

3. A bit is not placed into any rule part if it has already been placed there because its neighborhood is the negation or reflection of a previously encountered bit's neighborhood.

This results in the rule shown in Table 5.2(d). The symmetric rule part will be the same for a rule and the equivalent negated and/or reflected rule. The negation reversible part of the rule will simply be reversed between a rule and the equivalent negated rule. This will yield the same incompressibility factor, as described by Equation (5.15), for the negation reversible part of a rule and

its negated partner. Similarly, the reflection reversible rule part will yield the same incompressibility factors for a rule and its reflected partner.

Another version of incompressibility parameter can now be defined as follows in an attempt to minimize the difference in values between behaviorally equivalent rules

$$\upsilon_r = \frac{1}{n-3}\left(\Upsilon(r_{SYM}) + \Upsilon(r_{NEG}) + \Upsilon(r_{REF})\right) \tag{5.19}$$

where $n = 2^{\lceil m/2 \rceil} + 2\left(2^m - 2^{\lceil m/2 \rceil}\right)$ is the total size of the symmetric neighborhood rule ordering, m is the size of the neighborhood, 2^m is the total number of neighborhoods, $2^{\lceil m/2 \rceil}$ is the number of symmetric neighborhoods, r_{SYM} is the symmetric rule part, r_{NEG} is the negation reversible rule part, and r_{REF} is the reflection reversible rule part.

For the elementary rule space υ can take on nine distinct values, $\frac{0}{9}, \frac{2}{9}, \ldots, \frac{8}{9}$. The highest normalized incompressibility factor of 1 is not attainable because both the negation reversible and reflection reversible rule parts cannot be maximally incompressible at the same time.

This calculation of υ does not completely solve the problem of equivalent rules having different parameter values, but it does considerably better than Equations (5.16), (5.17), and (5.18). In the elementary rule space two different rules that are equivalent by negation or reflection will differ by $\frac{1}{9}$ and two different rules that are equivalent by both negation and reflection will differ by $\frac{2}{9}$. Unfortunately, the negation reversible and reflection reversible parts of the symmetric neighborhood ordering grow exponentially when compared to the symmetric part and it is these parts that will create discrepancies between a rule and the equivalent reflected rule. Correspondingly, the maximum discrepancy between behaviorally equivalent rules will increase with neighborhood size.

The classification efficacy of each variant of υ parameter, as well as each of the parameters from the literature presented here, will be examined in detail in Chapter 7.

Incompressibility has a relationship with λ, as many of the other parameters given above do. Just as the normalized λ in Equation (5.1) generally varies from order to chaos as it varies from 0 to 1 so does incompressibility, in each of its forms specified by Equations (5.16), (5.17), (5.18), and (5.19). The most compressible rules, homogeneous zeros or one, are null rules, the most ordered and simple. The least compressible rules are those with equal numbers of the two states, corresponding to the highest λ values. Incompressibility, however, attempts to define other regularities in the rule that may predict which dynamic class a CA rule is a member of.

Chapter 6

Class Prediction with Neural Networks

The parameters from Chapter 5 were used as inputs to a neural network (NN) for the purpose of classifying cellular automata (CA) rules into the six Li-Packard classes. This chapter will describe the NN architecture, learning algorithm, training and testing sets, and results from using the NN. Most of the results were obtained using the MATLAB Neural Network Toolkit. For more detail on network architecture and learning algorithms see [8].

Depending on the selection of training and testing sets, the trained NN was able to correctly classify between 90 and 100 percent of CA in the testing set.

6.1 Network Architecture

Classification was accomplished using a feedforward network with an input layer, two hidden layers, and an output layer. The input layer had seven neurons, one for each parameter used in classification; the two hidden layers had 30 neurons each, which was found to provide good learning rates by varying the number of neurons in each layer over a series of training trials; and the output layer had six neurons, one for each class. The transfer function for the input layer was a simple linear identity function; the two hidden layers used a tan-sigmoid transfer function, which maps values in the range $[-\infty, \infty]$ to $[-1, 1]$; and the output layer used the log-sigmoid transfer function, which maps values in the range $[-\infty, \infty]$ to $[0, 1]$.

Figure 6.1 shows a graphical representation of the NN described above. $R = 7$ inputs, labeled p, are shown to the far left. These feed into the first layer where the sum of the products of the inputs and input weights (labeled IW) is processed by function f^1 for each of the 30 neurons in layer 1. The 30 outputs of layer 1 are similarly processed by layer 2, and the outputs of layer 2 finally passed to the output layer, layer 3. The weight matrices, $IW^{1,1}$, $LW^{2,1}$, and $LW^{3,2}$, along with the transfer functions, determine the final output of the NN.

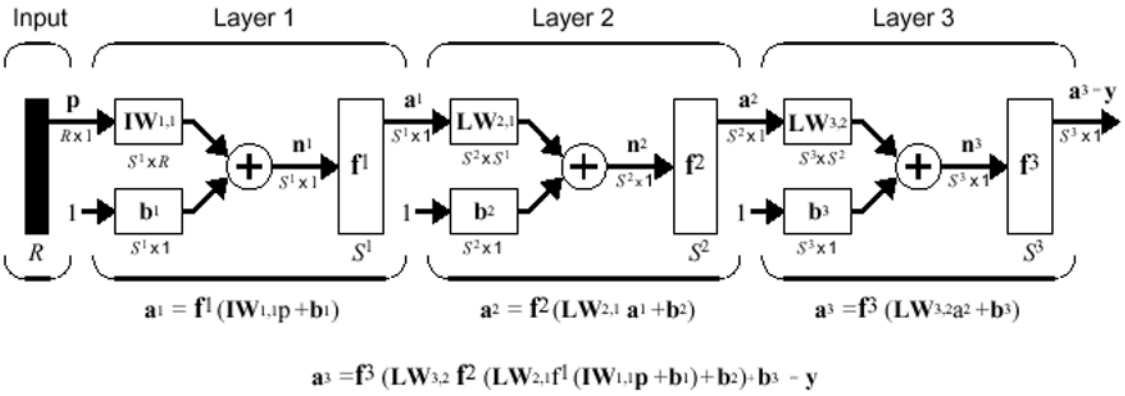


Figure 6.1: Neural network architecture (reproduced from [8]).

In this case the weight matrices are of size 30×7, 30×30, and 6×30, respectively. These weight matrices are iteratively altered by the learning algorithm presented in the next section, resulting in a network that classifies CA.

6.2 Learning Algorithm

The NN was trained using *resilient backpropagation*, a high performance version of backpropagation learning. Backpropagation in general makes small, incremental changes in the weight matrices of the network based on the error of the outputs (the error is *propagated backward* through the network). The backward propagation of error results in a *gradient* measure. In basic backpropagation the weights are modified in the direction of the negative gradient an amount proportional to the magnitude of the gradient.

Resilient backpropagation is most useful in multilayer networks using sigmoid transfer functions, such as the one presented earlier. In these networks the gradient often has very small magnitude because all of the inputs are "squashed" into a small, finite range by the sigmoid transfer functions. The small gradient magnitude results in only small changes to the weights, even though the network may be far from optimal. Resilient backpropagation speeds up the weight change, and therefore the convergence to a solution, by ignoring the magnitude of the gradient and focusing only on the sign of the gradient. While the sign of the gradient remains the same the magnitude of weight change is increased. As a minimal gradient is approached the sign of the gradient will begin to oscillate rapidly, causing a decrease in the rate of weight change.

The resilient backpropagation function in MATLAB is named `trainrp`, and is described in more detail in [8]. Resilient backpropagation was chosen over the other learning algorithms provided by MATLAB because of the learning time and memory requirements of each algorithm presented in [8], and because of similar learning times in experiments conducted with CA classification tasks.

6.3 Training and Testing Results

Training and testing sets were chosen from the 256 elementary CA and the 256 totalistic CA presented in Chapter 2. All of these CA have been manually classified, the elementary CA classifications appear in existing literature [24] and the totalistic CA having been classified by the author.

Two variants of training and testing were conducted. In the first, half of the elementary CA were used to train the NN and the remaining half were used to test the accuracy of the NN classification. Both the training and testing sets had an equal number of rules from each of the six Li-Packard classes. In the second variant of training and testing the same half-and-half split was performed using totalistic $k = 2$ $r = 3$ CA.

In the testing phase, the NN outputs six values in the range $[0, 1]$ for each input of the seven parameters. These six outputs represent the likelihood that the presented CA parameters were for a CA from each of the six Li-Packard classes. The NN is said to have correctly classified the inputs if the maximum of the six output corresponds to the actual classification of the CA. The *percent correct* is the ratio of the number of correctly classified rules from the test set and the total number of rules in the test set.

Ten separate training/testing sessions were conducted for both the elementary and totalistic CA variants. For each, a new random half of the CA set was chosen for testing and training, and a newly initialized NN was trained and tested. The NN correctly classified an average of **98.3%** of the elementary CA and **93.9%** of the totalistic CA. The slightly lower effectiveness in the totalistic space may be due to missclassificaion by the author as the process of manually observing and classifying a large number CA based on their space-time diagramss is error-prone. Further complicating the matter, the classifications are fuzzy, as mentioned earlier in Chapter 3.3, many of the CA display several classses of behavior.

Unfortunately, the NN can not be directly trained with one set of CA and be used to classify another set with a different rule size. This is because five of the seven parameters used here have different values for equivalent rules that differ only in the size of the rule table used to define them (λ and Z are the two parameters used here that are equivalent over different rule sizes). For example, a rule of neighborhood size $m = 3$ corresponds to an equivalent rule with $m = 5$, which in essence "ignores" the left-most and right-most inputs. Despite the behavioral equivalence of the CA, the parameters μ, AA, ND, AP, and υ can have different values.

It is very possible, however, that some preprocessing or separate learning process could map the parameters of the second set of CA (with different rule size) to values appropriate for the trained network. The table below shows the correlation coefficient between parameter values for the elementary CA and for the 256 equivalent CA with neighborhood size $m = 5$. The first four, λ, Z, μ, and AA all have a correlation coefficient of 1 and have simple functions to translate their values for the elementary CA into the values calculated for $m = 5$ CA. For those the function is given in the table as $y = f(x)$, where x is the

parameter value for $m = 3$ rules and y is the parameter value for $m = 5$ rules.

Parameter	Correlation Coefficient	$y = f(x)$
λ	1.00	$y = x$
Z	1.00	$y = x$
μ	1.00	$y = \frac{3x}{5}$
AA	1.00	$y = \frac{8x}{9} + \frac{1}{16}$
ND	0.99	
AP	0.90	
υ	0.51	

Chapter 7

Parameter Efficacy

It was found in Chapter 6 that a neural network (NN) can be trained to classify cellular automata (CA) based on the seven parameter set detailed in Chapter 5. This chapter considers the efficacy of each individual parameter. The first section presents a number of charts, each representing a parameter space, allowing an intuitive, qualitative perspective on the usefulness of each parameter. The second section give statistical measures of how well each subset of parameters separates the space of CA into separate classes. Lastly, the error rates of a NN trained with subsets of parameters is considered as a measure of efficacy. The quantitative measures are then used to rank the parameters by their usefulness in classifying CA.

7.1 Visualizing Parameter Space

Figures 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, and 7.7 show the distribution of the 256 elementary CA among the six Li-Packard classes for a number of parameter spaces.

The first figure, 7.1, shows all of the one-dimensional parameters in Chapter 5 that come from existing literature: λ, Z, μ, AA, ND, and AP. If any of these were a perfect classifier there would be no fewer than six values for the parameter and each value would contain CA from only one of the six classes. This is not the case, which is the reason why many parameters are required for classification. A few things are made clear by these graphs. The traditional parameters, λ, Z, μ, all range from ordered rules on the low end to chaotic rules on the high end. This makes them most useful for discriminating between null and chaotic rules. AA, ND, and AP are all useful discriminators for two-cycle rules, particularly for separating two-cycle rules from closely related fixed point rules.

Figure 7.2 displays a similar set of charts for four variants of the υ parameter. The variants differ in the ordering of the rule table that the incompressibility measure is calculated for. The nature of the incompressibility measure is to give complex, difficult to compress rules high values and simple, easily compressed

rules low values. Both ordered and chaotic rules are in a sense "simple", in that their average behavior over a long period of time is easily determined. Complex rules, however, yield behavior that is difficult to predict and require larger descriptions. The symmetric neighborhood ordering variant of υ comes closest to placing both ordered and chaotic rules at the low end while maintaining high values for complex rules. It is this variant of the rule that is used throughout this work.

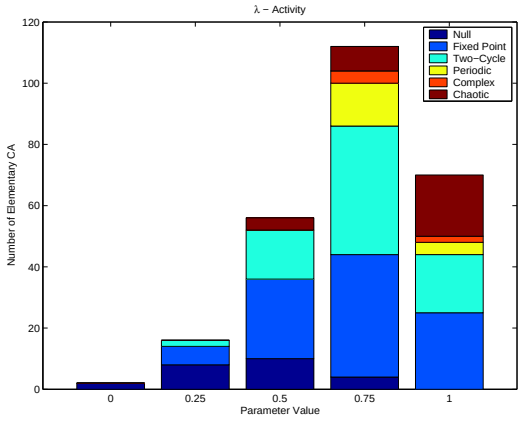
The remaining figures in this section show the distribution of elementary CA in the six Li-Packard class for combinations of the four mean field parameters. Though the mean field parameters are not used for classification here an examination of their properties is useful in understanding the space of CA rules.

Figure 7.3 shows each of the four mean field parameters as a one dimensional space by itself. Because of the small number of values for each, none are very useful for classification on their own. Figures 7.4, 7.5, and 7.6 show three of the six possible combinations of two of the four mean field parameters (the other three are simple transformations of the $n_0 n_1$ case). These cases show the use of two mean field parameters to be more useful than any one alone. $n_0 n_1$ is strong in classifying null rules; $n_0 n_3$ in two-cycle rules, and $n_1 n_2$ in fixed point rules.

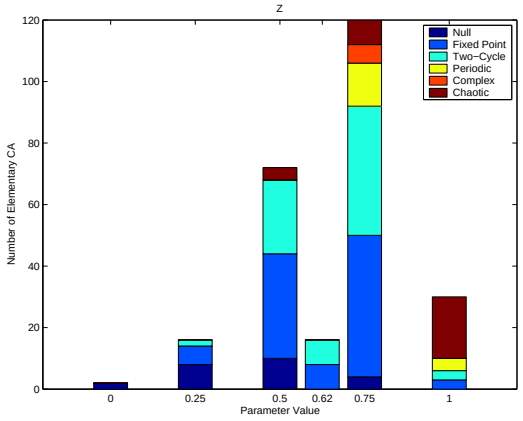
Visualizing spaces of more than two parameters is often difficult, but figure 7.7 is an attempt to visualize the four-dimensional space including all mean field parameters of elementary CA. Mean field parameters n_0 and n_3 can take on two different values; n_1 and n_2 can take on four different values. This yields 64 possible values for the combined mean field parameters. Figure 7.7(a) shows these 64 possible values in lexicographic order and the number of elementary CA from each class with that value. The other two charts show the 64 possible values in two of the possible gray code orderings. These gray code orderings were produced by the methods presented in [13]. Each set of mean field values has a Hamming distance of one from the set of values before it. These orderings, in a sense, sort the possible mean field values, placing similar values closer to each other. This captures a portion of the possible regularities in the space of CA rules. It seems from these charts that the space of all four mean field parameters is good at discriminating null, fixed point, and two-cycle CA.

7.2 Clustering and Class Overlap Measures

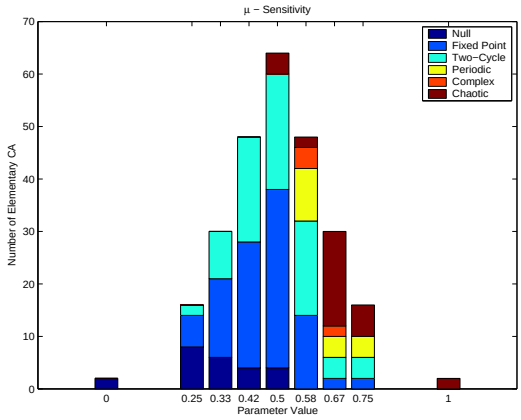
The degree to which CA cluster into distinct groups in certain parameter spaces contributes to the ease with which they can be automatically classified. Given a set of CA that are each a member of one class, two statistics can be calculated: intra and inter-cluster distances. Each CA can be represented as a seven-dimensional point in parameter space. The six classes of CA are clusters composed of the points representing the CA in that class. The *centroid* of a cluster is the mean value of all points in the cluster. The *intra-cluster distance* is defined as the mean distance of all of the points in a cluster to the corresponding centroid. The mean intra-cluster distance over all clusters is the intra-cluster distance for the whole set of data points. The *inter-cluster distance* for the



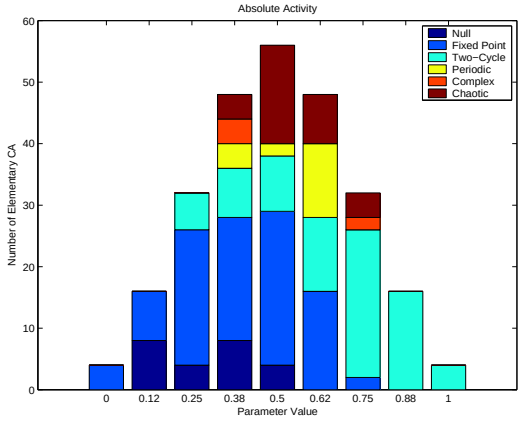
(a) λ - Activity



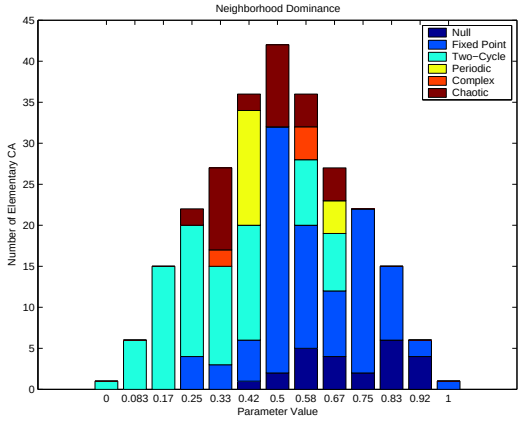
(b) Z



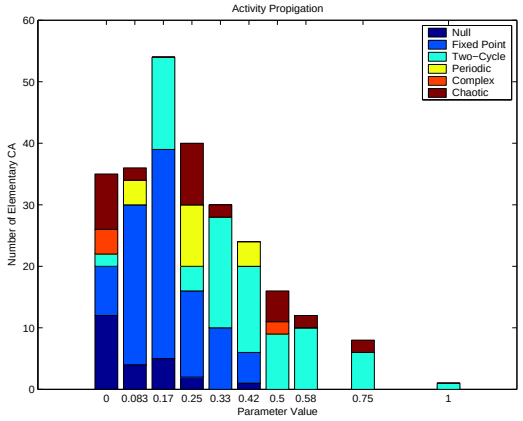
(c) μ - Sensitivity



(d) Absolute Activity

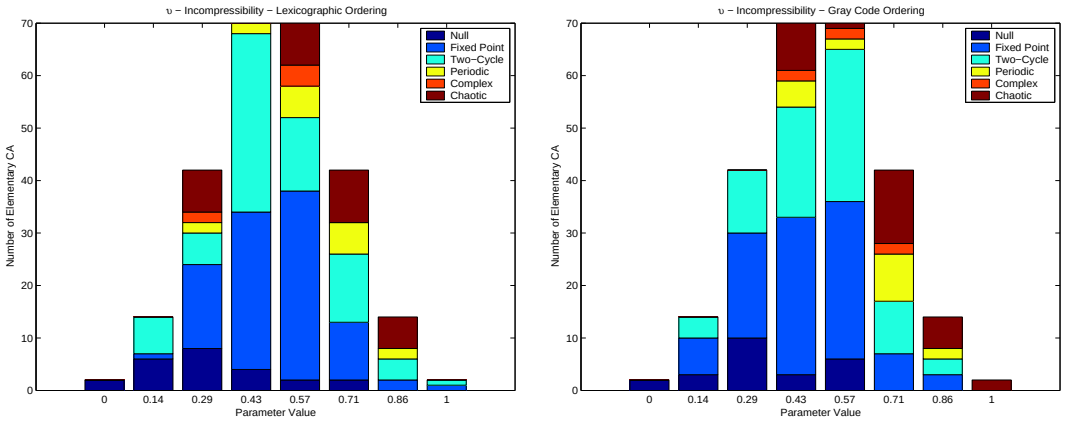


(e) Neighborhood Dominance



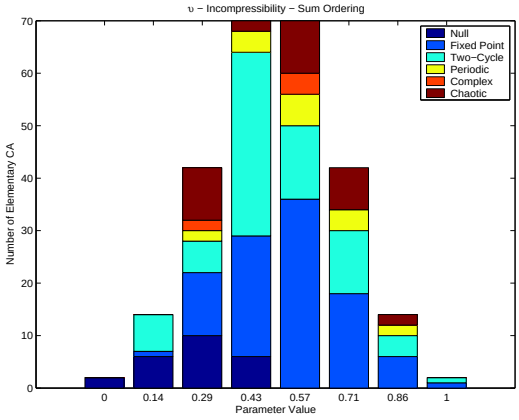
(f) Activity Propagation

Figure 7.1: Number of elementary CA rules in each Li-Packard class for each parameter value for each of six parameters from the literature.

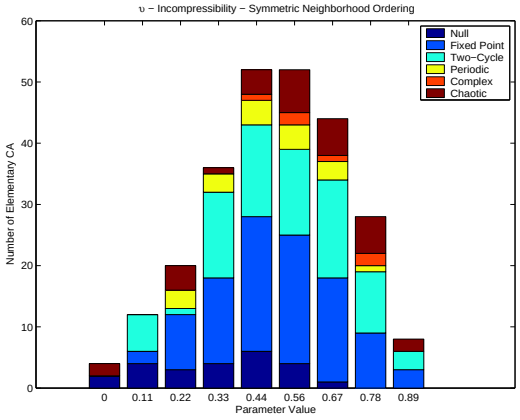


(a) υ - Lexicographic Ordering

(b) υ - Gray Code Ordering

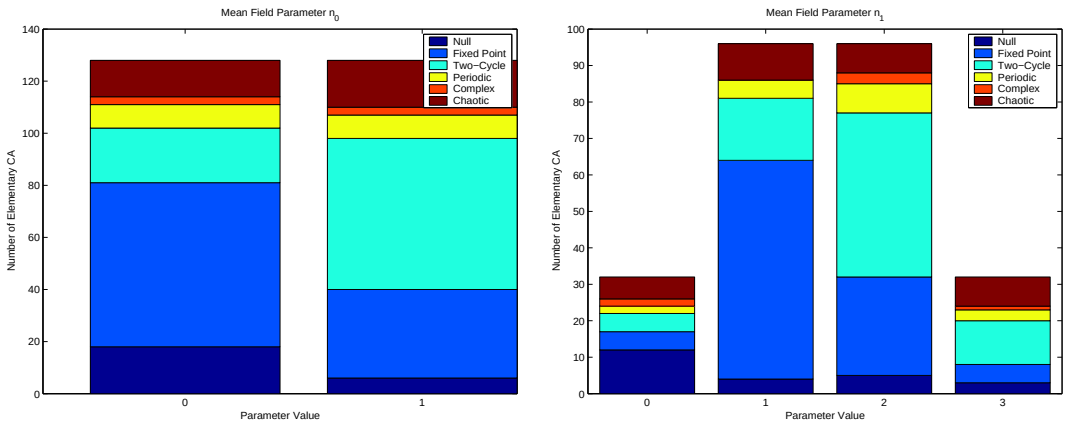


(c) υ - Sum Ordering



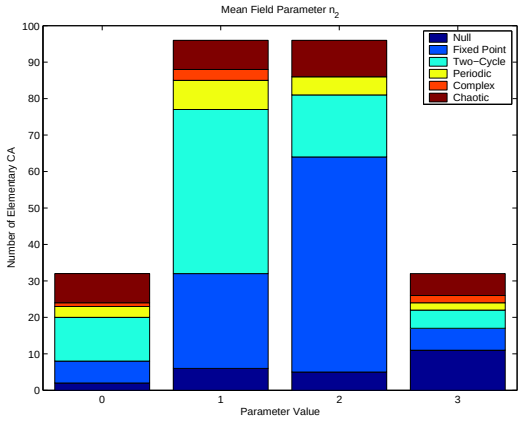
(d) υ - Symmetric Neighborhood Ordering

Figure 7.2: Number of elementary CA rules in each Li-Packard class for each parameter value for each of four variants of the incompressibility parameter.

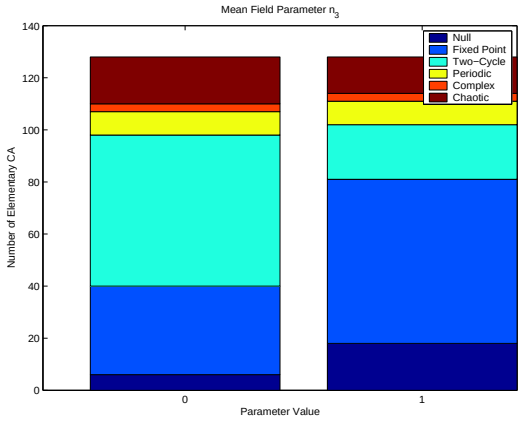


(a) n_0

(b) n_1



(c) n_2



(d) n_3

Figure 7.3: Number of elementary CA rules in each Li-Packard class for each value of each of four the four mean field parameters.

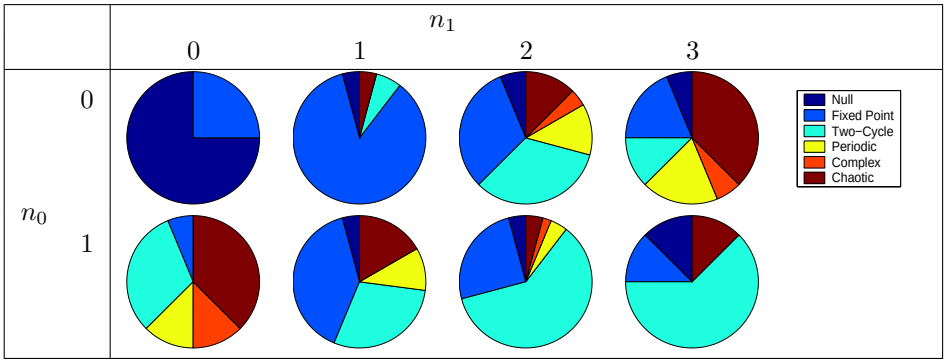


Figure 7.4: Distribution of elementary CA rules for each Li-Packard class over the two dimensional space of mean field parameters n_0 and n_1.

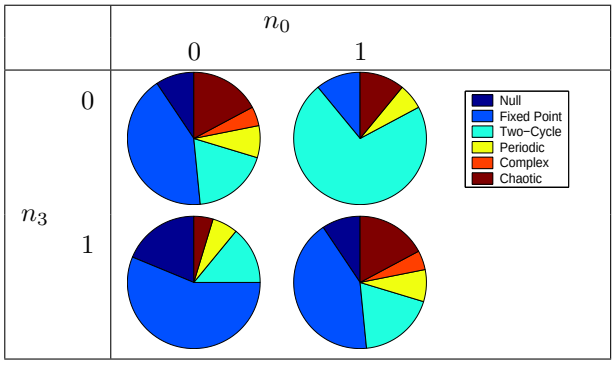


Figure 7.5: Distribution of elementary CA rules for each Li-Packard class over the two dimensional space of mean field parameters n_0 and n_3.

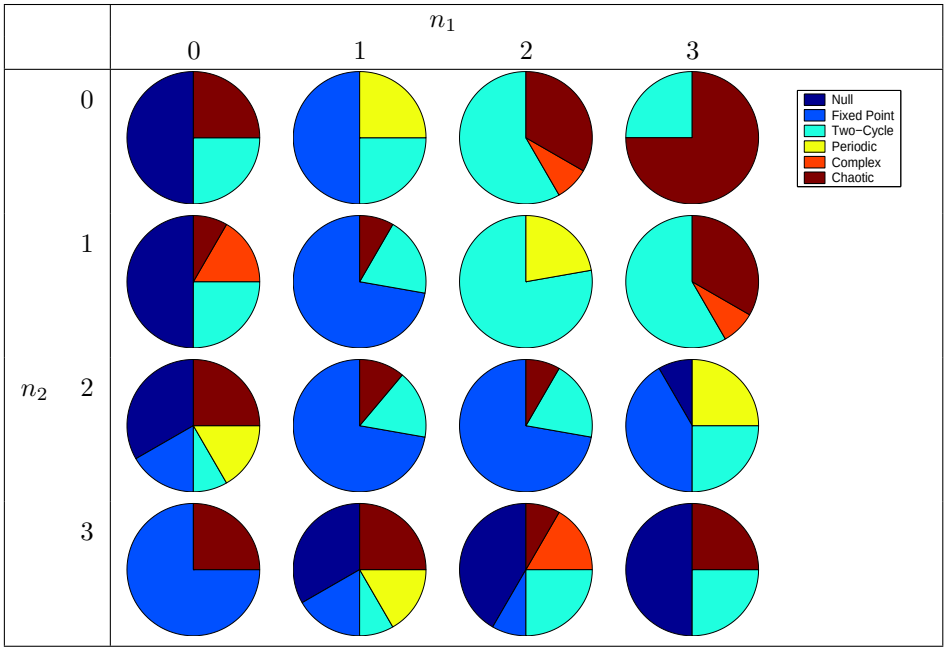
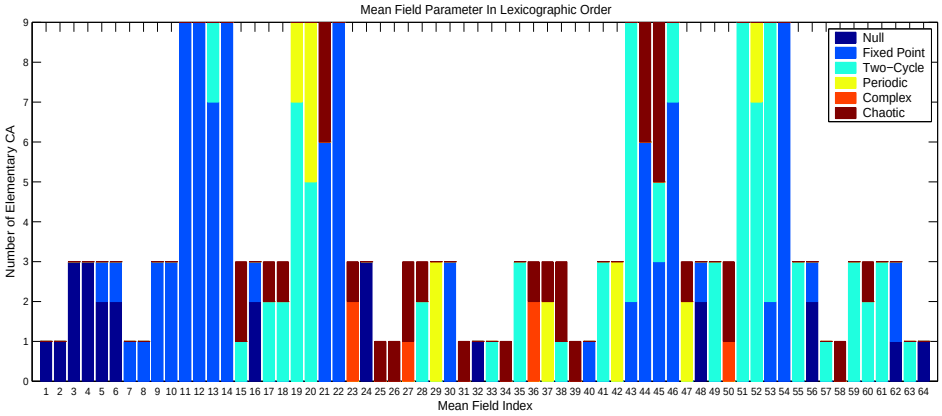


Figure 7.6: Distribution of elementary CA rules for each Li-Packard class over the two dimensional space of mean field parameters n_1 and n_2.

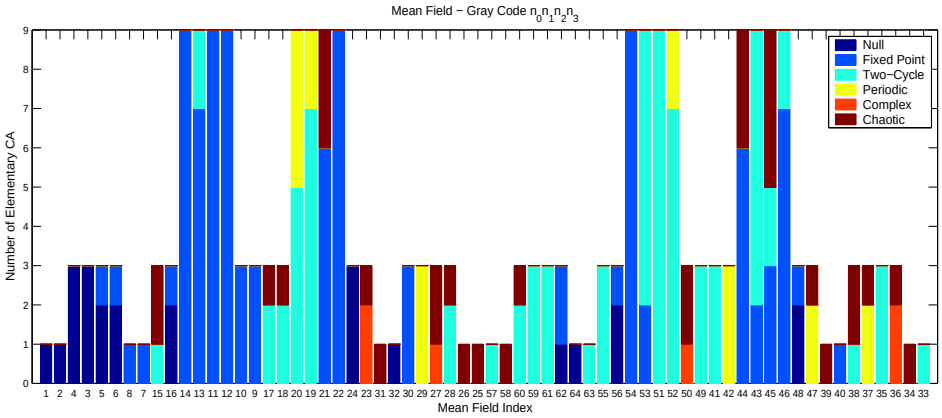
set is the mean distance between all pairs of centroids. The *clustering ratio* is defined as the ratio of inter to intra-cluster distances. A higher clustering ratio results in more compact clusters that are farther apart, making the classes of data points easier to recognize.

The intra-cluster distance, inter-cluster distance, and clustering ratio were calculated for the set of seven parameters and the 256 elementary CA, as well as for all possible subsets of the seven parameters. This data is available in Appendix B. Figure 7.8 shows the average intra and inter-cluster distances averaged over all subsets of a given size. As the number of parameters used increases so does both the intra and inter-cluster distance. The increases are nearly proportional resulting in an stable average clustering ratio over all parameter subset sizes.

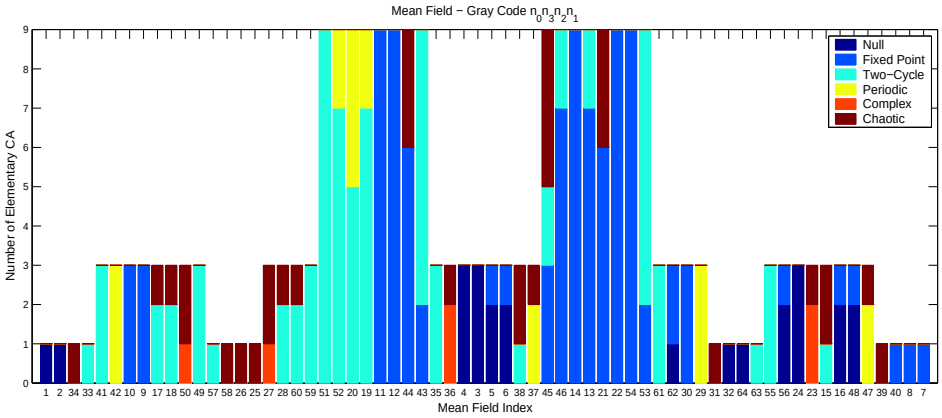
Perhaps an even more critical measure of how easily a set of data points can be classified is the amount of overlap between different classes. If two points, from different classes, have the same positions in parameter space it is impossible to tell for sure which class that point corresponds to. The best classification would be to choose the class with more representative elements at that point. Class overlap was calculated for all possible subsets of parameters, which is also available in Appendix B. Class overlap is defined at the number of pairs of elementary CA rules in different classes with the same set of parameter values. The minimal number of overlaps, two, requires at least five parameters.



(a) Lexicographic order



(b) Reflected-binary Gray code $n_0n_1n_2n_3$



(c) Reflected-binary Gray code $n_0n_3n_2n_1$

Figure 7.7: Number of elementary CA rules in each Li-Packard class for each value of combined mean field parameter. Several orderings of the mean field values are given, including lexicographic and Gray code.

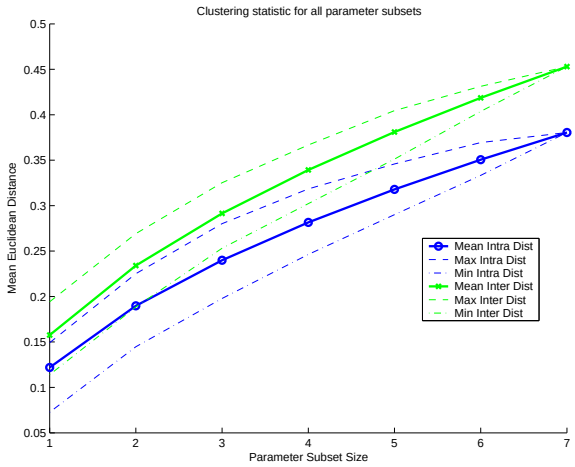


Figure 7.8: Statistics for intra-cluster and inter-cluster distances vs. size of parameter subset averaged over all parameter subsets.

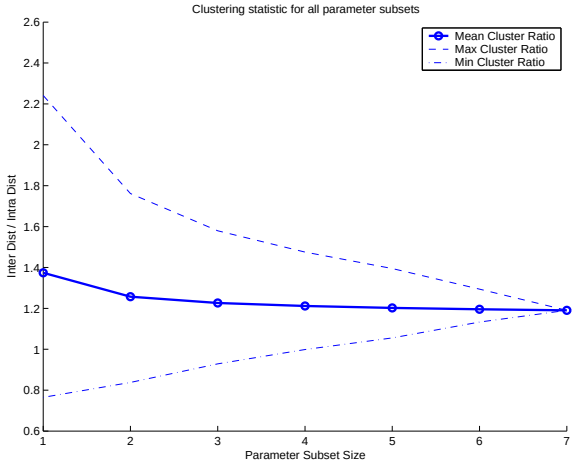


Figure 7.9: Statistics for clustering ratio vs. size of parameter subset averaged over all parameter subsets.

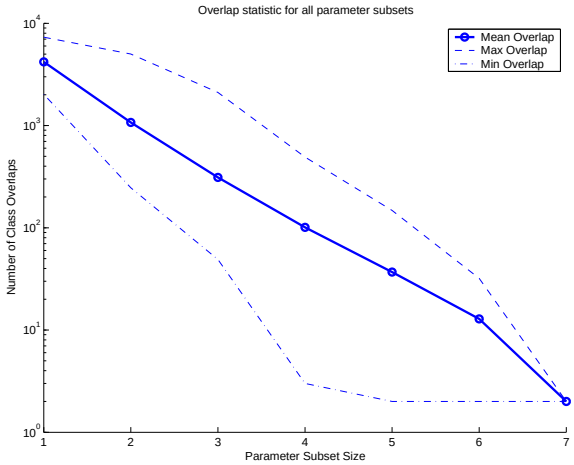


Figure 7.10: Amount of class overlap vs. size of parameter subset averaged over all parameter subsets (note the logarithmic scaling of the y-axis).

There are three sets of five parameters achieving this value, all of which include AA, ND, AP, and υ, and any one of the other three parameters, λ, Z, or μ. Figure 7.10 shows a large decrease in class overlap as the number of parameters is increased.

7.3 Neural Network Error Measure

Because the parameters are used as inputs to a NN for the purpose of classifying CA it is natural to define an efficacy measure based on the error level of those NN. *NN error* is defined as the mean-squared error of a neural network after training to classify the 256 elementary CA into the six Li-Packard classes. Again, this calculation was done for all possible subsets of the seven parameters. The NN error measures given in Appendix B are averaged over five training runs to account for the random weights each NN is initialized with. Lower NN error values are better. The minimum value, 0.0027, was achieved by the set containing λ, AA, ND, AP, and υ, which is also one of the subsets of parameters achieving the minimum class overlap. In fact, class overlap and NN error have a correlation coefficient of 0.76.

7.4 Parameter Ranking

As a method of ranking the seven parameters, the mean values for overlap, clustering, and NN error values were calculated for each parameter over the set of subsets the parameter appears is. Each parameter appears in 64 of the 127

parameter subsets. The table below shows the ranking of the seven parameters, sorted by increasing NN error values.

Rank	Parameter	Overlap	Intra	Inter	Cluster Ratio	NN Error
1	AP	160	0.283	0.328	1.161	0.0215
2	AA	189	0.278	0.334	1.206	0.0235
3	ND	185	0.273	0.334	1.235	0.0240
4	υ	204	0.287	0.325	1.125	0.0297
5	μ	311	0.264	0.335	1.296	0.0304
6	Z	450	0.274	0.345	1.278	0.0322
7	λ	461	0.281	0.341	1.225	0.0326

Again, the strong correlation between the number of overlaps between classes and the resulting error of a trained NN can be seen.

The parameters can also be ranked by how they performed individually. The table below shows efficacy measures as calculated on each single parameter, again ranked by NN error.

Rank	Parameter	Overlap	Intra	Inter	Cluster Ratio	NN Error
1	ND	2022	0.117	0.161	1.373	0.0850
2	AP	2630	0.133	0.131	0.982	0.0950
3	AA	3309	0.132	0.160	1.212	0.0953
4	μ	3415	0.073	0.163	2.241	0.1024
5	Z	7145	0.112	0.194	1.734	0.1071
6	λ	7311	0.137	0.180	1.311	0.1146
7	υ	3557	0.149	0.114	0.765	0.1165

This second ranking is similar to the first. The three parameters developed by Oliveira et al. [26, 24], absolute activity, neighborhood dominance, and activity propagation, hold the first three ranks in both cases. Also, μ, Z, and λ appear in that order in both rankings. The biggest change is in the ranking of υ. It is not nearly as useful alone as it is when included in a set of other parameters. This makes sense as the main purpose of υ is to separate complex rules from all other rules, and not to distinguish between all classes. Also, there are very few complex rules in the set of elementary CA, diminishing the utility of υ. This may also explain why υ is more useful when combined with other parameters, which have more difficulty discriminating between complex and non-complex rules.

Chapter 8

Uses of Automatic Classification

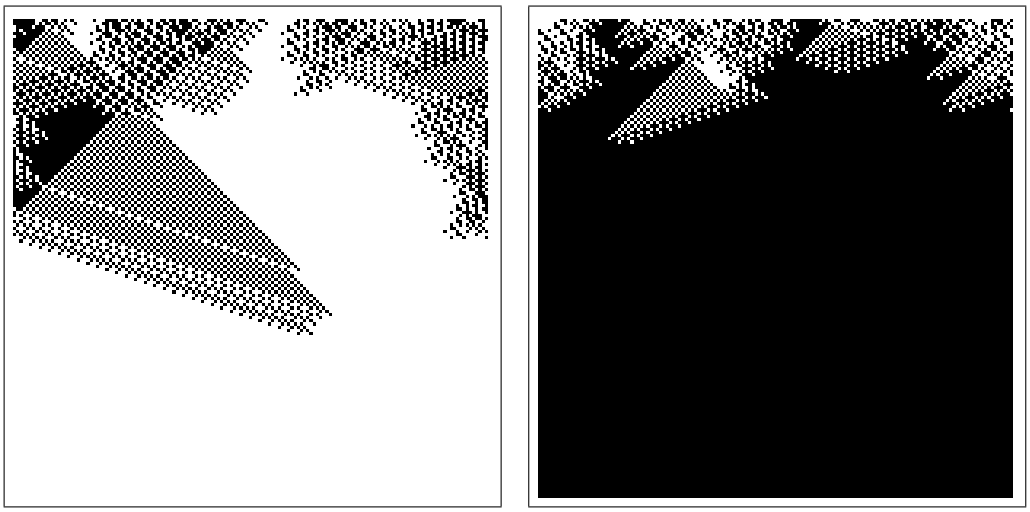
8.1 Searching for Constraint Satisfiers

It is not a trivial matter to construct a cellular automaton (CA) to perform a specific task or satisfy some set of constraints. Their distributed natures makes them difficult to design and the very large number of CA in most rule spaces makes searching for specific behavior time consuming. Two tasks explored in depth by previous work are *density classification* [1, 16, 22, 23, 27] and *synchronization* [7, 22, 29].

The density classification task is to find a CA that will converge to a homogeneous configuration, where all cells have the same state, and where that state is the one with the greatest frequency in the initial condition. For example, Figure 8.1 shows space-time diagrams for one of the best density classifying CA from the literature [16]. The diagrams are 149 cells wide and 150 cells high. The initial conditions have a ratio of 76 white to 73 black and the inverse. The CA quickly converges on a homogeneous state corresponding to the most frequent state in the initial condition.

The synchronization task is to find a CA that, given any initial condition, reaches a two-cycle where the two configurations are homogeneous and of different state. That is, after a sufficient number of steps, all cells will synchronize and "blink" on and off. Figure 8.2 shows a space-time diagram of a CA rule from [22] that performs this task.

The best performing rules for both density classification and synchronization were found using extensive searches in the space of $r = 3$ CA. Improvements in solutions were achieved by using better and faster search techniques, most efforts focusing on the use of genetic algorithms. The search in these cases is guided by a *fitness function* that determines how well a CA accomplishes a given task. If it performs well other CA like it are tried, and so forth. However, the CA that accomplish these tasks are all of a specific dynamic class: density



(a) 76 white, 73 black

(b) 73 white, 76 black

Figure 8.1: Density classification task space-time diagrams.

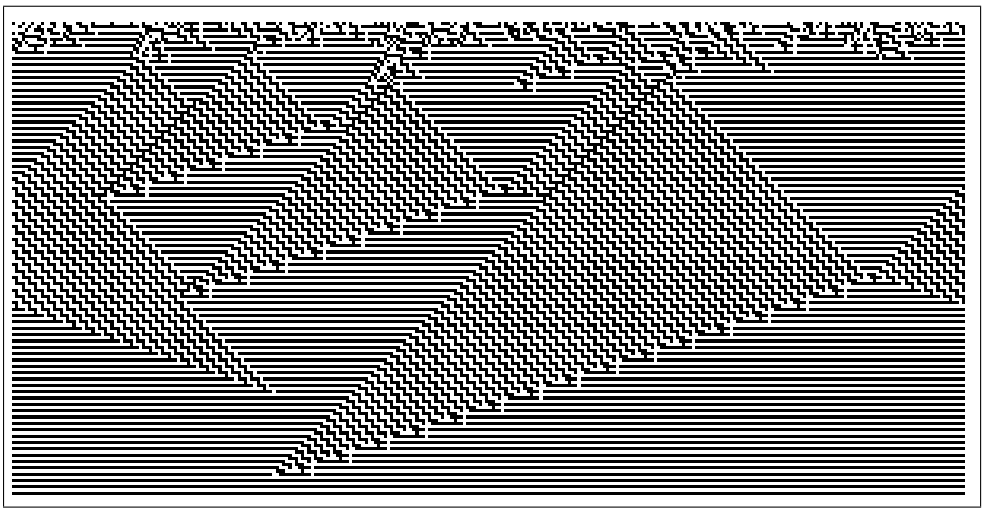


Figure 8.2: Synchronization task space-time diagram.

classifiers are null, synchronizers are two-cycle.

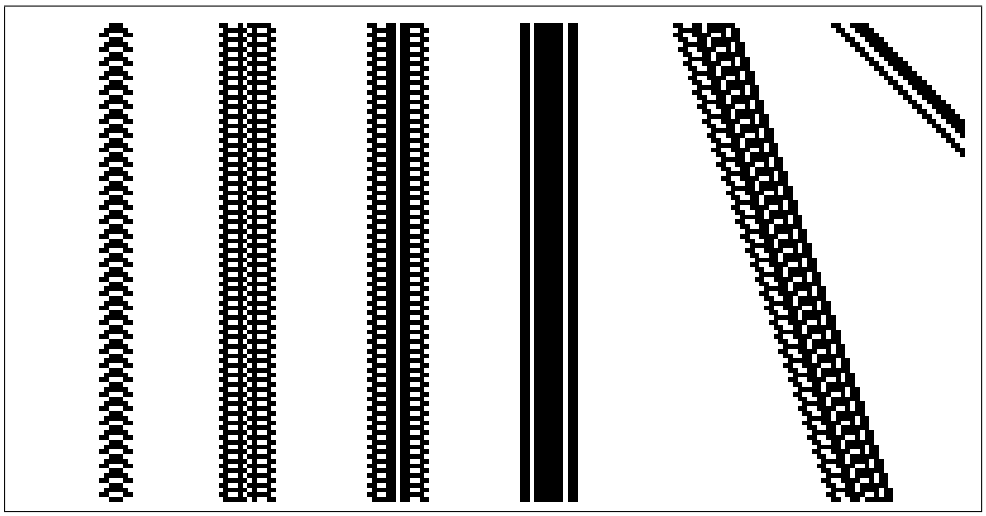
Oliveira et al. used a five parameter set, including Z, μ, AA, ND, and AP, to guide the search for density classifiers and synchronizers [24, 25]. They used a heuristic method based on the observed ranges of parameter values to guide the search toward rules of appropriate behavior. They found parameter-based heuristics improved the search for specific CA significantly. The classification methods presented here should be able to improve the search even further. For example, a NN trained to classify CA into the six Li-Packard classes could be used to guide the search toward null rules for density classification. Even further, the NN could be trained to sort CA rules into two classes: those that perform a given task well and those that do not. The NN could then be used to guide the search toward the class of CA that are known to perform well.

8.2 Searching for Complexity

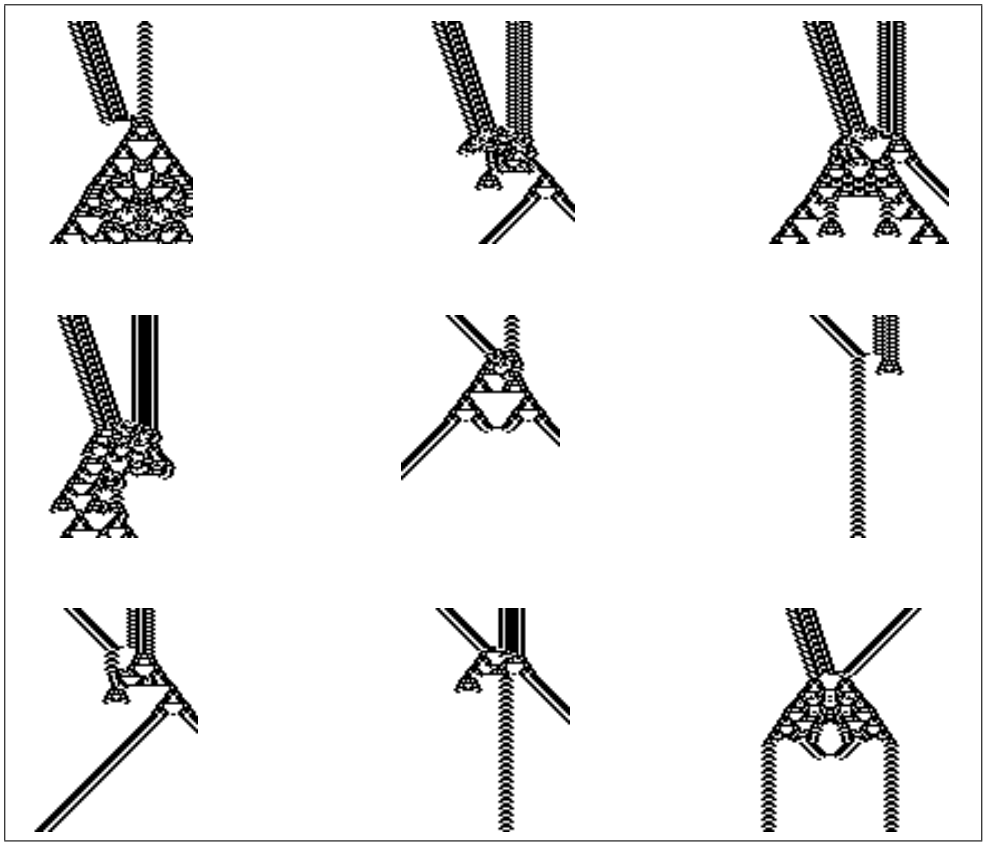
CA with complex dynamics are rare, as seen by their small numbers in both the elementary and totalistic $r = 3$ sets. Along with being rare they are often the most interesting to study. Complex CA have stable structures, often called *gliders* or *particles*, that move through space over time. These structures can collide and produce new particles, allowing for a form of computation to occur. In fact, several of these complex rules have been found to be computationally universal, including the elementary rule 110 [33].

To study these complex CA you first need to find them. The classification methods presented here can aid in this search. Like in finding CA to perform specific tasks, NN methods can be used in two ways to find complex CA: (1) the NN is trained to sort CA into the six Li-Packard classes and those with high complex outputs are studied, or (2) the NN is trained to sort CA into complex and non-complex classes. The second approach is more specific to the task and should be more effective.

Four rules in the totalistic $k = 2$ $r = 3$ set show stable particles: 88, 100, 164, and 216. Figures 8.3, 8.4, 8.5, and 8.6 show these particles and their interactions.

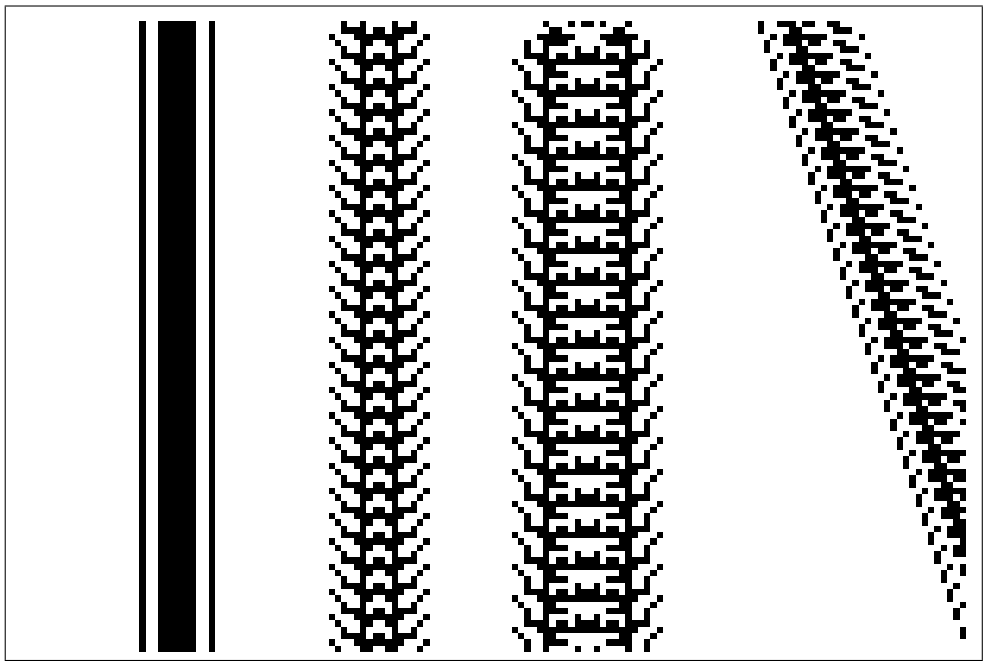


(a) rule 88 particles

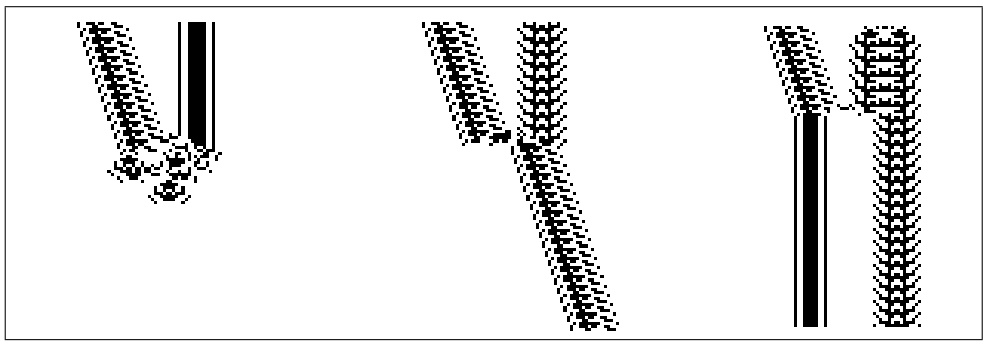


(b) rule 88 particle interaction

Figure 8.3: Particles and interaction for totalistic $k = 2$ $r = 3$ rule 88.

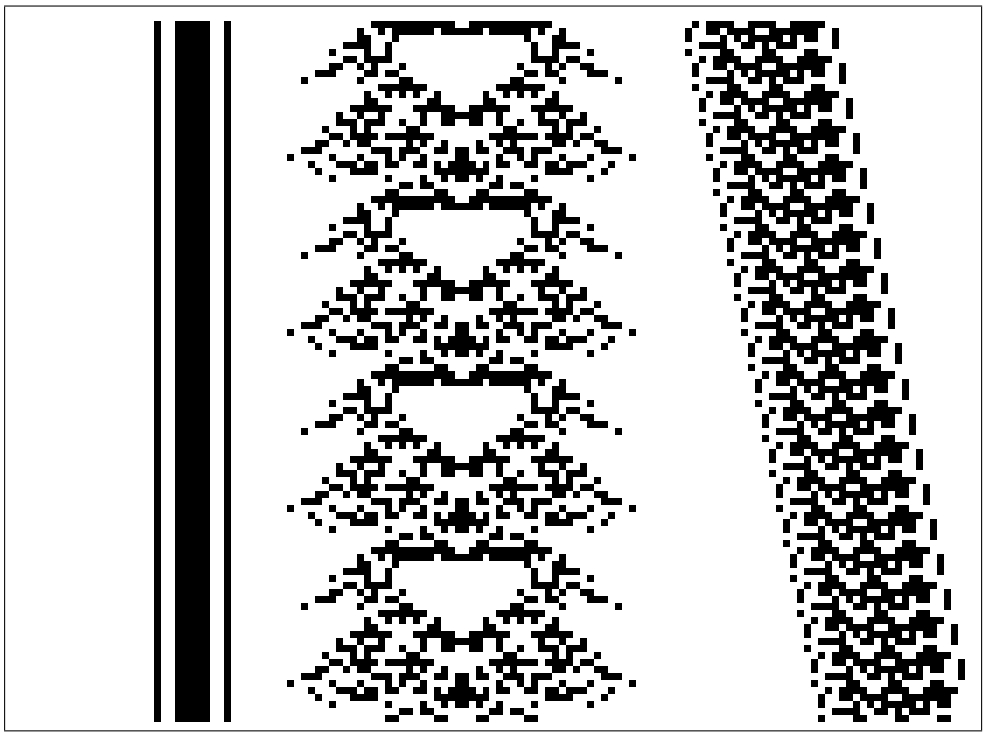


(a) rule 100 particles

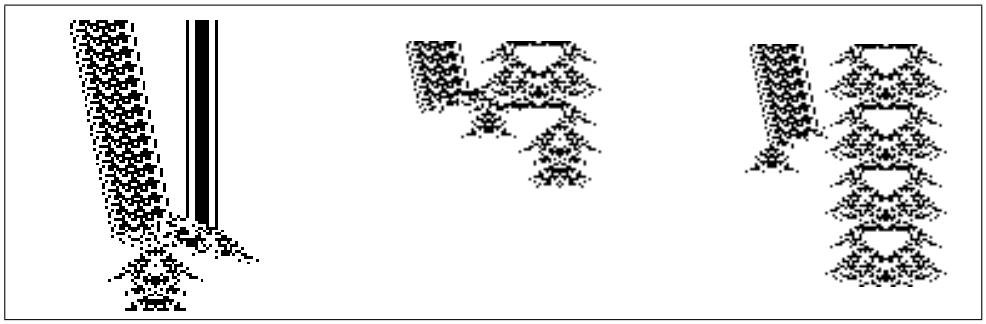


(b) rule 100 particle interaction

Figure 8.4: Particles and interaction for totalistic $k = 2$ $r = 3$ rule 100.

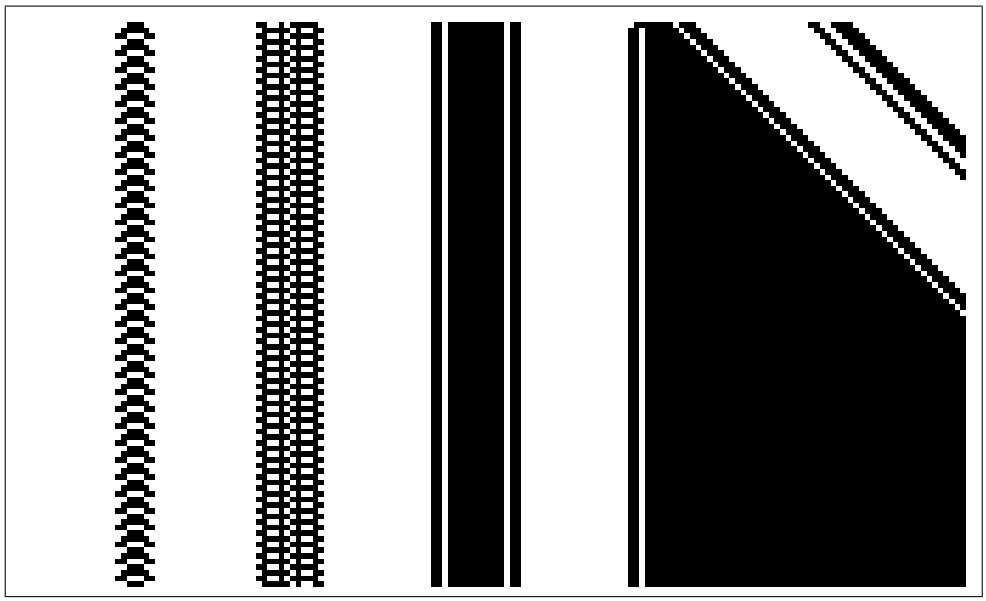


(a) rule 164 particles

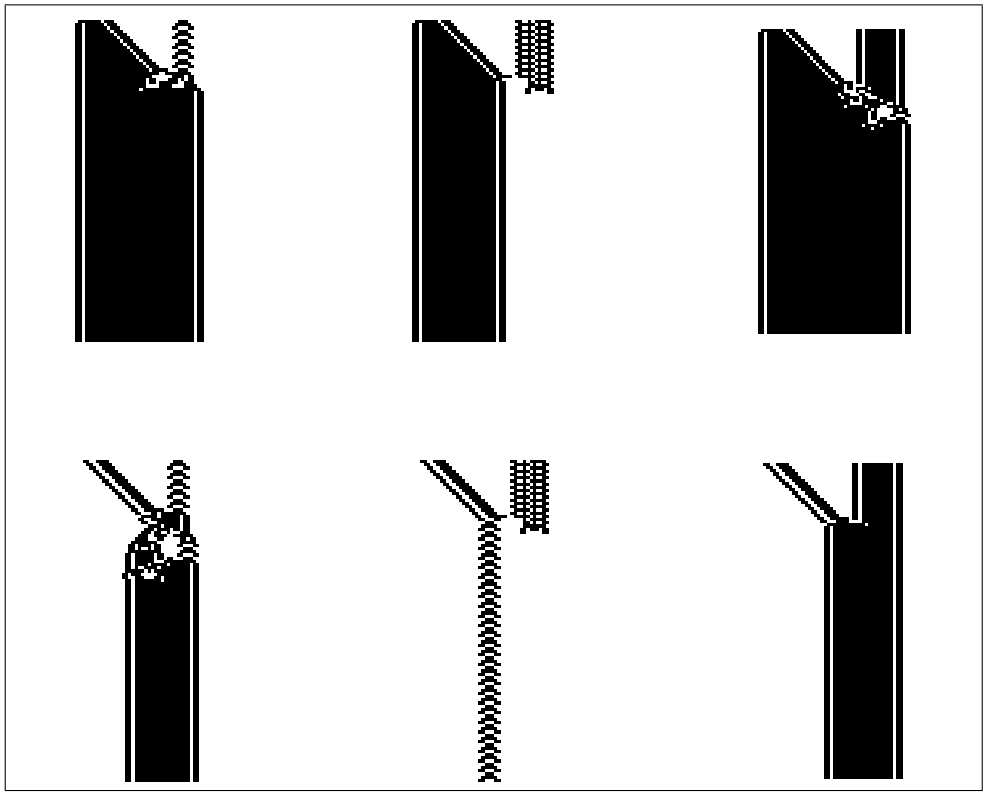


(b) rule 164 particle interaction

Figure 8.5: Particles and interaction for totalistic $k = 2$ $r = 3$ rule 164.



(a) rule 216 particles



(b) rule 216 particle interaction

Figure 8.6: Particles and interaction for totalistic $k = 2$ $r = 3$ rule 216.

8.3 Mapping the CA Rule Space

The first classification parameters were designed to arrive at a better understanding of CA rule space and how the rules and classes of rules related to each other. Langton's λ parameter has supported the notion that complexity occurs "at the edge of chaos", because as λ increases from 0 to 1 the dominant behavior transitions from ordered to complex to chaotic [17].

The classification methods presented here could be used to further understand the space of CA rules. Li, Packard, and Langton stress this use of classification in [20]

> However, *classification alone is not enough!* What is needed is a deeper understanding of the structure of cellular automata rule space, one which provides an explanation for the existence of the observed classes and for their relationship to one another. Choosing an appropriate parameterization of the space of cellular automata rules, such as λ, allows direct observation of the way in which different statistical measures are related as a function of the parameter(s), and these relationships in turn provide an explanation for the existence and ordering of the various qualitatively distinguishable classes of cellular automata dynamics. Once one understands the "deep structure" of cellular automata rule space, it is easy to see that many different classification schemes are possible and equally "correct," depending on which aspects of cellular automata dynamic one is interested in.

Chapter 9

Conclusion

Cellular automata (CA) show a wide range of dynamic behavior. This range of behavior can be organized into six classes, according to the Li-Packard system: null, fixed point, two-cycle, periodic, complex, and chaotic. The original method for classifying CA was manual inspection of space-time diagrams. Then, automatic classification was attempted by quantifying the patterns observed in space-time diagrams. More recently a number of parameterizations of CA rule tables have been proposed as another means for automatically classifying CA. These attempts have primarily focused on calculating parameter values and determining the class of the CA by the range of parameter values observed. A more advanced automatic classification process has been introduced here.

Seven parameters were used for automatic classification here, including those listed below.

	Parameter	**References**
λ	activity	Langton [17]
Z	reverse determinism	Wuensche [34, 35, 36]
μ	sensitivity	Binder [3]
AA	absolute activity	Oliveira et al. [25, 26, 24]
ND	neighborhood dominance	Oliveira et al. [25, 26, 24]
AP	activity propagation	Oliveira et al. [25, 26, 24]
υ	incompressibility	author, Chapter 5.8

These seven parameters were used as inputs to a neural network (NN) that was trained to automatically classify CA into the six classes of the Li-Packard system. The trained NN was able to correctly classify an average of 98.3% of the elementary CA and 93.9% of the totalistic $k = 2$ $r = 3$ CA. These averages were based on 10 NN training trials. In each trial the NN was trained using a random half of CA in the set (either elementary or totalistic) and tested using the other half of the CA. The percent correct each time is based on the number of correctly classified CA in the testing set. Though no other statistics of this nature are published in existing literature, it is believed that the automatic classification method presented here outperform any other to date.

Using the performance of the NN, and other statistical measures including class overlap and clustering ratio, a ranking for the seven parameters is given as follows:

Rank	1	2	3	4	5	6	7
Parameter	AP	AA	ND	υ	μ	Z	λ

A higher rank is given to parameters that are more effective at classifying CA into the six Li-Packard classes.

Possible future work includes improving automatic classification and using automatic classification in other applications. Improvements can be made either in the parameters or in the classification method, in this case neural networks. Some uses include searching for specific, constraint-satisfying CA, searching for and studying complex CA, and mapping the space of CA.

Appendix A

Parameter Values

A.1 Elementary CA

The following table gives the value of each parameter presented in Chapter 5 for each elementary CA rule. λ is an activity parameter, Z is based on a CA reverse algorithm, μ is a sensitivity parameter, AA is absolute activity, ND is neighborhood dominance, AP is activity propagation, and υ is incompressibility (the four mean field parameters are not included).

Rule	Binary	Class	λ	Z	μ	AA	ND	AP	υ
0	00000000	Null	0.00	0.00	0.00	0.50	0.50	0.25	0.00
1	00000001	Two-Cycle	0.25	0.25	0.25	0.75	0.25	0.25	0.11
2	00000010	Fixed-Point	0.25	0.25	0.25	0.62	0.42	0.42	0.33
3	00000011	Two-Cycle	0.50	0.50	0.33	0.88	0.17	0.50	0.44
4	00000100	Fixed-Point	0.25	0.25	0.25	0.50	0.42	0.42	0.22
5	00000101	Two-Cycle	0.50	0.50	0.33	0.75	0.17	0.50	0.11
6	00000110	Two-Cycle	0.50	0.50	0.50	0.62	0.33	0.58	0.56
7	00000111	Two-Cycle	0.75	0.75	0.42	0.88	0.08	0.75	0.44
8	00001000	Null	0.25	0.25	0.25	0.38	0.58	0.17	0.33
9	00001001	Two-Cycle	0.50	0.50	0.50	0.62	0.33	0.17	0.44
10	00001010	Fixed-Point	0.50	0.50	0.33	0.50	0.50	0.25	0.22
11	00001011	Two-Cycle	0.75	0.75	0.42	0.75	0.25	0.33	0.33
12	00001100	Fixed-Point	0.50	0.50	0.33	0.38	0.50	0.25	0.56
13	00001101	Fixed-Point	0.75	0.75	0.42	0.62	0.25	0.33	0.44
14	00001110	Two-Cycle	0.75	0.75	0.42	0.50	0.42	0.33	0.44
15	00001111	Two-Cycle	1.00	1.00	0.33	0.75	0.17	0.50	0.33
16	00010000	Fixed-Point	0.25	0.25	0.25	0.62	0.42	0.42	0.44
17	00010001	Two-Cycle	0.50	0.50	0.33	0.88	0.17	0.50	0.56
18	00010010	Chaotic	0.50	0.50	0.50	0.75	0.33	0.58	0.56
19	00010011	Two-Cycle	0.75	0.62	0.42	1.00	0.08	0.75	0.67
20	00010100	Two-Cycle	0.50	0.50	0.50	0.62	0.33	0.58	0.67
21	00010101	Two-Cycle	0.75	0.75	0.42	0.88	0.08	0.75	0.56
22	00010110	Chaotic	0.75	0.75	0.75	0.75	0.25	0.75	0.78
23	00010111	Two-Cycle	1.00	0.50	0.50	1.00	0.00	1.00	0.67
24	00011000	Fixed-Point	0.50	0.50	0.50	0.50	0.50	0.33	0.56
25	00011001	Two-Cycle	0.75	0.75	0.58	0.75	0.25	0.42	0.67
26	00011010	Periodic	0.75	0.75	0.58	0.62	0.42	0.42	0.22

Rule	Binary	Class	λ	Z	μ	AA	ND	AP	v
27	00011011	Two-Cycle	1.00	0.75	0.50	0.88	0.17	0.58	0.33
28	00011100	Two-Cycle	0.75	0.75	0.58	0.50	0.42	0.42	0.78
29	00011101	Two-Cycle	1.00	0.50	0.50	0.75	0.17	0.58	0.67
30	00011110	Chaotic	1.00	1.00	0.67	0.62	0.33	0.50	0.44
31	00011111	Two-Cycle	0.75	0.75	0.42	0.88	0.08	0.75	0.33
32	00100000	Null	0.25	0.25	0.25	0.50	0.58	0.17	0.22
33	00100001	Two-Cycle	0.50	0.50	0.50	0.75	0.33	0.17	0.33
34	00100010	Fixed-Point	0.50	0.50	0.33	0.62	0.50	0.25	0.56
35	00100011	Two-Cycle	0.75	0.62	0.42	0.88	0.25	0.33	0.67
36	00100100	Fixed-Point	0.50	0.50	0.50	0.50	0.50	0.33	0.22
37	00100101	Two-Cycle	0.75	0.75	0.58	0.75	0.25	0.42	0.11
38	00100110	Two-Cycle	0.75	0.75	0.58	0.62	0.42	0.42	0.56
39	00100111	Two-Cycle	1.00	0.75	0.50	0.88	0.17	0.58	0.44
40	00101000	Null	0.50	0.50	0.50	0.38	0.67	0.08	0.56
41	00101001	Periodic	0.75	0.75	0.75	0.62	0.42	0.08	0.67
42	00101010	Fixed-Point	0.75	0.75	0.42	0.50	0.58	0.08	0.44
43	00101011	Two-Cycle	1.00	0.50	0.50	0.75	0.33	0.17	0.56
44	00101100	Fixed-Point	0.75	0.75	0.58	0.38	0.58	0.17	0.56
45	00101101	Chaotic	1.00	1.00	0.67	0.62	0.33	0.25	0.44
46	00101110	Fixed-Point	1.00	0.50	0.50	0.50	0.50	0.17	0.44
47	00101111	Two-Cycle	0.75	0.75	0.42	0.75	0.25	0.33	0.33
48	00110000	Fixed-Point	0.50	0.50	0.33	0.62	0.50	0.25	0.67
49	00110001	Two-Cycle	0.75	0.62	0.42	0.88	0.25	0.33	0.78
50	00110010	Two-Cycle	0.75	0.62	0.42	0.75	0.42	0.33	0.78
51	00110011	Two-Cycle	1.00	1.00	0.33	1.00	0.17	0.50	0.89
52	00110100	Two-Cycle	0.75	0.75	0.58	0.62	0.42	0.42	0.67
53	00110101	Two-Cycle	1.00	0.75	0.50	0.88	0.17	0.58	0.56
54	00110110	Complex	1.00	0.75	0.67	0.75	0.33	0.50	0.78
55	00110111	Two-Cycle	0.75	0.62	0.42	1.00	0.08	0.75	0.67
56	00111000	Fixed-Point	0.75	0.75	0.58	0.50	0.58	0.17	0.78
57	00111001	Fixed-Point	1.00	0.75	0.67	0.75	0.33	0.25	0.89
58	00111010	Fixed-Point	1.00	0.75	0.50	0.62	0.50	0.17	0.44
59	00111011	Two-Cycle	0.75	0.62	0.42	0.88	0.25	0.33	0.56
60	00111100	Chaotic	1.00	1.00	0.67	0.50	0.50	0.25	0.78
61	00111101	Two-Cycle	0.75	0.75	0.58	0.75	0.25	0.42	0.67
62	00111110	Periodic	0.75	0.75	0.58	0.62	0.42	0.25	0.44
63	00111111	Two-Cycle	0.50	0.50	0.33	0.88	0.17	0.50	0.33
64	01000000	Null	0.25	0.25	0.25	0.38	0.58	0.17	0.22
65	01000001	Two-Cycle	0.50	0.50	0.50	0.62	0.33	0.17	0.33
66	01000010	Fixed-Point	0.50	0.50	0.50	0.50	0.50	0.33	0.56
67	01000011	Two-Cycle	0.75	0.75	0.58	0.75	0.25	0.42	0.67
68	01000100	Fixed-Point	0.50	0.50	0.33	0.38	0.50	0.25	0.44
69	01000101	Fixed-Point	0.75	0.75	0.42	0.62	0.25	0.33	0.33
70	01000110	Two-Cycle	0.75	0.75	0.58	0.50	0.42	0.42	0.78
71	01000111	Two-Cycle	1.00	0.50	0.50	0.75	0.17	0.58	0.67
72	01001000	Fixed-Point	0.50	0.50	0.50	0.25	0.67	0.08	0.56
73	01001001	Chaotic	0.75	0.75	0.75	0.50	0.42	0.08	0.67
74	01001010	Two-Cycle	0.75	0.75	0.58	0.38	0.58	0.17	0.44
75	01001011	Chaotic	1.00	1.00	0.67	0.62	0.33	0.25	0.56
76	01001100	Fixed-Point	0.75	0.62	0.42	0.25	0.58	0.08	0.78
77	01001101	Fixed-Point	1.00	0.50	0.50	0.50	0.33	0.17	0.67
78	01001110	Fixed-Point	1.00	0.75	0.50	0.38	0.50	0.17	0.67
79	01001111	Fixed-Point	0.75	0.75	0.42	0.62	0.25	0.33	0.56
80	01010000	Fixed-Point	0.50	0.50	0.33	0.50	0.50	0.25	0.22
81	01010001	Two-Cycle	0.75	0.75	0.42	0.75	0.25	0.33	0.33
82	01010010	Periodic	0.75	0.75	0.58	0.62	0.42	0.42	0.33
83	01010011	Two-Cycle	1.00	0.75	0.50	0.88	0.17	0.58	0.44
84	01010100	Two-Cycle	0.75	0.75	0.42	0.50	0.42	0.33	0.44
85	01010101	Two-Cycle	1.00	1.00	0.33	0.75	0.17	0.50	0.33

Rule	Binary	Class	λ	Z	μ	AA	ND	AP	v
86	01010110	Chaotic	1.00	1.00	0.67	0.62	0.33	0.50	0.56
87	01010111	Two-Cycle	0.75	0.75	0.42	0.88	0.08	0.75	0.44
88	01011000	Two-Cycle	0.75	0.75	0.58	0.38	0.58	0.17	0.33
89	01011001	Chaotic	1.00	1.00	0.67	0.62	0.33	0.25	0.44
90	01011010	Chaotic	1.00	1.00	0.67	0.50	0.50	0.25	0.00
91	01011011	Two-Cycle	0.75	0.75	0.58	0.75	0.25	0.42	0.11
92	01011100	Fixed-Point	1.00	0.75	0.50	0.38	0.50	0.17	0.56
93	01011101	Fixed-Point	0.75	0.75	0.42	0.62	0.25	0.33	0.44
94	01011110	Periodic	0.75	0.75	0.58	0.50	0.42	0.25	0.22
95	01011111	Two-Cycle	0.50	0.50	0.33	0.75	0.17	0.50	0.11
96	01100000	Null	0.50	0.50	0.50	0.38	0.67	0.08	0.44
97	01100001	Periodic	0.75	0.75	0.75	0.62	0.42	0.08	0.56
98	01100010	Fixed-Point	0.75	0.75	0.58	0.50	0.58	0.17	0.78
99	01100011	Fixed-Point	1.00	0.75	0.67	0.75	0.33	0.25	0.89
100	01100100	Fixed-Point	0.75	0.75	0.58	0.38	0.58	0.17	0.44
101	01100101	Chaotic	1.00	1.00	0.67	0.62	0.33	0.25	0.33
102	01100110	Chaotic	1.00	1.00	0.67	0.50	0.50	0.25	0.78
103	01100111	Two-Cycle	0.75	0.75	0.58	0.75	0.25	0.42	0.67
104	01101000	Fixed-Point	0.75	0.75	0.75	0.25	0.75	0.00	0.78
105	01101001	Chaotic	1.00	1.00	1.00	0.50	0.50	0.00	0.89
106	01101010	Chaotic	1.00	1.00	0.67	0.38	0.67	0.00	0.67
107	01101011	Periodic	0.75	0.75	0.75	0.62	0.42	0.08	0.78
108	01101100	Two-Cycle	1.00	0.75	0.67	0.25	0.67	0.00	0.78
109	01101101	Chaotic	0.75	0.75	0.75	0.50	0.42	0.08	0.67
110	01101110	Complex	0.75	0.75	0.58	0.38	0.58	0.00	0.67
111	01101111	Two-Cycle	0.50	0.50	0.50	0.62	0.33	0.17	0.56
112	01110000	Fixed-Point	0.75	0.75	0.42	0.50	0.58	0.08	0.44
113	01110001	Two-Cycle	1.00	0.50	0.50	0.75	0.33	0.17	0.56
114	01110010	Fixed-Point	1.00	0.75	0.50	0.62	0.50	0.17	0.56
115	01110011	Two-Cycle	0.75	0.62	0.42	0.88	0.25	0.33	0.67
116	01110100	Fixed-Point	1.00	0.50	0.50	0.50	0.50	0.17	0.44
117	01110101	Two-Cycle	0.75	0.75	0.42	0.75	0.25	0.33	0.33
118	01110110	Periodic	0.75	0.75	0.58	0.62	0.42	0.25	0.56
119	01110111	Two-Cycle	0.50	0.50	0.33	0.88	0.17	0.50	0.44
120	01111000	Chaotic	1.00	1.00	0.67	0.38	0.67	0.00	0.56
121	01111001	Periodic	0.75	0.75	0.75	0.62	0.42	0.08	0.67
122	01111010	Chaotic	0.75	0.75	0.58	0.50	0.58	0.00	0.22
123	01111011	Two-Cycle	0.50	0.50	0.50	0.75	0.33	0.17	0.33
124	01111100	Complex	0.75	0.75	0.58	0.38	0.58	0.00	0.56
125	01111101	Two-Cycle	0.50	0.50	0.50	0.62	0.33	0.17	0.44
126	01111110	Chaotic	0.50	0.50	0.50	0.50	0.50	0.00	0.22
127	01111111	Two-Cycle	0.25	0.25	0.25	0.75	0.25	0.25	0.11
128	10000000	Null	0.25	0.25	0.25	0.25	0.75	0.00	0.11
129	10000001	Chaotic	0.50	0.50	0.50	0.50	0.50	0.00	0.22
130	10000010	Fixed-Point	0.50	0.50	0.50	0.38	0.67	0.17	0.44
131	10000011	Periodic	0.75	0.75	0.58	0.62	0.42	0.25	0.56
132	10000100	Fixed-Point	0.50	0.50	0.50	0.25	0.67	0.17	0.33
133	10000101	Periodic	0.75	0.75	0.58	0.50	0.42	0.25	0.22
134	10000110	Two-Cycle	0.75	0.75	0.75	0.38	0.58	0.33	0.67
135	10000111	Chaotic	1.00	1.00	0.67	0.62	0.33	0.50	0.56
136	10001000	Null	0.50	0.50	0.33	0.12	0.83	0.00	0.44
137	10001001	Complex	0.75	0.75	0.58	0.38	0.58	0.00	0.56
138	10001010	Fixed-Point	0.75	0.75	0.42	0.25	0.75	0.08	0.33
139	10001011	Fixed-Point	1.00	0.50	0.50	0.50	0.50	0.17	0.44
140	10001100	Fixed-Point	0.75	0.62	0.42	0.12	0.75	0.08	0.67
141	10001101	Fixed-Point	1.00	0.75	0.50	0.38	0.50	0.17	0.56
142	10001110	Two-Cycle	1.00	0.50	0.50	0.25	0.67	0.17	0.56
143	10001111	Two-Cycle	0.75	0.75	0.42	0.50	0.42	0.33	0.44
144	10010000	Fixed-Point	0.50	0.50	0.50	0.38	0.67	0.17	0.56

Rule	Binary	Class	λ	Z	μ	AA	ND	AP	v
145	10010001	Periodic	0.75	0.75	0.58	0.62	0.42	0.25	0.67
146	10010010	Chaotic	0.75	0.75	0.75	0.50	0.58	0.33	0.67
147	10010011	Complex	1.00	0.75	0.67	0.75	0.33	0.50	0.78
148	10010100	Two-Cycle	0.75	0.75	0.75	0.38	0.58	0.33	0.78
149	10010101	Chaotic	1.00	1.00	0.67	0.62	0.33	0.50	0.67
150	10010110	Chaotic	1.00	1.00	1.00	0.50	0.50	0.50	0.89
151	10010111	Chaotic	0.75	0.75	0.75	0.75	0.25	0.75	0.78
152	10011000	Fixed-Point	0.75	0.75	0.58	0.25	0.75	0.17	0.67
153	10011001	Chaotic	1.00	1.00	0.67	0.50	0.50	0.25	0.78
154	10011010	Periodic	1.00	1.00	0.67	0.38	0.67	0.25	0.33
155	10011011	Two-Cycle	0.75	0.75	0.58	0.62	0.42	0.42	0.44
156	10011100	Two-Cycle	1.00	0.75	0.67	0.25	0.67	0.25	0.89
157	10011101	Two-Cycle	0.75	0.75	0.58	0.50	0.42	0.42	0.78
158	10011110	Two-Cycle	0.75	0.75	0.75	0.38	0.58	0.33	0.56
159	10011111	Two-Cycle	0.50	0.50	0.50	0.62	0.33	0.58	0.44
160	10100000	Null	0.50	0.50	0.33	0.25	0.83	0.00	0.11
161	10100001	Chaotic	0.75	0.75	0.58	0.50	0.58	0.00	0.22
162	10100010	Fixed-Point	0.75	0.75	0.42	0.38	0.75	0.08	0.44
163	10100011	Fixed-Point	1.00	0.75	0.50	0.62	0.50	0.17	0.56
164	10100100	Fixed-Point	0.75	0.75	0.58	0.25	0.75	0.17	0.11
165	10100101	Chaotic	1.00	1.00	0.67	0.50	0.50	0.25	0.00
166	10100110	Periodic	1.00	1.00	0.67	0.38	0.67	0.25	0.44
167	10100111	Periodic	0.75	0.75	0.58	0.62	0.42	0.42	0.33
168	10101000	Null	0.75	0.75	0.42	0.12	0.92	0.00	0.44
169	10101001	Chaotic	1.00	1.00	0.67	0.38	0.67	0.00	0.56
170	10101010	Fixed-Point	1.00	1.00	0.33	0.25	0.83	0.00	0.33
171	10101011	Fixed-Point	0.75	0.75	0.42	0.50	0.58	0.08	0.44
172	10101100	Fixed-Point	1.00	0.75	0.50	0.12	0.83	0.08	0.44
173	10101101	Two-Cycle	0.75	0.75	0.58	0.38	0.58	0.17	0.33
174	10101110	Fixed-Point	0.75	0.75	0.42	0.25	0.75	0.08	0.33
175	10101111	Fixed-Point	0.50	0.50	0.33	0.50	0.50	0.25	0.22
176	10110000	Fixed-Point	0.75	0.75	0.42	0.38	0.75	0.08	0.56
177	10110001	Fixed-Point	1.00	0.75	0.50	0.62	0.50	0.17	0.67
178	10110010	Two-Cycle	1.00	0.50	0.50	0.50	0.67	0.17	0.67
179	10110011	Two-Cycle	0.75	0.62	0.42	0.75	0.42	0.33	0.78
180	10110100	Periodic	1.00	1.00	0.67	0.38	0.67	0.25	0.56
181	10110101	Periodic	0.75	0.75	0.58	0.62	0.42	0.42	0.44
182	10110110	Chaotic	0.75	0.75	0.75	0.50	0.58	0.33	0.67
183	10110111	Chaotic	0.50	0.50	0.50	0.75	0.33	0.58	0.56
184	10111000	Fixed-Point	1.00	0.50	0.50	0.25	0.83	0.08	0.67
185	10111001	Fixed-Point	0.75	0.75	0.58	0.50	0.58	0.17	0.78
186	10111010	Fixed-Point	0.75	0.75	0.42	0.38	0.75	0.08	0.33
187	10111011	Fixed-Point	0.50	0.50	0.33	0.62	0.50	0.25	0.44
188	10111100	Fixed-Point	0.75	0.75	0.58	0.25	0.75	0.17	0.67
189	10111101	Fixed-Point	0.50	0.50	0.50	0.50	0.50	0.33	0.56
190	10111110	Fixed-Point	0.50	0.50	0.50	0.38	0.67	0.17	0.33
191	10111111	Fixed-Point	0.25	0.25	0.25	0.62	0.42	0.42	0.22
192	11000000	Null	0.50	0.50	0.33	0.12	0.83	0.00	0.33
193	11000001	Complex	0.75	0.75	0.58	0.38	0.58	0.00	0.44
194	11000010	Fixed-Point	0.75	0.75	0.58	0.25	0.75	0.17	0.67
195	11000011	Chaotic	1.00	1.00	0.67	0.50	0.50	0.25	0.78
196	11000100	Fixed-Point	0.75	0.62	0.42	0.12	0.75	0.08	0.56
197	11000101	Fixed-Point	1.00	0.75	0.50	0.38	0.50	0.17	0.44
198	11000110	Two-Cycle	1.00	0.75	0.67	0.25	0.67	0.25	0.89
199	11000111	Two-Cycle	0.75	0.75	0.58	0.50	0.42	0.42	0.78
200	11001000	Fixed-Point	0.75	0.62	0.42	0.00	0.92	0.00	0.67
201	11001001	Two-Cycle	1.00	0.75	0.67	0.25	0.67	0.00	0.78
202	11001010	Fixed-Point	1.00	0.75	0.50	0.12	0.83	0.08	0.56
203	11001011	Fixed-Point	0.75	0.75	0.58	0.38	0.58	0.17	0.67

Rule	Binary	Class	λ	Z	μ	AA	ND	AP	v
204	11001100	Fixed-Point	1.00	1.00	0.33	0.00	0.83	0.00	0.89
205	11001101	Fixed-Point	0.75	0.62	0.42	0.25	0.58	0.08	0.78
206	11001110	Fixed-Point	0.75	0.62	0.42	0.12	0.75	0.08	0.78
207	11001111	Fixed-Point	0.50	0.50	0.33	0.38	0.50	0.25	0.67
208	11010000	Fixed-Point	0.75	0.75	0.42	0.25	0.75	0.08	0.33
209	11010001	Fixed-Point	1.00	0.50	0.50	0.50	0.50	0.17	0.44
210	11010010	Periodic	1.00	1.00	0.67	0.38	0.67	0.25	0.44
211	11010011	Two-Cycle	0.75	0.75	0.58	0.62	0.42	0.42	0.56
212	11010100	Two-Cycle	1.00	0.50	0.50	0.25	0.67	0.17	0.56
213	11010101	Two-Cycle	0.75	0.75	0.42	0.50	0.42	0.33	0.44
214	11010110	Two-Cycle	0.75	0.75	0.75	0.38	0.58	0.33	0.67
215	11010111	Two-Cycle	0.50	0.50	0.50	0.62	0.33	0.58	0.56
216	11011000	Fixed-Point	1.00	0.75	0.50	0.12	0.83	0.08	0.44
217	11011001	Fixed-Point	0.75	0.75	0.58	0.38	0.58	0.17	0.56
218	11011010	Fixed-Point	0.75	0.75	0.58	0.25	0.75	0.17	0.11
219	11011011	Fixed-Point	0.50	0.50	0.50	0.50	0.50	0.33	0.22
220	11011100	Fixed-Point	0.75	0.62	0.42	0.12	0.75	0.08	0.67
221	11011101	Fixed-Point	0.50	0.50	0.33	0.38	0.50	0.25	0.56
222	11011110	Fixed-Point	0.50	0.50	0.50	0.25	0.67	0.17	0.33
223	11011111	Fixed-Point	0.25	0.25	0.25	0.50	0.42	0.42	0.22
224	11100000	Null	0.75	0.75	0.42	0.12	0.92	0.00	0.33
225	11100001	Chaotic	1.00	1.00	0.67	0.38	0.67	0.00	0.44
226	11100010	Fixed-Point	1.00	0.50	0.50	0.25	0.83	0.08	0.67
227	11100011	Fixed-Point	0.75	0.75	0.58	0.50	0.58	0.17	0.78
228	11100100	Fixed-Point	1.00	0.75	0.50	0.12	0.83	0.08	0.33
229	11100101	Two-Cycle	0.75	0.75	0.58	0.38	0.58	0.17	0.22
230	11100110	Fixed-Point	0.75	0.75	0.58	0.25	0.75	0.17	0.67
231	11100111	Fixed-Point	0.50	0.50	0.50	0.50	0.50	0.33	0.56
232	11101000	Fixed-Point	1.00	0.50	0.50	0.00	1.00	0.00	0.67
233	11101001	Fixed-Point	0.75	0.75	0.75	0.25	0.75	0.00	0.78
234	11101010	Null	0.75	0.75	0.42	0.12	0.92	0.00	0.56
235	11101011	Null	0.50	0.50	0.50	0.38	0.67	0.08	0.67
236	11101100	Fixed-Point	0.75	0.62	0.42	0.00	0.92	0.00	0.67
237	11101101	Fixed-Point	0.50	0.50	0.50	0.25	0.67	0.08	0.56
238	11101110	Null	0.50	0.50	0.33	0.12	0.83	0.00	0.56
239	11101111	Null	0.25	0.25	0.25	0.38	0.58	0.17	0.44
240	11110000	Fixed-Point	1.00	1.00	0.33	0.25	0.83	0.00	0.33
241	11110001	Fixed-Point	0.75	0.75	0.42	0.50	0.58	0.08	0.44
242	11110010	Fixed-Point	0.75	0.75	0.42	0.38	0.75	0.08	0.44
243	11110011	Fixed-Point	0.50	0.50	0.33	0.62	0.50	0.25	0.56
244	11110100	Fixed-Point	0.75	0.75	0.42	0.25	0.75	0.08	0.33
245	11110101	Fixed-Point	0.50	0.50	0.33	0.50	0.50	0.25	0.22
246	11110110	Fixed-Point	0.50	0.50	0.50	0.38	0.67	0.17	0.44
247	11110111	Fixed-Point	0.25	0.25	0.25	0.62	0.42	0.42	0.33
248	11111000	Null	0.75	0.75	0.42	0.12	0.92	0.00	0.44
249	11111001	Null	0.50	0.50	0.50	0.38	0.67	0.08	0.56
250	11111010	Null	0.50	0.50	0.33	0.25	0.83	0.00	0.11
251	11111011	Null	0.25	0.25	0.25	0.50	0.58	0.17	0.22
252	11111100	Null	0.50	0.50	0.33	0.12	0.83	0.00	0.44
253	11111101	Null	0.25	0.25	0.25	0.38	0.58	0.17	0.33
254	11111110	Null	0.25	0.25	0.25	0.25	0.75	0.00	0.11
255	11111111	Null	0.00	0.00	0.00	0.50	0.50	0.25	0.00

A.2 Totalistic $k = 2$ $r = 3$ CA

The following table gives the value of each parameter presented in Chapter 5 for each totalistic CA rule with 2 states and a radius of 3 (neighborhood size 7). λ is an activity parameter, Z is based on a CA reverse algorithm, μ is a sensitivity parameter, AA is absolute activity, ND is neighborhood dominance, and AP is activity propagation, and υ is incompressibility (the seven mean field parameters are not included).

Rule	Binary	λ	Z	μ	AA	ND	AP	υ
0	00000000	0.00	0.00	0.00	0.50	0.50	0.34	0.00
1	00000001	0.02	0.02	0.02	0.53	0.45	0.34	0.00
2	00000010	0.11	0.11	0.11	0.62	0.30	0.34	0.09
3	00000011	0.12	0.09	0.09	0.65	0.25	0.36	0.08
4	00000100	0.33	0.33	0.33	0.72	0.30	0.34	0.25
5	00000101	0.34	0.34	0.34	0.75	0.25	0.34	0.26
6	00000110	0.44	0.25	0.25	0.85	0.10	0.44	0.21
7	00000111	0.45	0.23	0.23	0.88	0.05	0.45	0.20
8	00001000	0.55	0.55	0.55	0.62	0.45	0.66	0.33
9	00001001	0.56	0.56	0.56	0.65	0.40	0.66	0.33
10	00001010	0.66	0.66	0.66	0.75	0.25	0.66	0.42
11	00001011	0.67	0.64	0.64	0.78	0.20	0.67	0.41
12	00001100	0.88	0.41	0.41	0.85	0.25	0.89	0.29
13	00001101	0.89	0.42	0.42	0.88	0.20	0.89	0.29
14	00001110	0.98	0.33	0.33	0.97	0.05	0.98	0.24
15	00001111	1.00	0.31	0.31	1.00	0.00	1.00	0.24
16	00010000	0.55	0.55	0.55	0.38	0.55	0.11	0.33
17	00010001	0.56	0.56	0.56	0.40	0.50	0.11	0.33
18	00010010	0.66	0.66	0.66	0.50	0.35	0.11	0.39
19	00010011	0.67	0.64	0.64	0.53	0.30	0.12	0.39
20	00010100	0.88	0.88	0.88	0.60	0.35	0.11	0.55
21	00010101	0.89	0.89	0.89	0.62	0.30	0.11	0.55
22	00010110	0.98	0.80	0.80	0.72	0.14	0.20	0.48
23	00010111	1.00	0.78	0.78	0.75	0.10	0.22	0.47
24	00011000	0.91	0.47	0.47	0.50	0.50	0.11	0.34
25	00011001	0.89	0.48	0.48	0.53	0.45	0.11	0.34
26	00011010	0.80	0.58	0.58	0.62	0.30	0.11	0.40
27	00011011	0.78	0.56	0.56	0.65	0.25	0.12	0.40
28	00011100	0.58	0.33	0.33	0.72	0.30	0.34	0.26
29	00011101	0.56	0.34	0.34	0.75	0.25	0.34	0.27
30	00011110	0.47	0.25	0.25	0.85	0.10	0.44	0.19
31	00011111	0.45	0.23	0.23	0.88	0.05	0.45	0.19
32	00100000	0.33	0.33	0.33	0.28	0.70	0.02	0.24
33	00100001	0.34	0.34	0.34	0.30	0.65	0.02	0.24
34	00100010	0.44	0.44	0.44	0.40	0.50	0.02	0.32
35	00100011	0.45	0.42	0.42	0.42	0.45	0.03	0.32
36	00100100	0.66	0.66	0.66	0.50	0.50	0.02	0.45
37	00100101	0.67	0.67	0.67	0.53	0.45	0.02	0.45
38	00100110	0.77	0.58	0.58	0.62	0.30	0.11	0.40
39	00100111	0.78	0.56	0.56	0.65	0.25	0.12	0.40
40	00101000	0.88	0.88	0.88	0.40	0.65	0.33	0.53
41	00101001	0.89	0.89	0.89	0.42	0.61	0.33	0.54
42	00101010	0.98	0.98	0.98	0.53	0.45	0.33	0.62
43	00101011	1.00	0.97	0.97	0.55	0.40	0.34	0.62
44	00101100	0.80	0.73	0.73	0.62	0.45	0.56	0.45
45	00101101	0.78	0.75	0.75	0.65	0.40	0.56	0.45

Rule	Binary	λ	Z	μ	AA	ND	AP	υ
46	00101110	0.69	0.66	0.66	0.75	0.25	0.66	0.40
47	00101111	0.67	0.64	0.64	0.78	0.20	0.67	0.40
48	00110000	0.88	0.41	0.41	0.15	0.75	0.02	0.29
49	00110001	0.89	0.42	0.42	0.17	0.70	0.02	0.29
50	00110010	0.98	0.52	0.52	0.28	0.55	0.02	0.35
51	00110011	1.00	0.50	0.50	0.30	0.50	0.03	0.35
52	00110100	0.80	0.73	0.73	0.38	0.55	0.02	0.46
53	00110101	0.78	0.75	0.75	0.40	0.50	0.02	0.47
54	00110110	0.69	0.66	0.66	0.50	0.35	0.11	0.39
55	00110111	0.67	0.64	0.64	0.53	0.30	0.12	0.39
56	00111000	0.58	0.33	0.33	0.28	0.70	0.02	0.26
57	00111001	0.56	0.34	0.34	0.30	0.65	0.02	0.27
58	00111010	0.47	0.44	0.44	0.40	0.50	0.02	0.32
59	00111011	0.45	0.42	0.42	0.42	0.45	0.03	0.32
60	00111100	0.25	0.19	0.19	0.50	0.50	0.25	0.14
61	00111101	0.23	0.20	0.20	0.53	0.45	0.25	0.15
62	00111110	0.14	0.11	0.11	0.62	0.30	0.34	0.07
63	00111111	0.12	0.09	0.09	0.65	0.25	0.36	0.07
64	01000000	0.11	0.11	0.11	0.38	0.70	0.23	0.07
65	01000001	0.12	0.12	0.12	0.40	0.65	0.23	0.08
66	01000010	0.22	0.22	0.22	0.50	0.50	0.23	0.15
67	01000011	0.23	0.20	0.20	0.53	0.45	0.25	0.15
68	01000100	0.44	0.44	0.44	0.60	0.50	0.23	0.32
69	01000101	0.45	0.45	0.45	0.62	0.45	0.23	0.33
70	01000110	0.55	0.36	0.36	0.72	0.30	0.33	0.27
71	01000111	0.56	0.34	0.34	0.75	0.25	0.34	0.27
72	01001000	0.66	0.66	0.66	0.50	0.65	0.55	0.39
73	01001001	0.67	0.67	0.67	0.53	0.61	0.55	0.40
74	01001010	0.77	0.77	0.77	0.62	0.45	0.55	0.47
75	01001011	0.78	0.75	0.75	0.65	0.40	0.56	0.47
76	01001100	0.98	0.52	0.52	0.72	0.45	0.78	0.35
77	01001101	1.00	0.53	0.53	0.75	0.40	0.78	0.35
78	01001110	0.91	0.44	0.44	0.85	0.25	0.88	0.30
79	01001111	0.89	0.42	0.42	0.88	0.20	0.89	0.29
80	01010000	0.66	0.66	0.66	0.25	0.75	0.00	0.40
81	01010001	0.67	0.67	0.67	0.28	0.70	0.00	0.41
82	01010010	0.77	0.77	0.77	0.38	0.55	0.00	0.46
83	01010011	0.78	0.75	0.75	0.40	0.50	0.02	0.45
84	01010100	0.98	0.98	0.98	0.47	0.55	0.00	0.62
85	01010101	1.00	1.00	1.00	0.50	0.50	0.00	0.62
86	01010110	0.91	0.91	0.91	0.60	0.35	0.09	0.54
87	01010111	0.89	0.89	0.89	0.62	0.30	0.11	0.54
88	01011000	0.80	0.58	0.58	0.38	0.70	0.00	0.40
89	01011001	0.78	0.59	0.59	0.40	0.65	0.00	0.41
90	01011010	0.69	0.69	0.69	0.50	0.50	0.00	0.46
91	01011011	0.67	0.67	0.67	0.53	0.45	0.02	0.45
92	01011100	0.47	0.44	0.44	0.60	0.50	0.23	0.32
93	01011101	0.45	0.45	0.45	0.62	0.45	0.23	0.33
94	01011110	0.36	0.36	0.36	0.72	0.30	0.33	0.24
95	01011111	0.34	0.34	0.34	0.75	0.25	0.34	0.24
96	01100000	0.44	0.25	0.25	0.15	0.90	0.00	0.19
97	01100001	0.45	0.27	0.27	0.17	0.86	0.00	0.19
98	01100010	0.55	0.36	0.36	0.28	0.70	0.00	0.27
99	01100011	0.56	0.34	0.34	0.30	0.65	0.02	0.27
100	01100100	0.77	0.58	0.58	0.38	0.70	0.00	0.40
101	01100101	0.78	0.59	0.59	0.40	0.65	0.00	0.41
102	01100110	0.88	0.50	0.50	0.50	0.50	0.09	0.35
103	01100111	0.89	0.48	0.48	0.53	0.45	0.11	0.34
104	01101000	0.98	0.80	0.80	0.28	0.86	0.31	0.48

Rule	Binary	λ	Z	μ	AA	ND	AP	υ
105	01101001	1.00	0.81	0.81	0.30	0.81	0.31	0.48
106	01101010	0.91	0.91	0.91	0.40	0.65	0.31	0.56
107	01101011	0.89	0.89	0.89	0.42	0.61	0.33	0.55
108	01101100	0.69	0.66	0.66	0.50	0.65	0.55	0.39
109	01101101	0.67	0.67	0.67	0.53	0.61	0.55	0.40
110	01101110	0.58	0.58	0.58	0.62	0.45	0.64	0.34
111	01101111	0.56	0.56	0.56	0.65	0.40	0.66	0.33
112	01110000	0.98	0.33	0.33	0.03	0.95	0.00	0.24
113	01110001	1.00	0.34	0.34	0.05	0.90	0.00	0.24
114	01110010	0.91	0.44	0.44	0.15	0.75	0.00	0.30
115	01110011	0.89	0.42	0.42	0.17	0.70	0.02	0.29
116	01110100	0.69	0.66	0.66	0.25	0.75	0.00	0.42
117	01110101	0.67	0.67	0.67	0.28	0.70	0.00	0.42
118	01110110	0.58	0.58	0.58	0.38	0.55	0.09	0.34
119	01110111	0.56	0.56	0.56	0.40	0.50	0.11	0.33
120	01111000	0.47	0.25	0.25	0.15	0.90	0.00	0.21
121	01111001	0.45	0.27	0.27	0.17	0.86	0.00	0.21
122	01111010	0.36	0.36	0.36	0.28	0.70	0.00	0.26
123	01111011	0.34	0.34	0.34	0.30	0.65	0.02	0.26
124	01111100	0.14	0.11	0.11	0.38	0.70	0.23	0.09
125	01111101	0.12	0.12	0.12	0.40	0.65	0.23	0.09
126	01111110	0.03	0.03	0.03	0.50	0.50	0.33	0.01
127	01111111	0.02	0.02	0.02	0.53	0.45	0.34	0.00
128	10000000	0.02	0.02	0.02	0.47	0.55	0.33	0.00
129	10000001	0.03	0.03	0.03	0.50	0.50	0.33	0.01
130	10000010	0.12	0.12	0.12	0.60	0.35	0.33	0.09
131	10000011	0.14	0.11	0.11	0.62	0.30	0.34	0.09
132	10000100	0.34	0.34	0.34	0.70	0.35	0.33	0.26
133	10000101	0.36	0.36	0.36	0.72	0.30	0.33	0.26
134	10000110	0.45	0.27	0.27	0.82	0.14	0.42	0.21
135	10000111	0.47	0.25	0.25	0.85	0.10	0.44	0.21
136	10001000	0.56	0.56	0.56	0.60	0.50	0.64	0.33
137	10001001	0.58	0.58	0.58	0.62	0.45	0.64	0.34
138	10001010	0.67	0.67	0.67	0.72	0.30	0.64	0.42
139	10001011	0.69	0.66	0.66	0.75	0.25	0.66	0.42
140	10001100	0.89	0.42	0.42	0.82	0.30	0.88	0.29
141	10001101	0.91	0.44	0.44	0.85	0.25	0.88	0.30
142	10001110	1.00	0.34	0.34	0.95	0.10	0.97	0.24
143	10001111	0.98	0.33	0.33	0.97	0.05	0.98	0.24
144	10010000	0.56	0.56	0.56	0.35	0.60	0.09	0.33
145	10010001	0.58	0.58	0.58	0.38	0.55	0.09	0.34
146	10010010	0.67	0.67	0.67	0.47	0.39	0.09	0.40
147	10010011	0.69	0.66	0.66	0.50	0.35	0.11	0.39
148	10010100	0.89	0.89	0.89	0.57	0.39	0.09	0.55
149	10010101	0.91	0.91	0.91	0.60	0.35	0.09	0.56
150	10010110	1.00	0.81	0.81	0.70	0.19	0.19	0.48
151	10010111	0.98	0.80	0.80	0.72	0.14	0.20	0.48
152	10011000	0.89	0.48	0.48	0.47	0.55	0.09	0.34
153	10011001	0.88	0.50	0.50	0.50	0.50	0.09	0.35
154	10011010	0.78	0.59	0.59	0.60	0.35	0.09	0.41
155	10011011	0.77	0.58	0.58	0.62	0.30	0.11	0.40
156	10011100	0.56	0.34	0.34	0.70	0.35	0.33	0.27
157	10011101	0.55	0.36	0.36	0.72	0.30	0.33	0.27
158	10011110	0.45	0.27	0.27	0.82	0.14	0.42	0.19
159	10011111	0.44	0.25	0.25	0.85	0.10	0.44	0.19
160	10100000	0.34	0.34	0.34	0.25	0.75	0.00	0.24
161	10100001	0.36	0.36	0.36	0.28	0.70	0.00	0.24
162	10100010	0.45	0.45	0.45	0.38	0.55	0.00	0.33
163	10100011	0.47	0.44	0.44	0.40	0.50	0.02	0.32

Rule	Binary	λ	Z	μ	AA	ND	AP	υ
164	10100100	0.67	0.67	0.67	0.47	0.55	0.00	0.45
165	10100101	0.69	0.69	0.69	0.50	0.50	0.00	0.46
166	10100110	0.78	0.59	0.59	0.60	0.35	0.09	0.41
167	10100111	0.80	0.58	0.58	0.62	0.30	0.11	0.40
168	10101000	0.89	0.89	0.89	0.38	0.70	0.31	0.54
169	10101001	0.91	0.91	0.91	0.40	0.65	0.31	0.54
170	10101010	1.00	1.00	1.00	0.50	0.50	0.31	0.62
171	10101011	0.98	0.98	0.98	0.53	0.45	0.33	0.62
172	10101100	0.78	0.75	0.75	0.60	0.50	0.55	0.45
173	10101101	0.77	0.77	0.77	0.62	0.45	0.55	0.46
174	10101110	0.67	0.67	0.67	0.72	0.30	0.64	0.41
175	10101111	0.66	0.66	0.66	0.75	0.25	0.66	0.40
176	10110000	0.89	0.42	0.42	0.12	0.80	0.00	0.29
177	10110001	0.91	0.44	0.44	0.15	0.75	0.00	0.30
178	10110010	1.00	0.53	0.53	0.25	0.60	0.00	0.35
179	10110011	0.98	0.52	0.52	0.28	0.55	0.02	0.35
180	10110100	0.78	0.75	0.75	0.35	0.60	0.00	0.47
181	10110101	0.77	0.77	0.77	0.38	0.55	0.00	0.47
182	10110110	0.67	0.67	0.67	0.47	0.39	0.09	0.40
183	10110111	0.66	0.66	0.66	0.50	0.35	0.11	0.39
184	10111000	0.56	0.34	0.34	0.25	0.75	0.00	0.27
185	10111001	0.55	0.36	0.36	0.28	0.70	0.00	0.27
186	10111010	0.45	0.45	0.45	0.38	0.55	0.00	0.33
187	10111011	0.44	0.44	0.44	0.40	0.50	0.02	0.32
188	10111100	0.23	0.20	0.20	0.47	0.55	0.23	0.15
189	10111101	0.22	0.22	0.22	0.50	0.50	0.23	0.15
190	10111110	0.12	0.12	0.12	0.60	0.35	0.33	0.08
191	10111111	0.11	0.11	0.11	0.62	0.30	0.34	0.07
192	11000000	0.12	0.09	0.09	0.35	0.75	0.23	0.07
193	11000001	0.14	0.11	0.11	0.38	0.70	0.23	0.07
194	11000010	0.23	0.20	0.20	0.47	0.55	0.23	0.15
195	11000011	0.25	0.19	0.19	0.50	0.50	0.25	0.14
196	11000100	0.45	0.42	0.42	0.57	0.55	0.23	0.32
197	11000101	0.47	0.44	0.44	0.60	0.50	0.23	0.32
198	11000110	0.56	0.34	0.34	0.70	0.35	0.33	0.27
199	11000111	0.58	0.33	0.33	0.72	0.30	0.34	0.26
200	11001000	0.67	0.64	0.64	0.47	0.70	0.55	0.39
201	11001001	0.69	0.66	0.66	0.50	0.65	0.55	0.39
202	11001010	0.78	0.75	0.75	0.60	0.50	0.55	0.47
203	11001011	0.80	0.73	0.73	0.62	0.45	0.56	0.46
204	11001100	1.00	0.50	0.50	0.70	0.50	0.78	0.35
205	11001101	0.98	0.52	0.52	0.72	0.45	0.78	0.35
206	11001110	0.89	0.42	0.42	0.82	0.30	0.88	0.29
207	11001111	0.88	0.41	0.41	0.85	0.25	0.89	0.29
208	11010000	0.67	0.64	0.64	0.23	0.80	0.00	0.40
209	11010001	0.69	0.66	0.66	0.25	0.75	0.00	0.40
210	11010010	0.78	0.75	0.75	0.35	0.60	0.00	0.45
211	11010011	0.80	0.73	0.73	0.38	0.55	0.02	0.45
212	11010100	1.00	0.97	0.97	0.45	0.60	0.00	0.62
213	11010101	0.98	0.98	0.98	0.47	0.55	0.00	0.62
214	11010110	0.89	0.89	0.89	0.57	0.39	0.09	0.54
215	11010111	0.88	0.88	0.88	0.60	0.35	0.11	0.53
216	11011000	0.78	0.56	0.56	0.35	0.75	0.00	0.40
217	11011001	0.77	0.58	0.58	0.38	0.70	0.00	0.40
218	11011010	0.67	0.67	0.67	0.47	0.55	0.00	0.45
219	11011011	0.66	0.66	0.66	0.50	0.50	0.02	0.45
220	11011100	0.45	0.42	0.42	0.57	0.55	0.23	0.32
221	11011101	0.44	0.44	0.44	0.60	0.50	0.23	0.32
222	11011110	0.34	0.34	0.34	0.70	0.35	0.33	0.24

Rule	Binary	λ	Z	μ	AA	ND	AP	υ
223	11011111	0.33	0.33	0.33	0.72	0.30	0.34	0.24
224	11100000	0.45	0.23	0.23	0.12	0.95	0.00	0.19
225	11100001	0.47	0.25	0.25	0.15	0.90	0.00	0.19
226	11100010	0.56	0.34	0.34	0.25	0.75	0.00	0.27
227	11100011	0.58	0.33	0.33	0.28	0.70	0.02	0.26
228	11100100	0.78	0.56	0.56	0.35	0.75	0.00	0.40
229	11100101	0.80	0.58	0.58	0.38	0.70	0.00	0.40
230	11100110	0.89	0.48	0.48	0.47	0.55	0.09	0.34
231	11100111	0.91	0.47	0.47	0.50	0.50	0.11	0.34
232	11101000	1.00	0.78	0.78	0.25	0.90	0.31	0.47
233	11101001	0.98	0.80	0.80	0.28	0.86	0.31	0.48
234	11101010	0.89	0.89	0.89	0.38	0.70	0.31	0.55
235	11101011	0.88	0.88	0.88	0.40	0.65	0.33	0.55
236	11101100	0.67	0.64	0.64	0.47	0.70	0.55	0.39
237	11101101	0.66	0.66	0.66	0.50	0.65	0.55	0.39
238	11101110	0.56	0.56	0.56	0.60	0.50	0.64	0.33
239	11101111	0.55	0.55	0.55	0.62	0.45	0.66	0.33
240	11110000	1.00	0.31	0.31	0.00	1.00	0.00	0.24
241	11110001	0.98	0.33	0.33	0.03	0.95	0.00	0.24
242	11110010	0.89	0.42	0.42	0.12	0.80	0.00	0.29
243	11110011	0.88	0.41	0.41	0.15	0.75	0.02	0.29
244	11110100	0.67	0.64	0.64	0.23	0.80	0.00	0.41
245	11110101	0.66	0.66	0.66	0.25	0.75	0.00	0.42
246	11110110	0.56	0.56	0.56	0.35	0.60	0.09	0.33
247	11110111	0.55	0.55	0.55	0.38	0.55	0.11	0.33
248	11111000	0.45	0.23	0.23	0.12	0.95	0.00	0.20
249	11111001	0.44	0.25	0.25	0.15	0.90	0.00	0.21
250	11111010	0.34	0.34	0.34	0.25	0.75	0.00	0.26
251	11111011	0.33	0.33	0.33	0.28	0.70	0.02	0.25
252	11111100	0.12	0.09	0.09	0.35	0.75	0.23	0.08
253	11111101	0.11	0.11	0.11	0.38	0.70	0.23	0.09
254	11111110	0.02	0.02	0.02	0.47	0.55	0.33	0.00
255	11111111	0.00	0.00	0.00	0.50	0.50	0.34	0.00

Appendix B

Parameter Efficacy Statistics

Below is a table of overlap, clustering, and NN error statistics for every subset of the seven parameters used in classification. λ is an activity parameter, Z is based on a CA reverse algorithm, μ is a sensitivity parameter, AA is absolute activity, ND is neighborhood dominance, AP is activity propagation, and υ is incompressibility.

Overlap is the number of CA rules that have the same value for every parameter in the set. The minimal number of overlaps, two, requires at least five parameters. There are three sets of five parameters achieving this value, all of which include AA, ND, AP, and υ, and any one of the other three parameters.

The cluster ratio is calculated as mean intra-cluster divided by mean inter-cluster. Higher cluster ratios are better. The maximum cluster ratio, 2.241, is achieved by the set of parameters including only μ.

The NN error is the mean-squared error of a neural network after training to classify the 256 elementary CA into the six Li-Packard classes. The error given is averaged over five training runs to account for the random weights each NN is initialized with. Lower NN error values are better. The minimum value, 0.0027, is achieved by the set containing λ, AA, ND, AP, and υ.

As a method of ranking the seven parameters, the mean values for overlap, clustering, and NN error values were calculated for parameter over the set of subsets that parameter appears is. Each parameter appears in 64 of the 127 parameter subsets. The table below shows the ranking of the seven parameters, sorted by increasing NN error values.

Rank	Parameter	Overlap	Intra	Inter	Cluster Ratio	NN Error
1	AP	160	0.283	0.328	1.161	0.0215
2	AA	189	0.278	0.334	1.206	0.0235
3	ND	185	0.273	0.334	1.235	0.0240
4	υ	204	0.287	0.325	1.125	0.0297
5	μ	311	0.264	0.335	1.296	0.0304
6	Z	450	0.274	0.345	1.278	0.0322
7	λ	461	0.281	0.341	1.225	0.0326

Parameter Set	Overlap	Intra	Inter	Cluster Ratio	NN Error
λ Z μ AA ND AP υ	2	0.380	0.453	1.191	0.0057
λ Z μ AA ND AP	32	0.333	0.431	1.294	0.0096
λ Z μ AA ND υ	27	0.342	0.428	1.253	0.0138
λ Z μ AA AP υ	3	0.358	0.420	1.174	0.0038
λ Z μ ND AP υ	22	0.350	0.420	1.201	0.0103
λ Z AA ND AP υ	2	0.369	0.418	1.133	0.0077
λ μ AA ND AP υ	2	0.356	0.404	1.134	0.0050
Z μ AA ND AP υ	2	0.346	0.408	1.180	0.0050
λ Z μ AA ND	148	0.290	0.405	1.395	0.0231
λ Z μ AA AP	37	0.308	0.397	1.290	0.0097
λ Z μ AA υ	68	0.315	0.392	1.242	0.0226
λ Z μ ND AP	122	0.299	0.397	1.328	0.0217
λ Z μ ND υ	51	0.305	0.392	1.283	0.0204
λ Z μ AP υ	55	0.324	0.382	1.181	0.0197
λ Z AA ND AP	32	0.319	0.395	1.238	0.0115
λ Z AA ND υ	47	0.330	0.392	1.189	0.0157
λ Z AA AP υ	3	0.346	0.383	1.107	0.0059
λ Z ND AP υ	22	0.338	0.383	1.134	0.0114
λ μ AA ND AP	32	0.305	0.379	1.243	0.0065
λ μ AA ND υ	27	0.315	0.376	1.193	0.0132
λ μ AA AP υ	3	0.331	0.367	1.108	0.0048
λ μ ND AP υ	23	0.324	0.368	1.135	0.0102
λ AA ND AP υ	2	0.344	0.363	1.056	0.0027
Z μ AA ND AP	36	0.295	0.383	1.302	0.0102
Z μ AA ND υ	35	0.303	0.381	1.258	0.0169
Z μ AA AP υ	3	0.321	0.372	1.160	0.0081
Z μ ND AP υ	24	0.312	0.372	1.193	0.0126
Z AA ND AP υ	2	0.334	0.369	1.106	0.0058
μ AA ND AP υ	2	0.317	0.351	1.109	0.0044
λ Z μ AA	213	0.260	0.367	1.409	0.0315
λ Z μ ND	310	0.249	0.367	1.475	0.0329
λ Z μ AP	289	0.269	0.358	1.331	0.0351
λ Z μ υ	491	0.273	0.345	1.263	0.0827
λ Z AA ND	276	0.274	0.367	1.336	0.0327
λ Z AA AP	53	0.293	0.358	1.223	0.0122
λ Z AA υ	126	0.303	0.352	1.164	0.0329
λ Z ND AP	122	0.284	0.358	1.264	0.0212
λ Z ND υ	103	0.292	0.353	1.207	0.0256

Parameter Set	Overlap	Intra	Inter	Cluster Ratio	NN Error
λ Z AP υ	63	0.311	0.342	1.101	0.0250
λ μ AA ND	148	0.259	0.349	1.346	0.0234
λ μ AA AP	37	0.277	0.341	1.231	0.0077
λ μ AA υ	69	0.287	0.336	1.170	0.0227
λ μ ND AP	178	0.268	0.341	1.271	0.0295
λ μ ND υ	52	0.277	0.336	1.212	0.0208
λ μ AP υ	74	0.295	0.325	1.102	0.0233
λ AA ND AP	32	0.289	0.336	1.162	0.0078
λ AA ND υ	47	0.302	0.333	1.101	0.0163
λ AA AP υ	3	0.318	0.322	1.012	0.0058
λ ND AP υ	23	0.311	0.322	1.038	0.0096
Z μ AA ND	168	0.246	0.355	1.440	0.0265
Z μ AA AP	45	0.266	0.346	1.299	0.0107
Z μ AA υ	76	0.273	0.341	1.248	0.0248
Z μ ND AP	146	0.256	0.346	1.349	0.0247
Z μ ND υ	67	0.262	0.342	1.305	0.0224
Z μ AP υ	69	0.282	0.330	1.170	0.0219
Z AA ND AP	36	0.279	0.342	1.225	0.0095
Z AA ND υ	55	0.290	0.340	1.174	0.0189
Z AA AP υ	8	0.308	0.329	1.070	0.0084
Z ND AP υ	24	0.299	0.330	1.102	0.0127
μ AA ND AP	40	0.259	0.322	1.242	0.0131
μ AA ND υ	35	0.270	0.321	1.188	0.0187
μ AA AP υ	7	0.288	0.310	1.076	0.0101
μ ND AP υ	27	0.280	0.310	1.108	0.0141
AA ND AP υ	20	0.303	0.302	0.999	0.0113
λ Z μ	2109	0.209	0.317	1.517	0.0964
λ Z AA	597	0.243	0.325	1.337	0.0505
λ Z ND	670	0.231	0.325	1.410	0.0421
λ Z AP	495	0.251	0.315	1.254	0.0510
λ Z υ	878	0.258	0.300	1.163	0.0910
λ μ AA	229	0.226	0.306	1.355	0.0347
λ μ ND	398	0.215	0.306	1.426	0.0404
λ μ AP	433	0.234	0.297	1.266	0.0420
λ μ υ	548	0.241	0.282	1.172	0.0873
λ AA ND	316	0.241	0.303	1.255	0.0353
λ AA AP	61	0.260	0.293	1.129	0.0141
λ AA υ	151	0.272	0.286	1.051	0.0411
λ ND AP	178	0.250	0.293	1.172	0.0297
λ ND υ	124	0.262	0.287	1.093	0.0314
λ AP υ	101	0.280	0.273	0.975	0.0331
Z μ AA	247	0.212	0.312	1.474	0.0338
Z μ ND	370	0.198	0.312	1.579	0.0369
Z μ AP	363	0.221	0.301	1.363	0.0364
Z μ υ	563	0.222	0.288	1.297	0.0860
Z AA ND	300	0.228	0.310	1.359	0.0354
Z AA AP	81	0.249	0.300	1.204	0.0172
Z AA υ	176	0.259	0.295	1.140	0.0439
Z ND AP	146	0.239	0.300	1.257	0.0233
Z ND υ	119	0.247	0.296	1.198	0.0295
Z AP υ	84	0.268	0.282	1.055	0.0285

Parameter Set	Overlap	Intra	Inter	Cluster Ratio	NN Error
μ AA ND	172	0.205	0.288	1.407	0.0297
μ AA AP	65	0.225	0.277	1.233	0.0162
μ AA υ	93	0.236	0.274	1.162	0.0320
μ ND AP	220	0.216	0.277	1.285	0.0359
μ ND υ	70	0.225	0.275	1.223	0.0253
μ AP υ	104	0.244	0.260	1.069	0.0279
AA ND AP	168	0.240	0.267	1.114	0.0278
AA ND υ	133	0.255	0.268	1.052	0.0403
AA AP υ	75	0.272	0.253	0.929	0.0281
ND AP υ	49	0.264	0.253	0.957	0.0233
λ Z	5009	0.187	0.269	1.440	0.1052
λ μ	2701	0.169	0.249	1.475	0.1013
λ AA	773	0.206	0.253	1.230	0.0659
λ ND	984	0.193	0.252	1.309	0.0586
λ AP	726	0.212	0.240	1.136	0.0594
λ υ	1273	0.224	0.221	0.990	0.1049
Z μ	2429	0.145	0.255	1.762	0.0969
Z AA	867	0.192	0.262	1.369	0.0628
Z ND	742	0.176	0.263	1.490	0.0475
Z AP	679	0.200	0.249	1.243	0.0572
Z υ	1229	0.205	0.234	1.143	0.0978
μ AA	315	0.162	0.237	1.461	0.0420
μ ND	492	0.149	0.237	1.596	0.0462
μ AP	597	0.169	0.223	1.323	0.0490
μ υ	704	0.174	0.210	1.203	0.0898
AA ND	852	0.181	0.228	1.259	0.0673
AA AP	493	0.202	0.212	1.046	0.0587
AA υ	633	0.218	0.210	0.962	0.0748
ND AP	476	0.192	0.212	1.102	0.0534
ND υ	246	0.207	0.211	1.020	0.0531
AP υ	314	0.225	0.189	0.838	0.0595
λ	7311	0.137	0.180	1.311	0.1146
Z	7145	0.112	0.194	1.734	0.1071
μ	3415	0.073	0.163	2.241	0.1024
AA	3309	0.132	0.160	1.212	0.0953
ND	2022	0.117	0.161	1.373	0.0850
AP	2630	0.133	0.131	0.982	0.0950
υ	3557	0.149	0.114	0.765	0.1165

Appendix C

Examples of CA Patterns

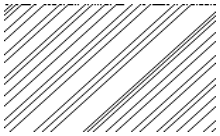
C.1 Elementary CA

The images on the following pages provide examples of the evolution of the 256 elementary CA (see section 2.1 for definition). The CA have periodic boundary conditions, are 150 cells wide, 100 time steps are shown, and all begin from the same initial condition, which was randomly generated with each cell having an equal probability of being in state 0 or 1.

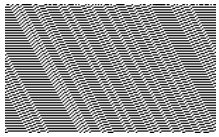
elem 0 (00000000)

elem 1 (00000001)

elem 2 (00000010)

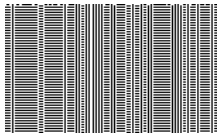


elem 3 (00000011)

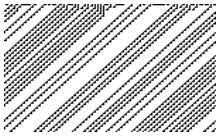


elem 4 (00000100)

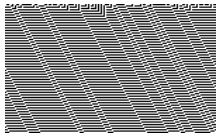
elem 5 (00000101)



elem 6 (00000110)

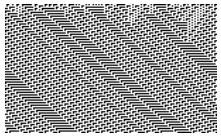


elem 7 (00000111)

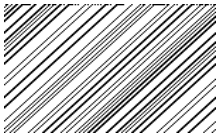


elem 8 (00001000)

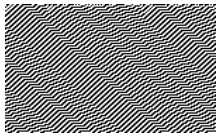
elem 9 (00001001)



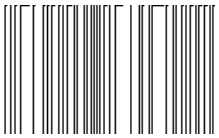
elem 10 (00001010)



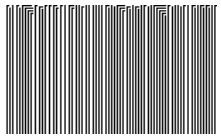
elem 11 (00001011)



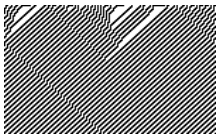
elem 12 (00001100)



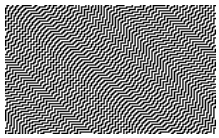
elem 13 (00001101)



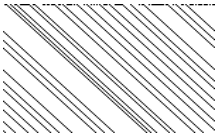
elem 14 (00001110)



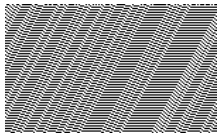
elem 15 (00001111)



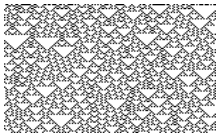
elem 16 (00010000)



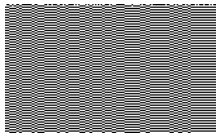
elem 17 (00010001)



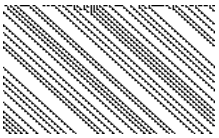
elem 18 (00010010)



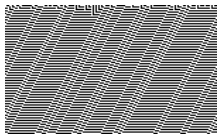
elem 19 (00010011)



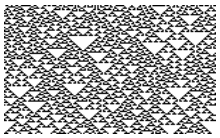
elem 20 (00010100)



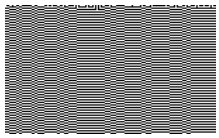
elem 21 (00010101)



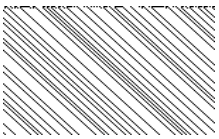
elem 22 (00010110)



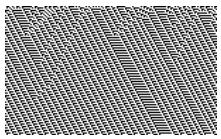
elem 23 (00010111)



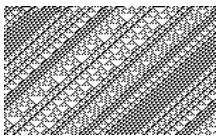
elem 24 (00011000)



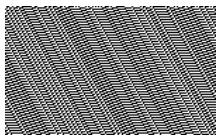
elem 25 (00011001)



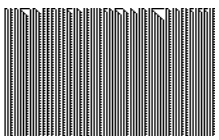
elem 26 (00011010)



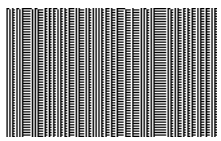
elem 27 (00011011)



elem 28 (00011100)

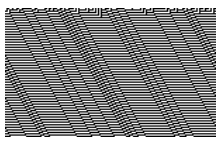


elem 29 (00011101)



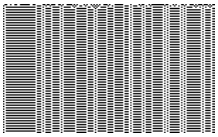
elem 30 (00011110)

elem 31 (00011111)

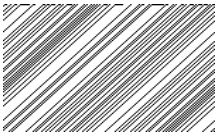


elem 32 (00100000)

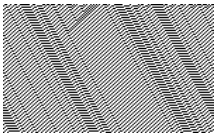
elem 33 (00100001)



elem 34 (00100010)

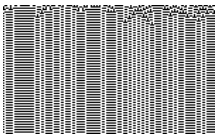


elem 35 (00100011)

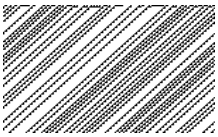


elem 36 (00100100)

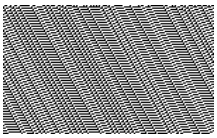
elem 37 (00100101)



elem 38 (00100110)

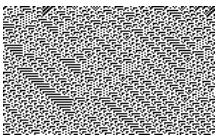


elem 39 (00100111)



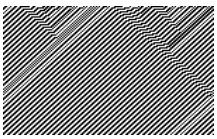
elem 40 (00101000)

elem 41 (00101001)

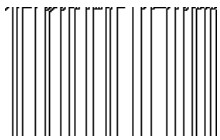


elem 42 (00101010)

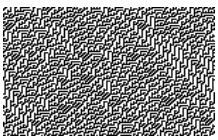
elem 43 (00101011)



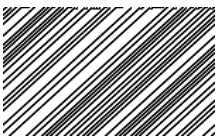
elem 44 (00101100)



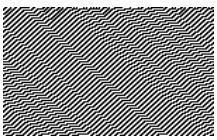
elem 45 (00101101)



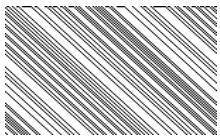
elem 46 (00101110)



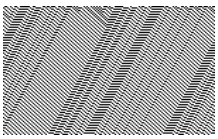
elem 47 (00101111)



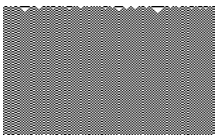
elem 48 (00110000)



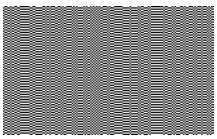
elem 49 (00110001)



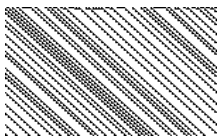
elem 50 (00110010)



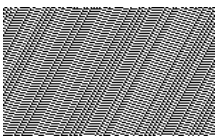
elem 51 (00110011)



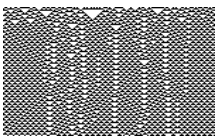
elem 52 (00110100)



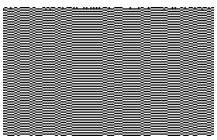
elem 53 (00110101)



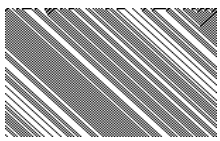
elem 54 (00110110)



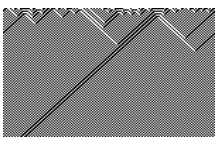
elem 55 (00110111)



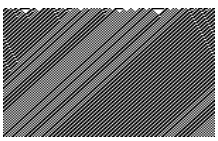
elem 56 (00111000)



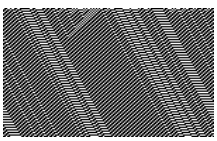
elem 57 (00111001)



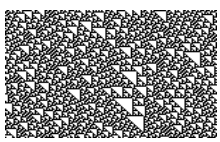
elem 58 (00111010)



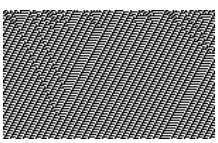
elem 59 (00111011)



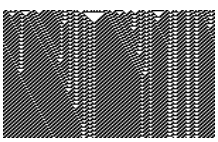
elem 60 (00111100)



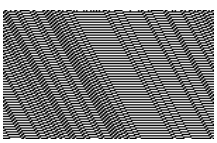
elem 61 (00111101)



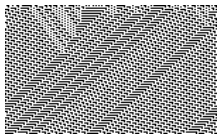
elem 62 (00111110)

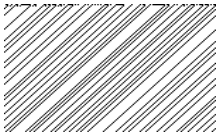


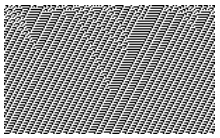
elem 63 (00111111)

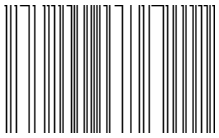


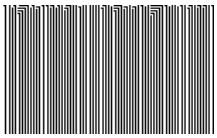
elem 64 (01000000)

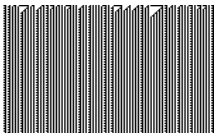
elem 65 (01000001)


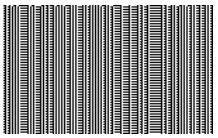
elem 66 (01000010)


elem 67 (01000011)


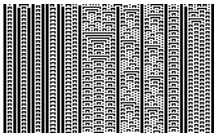
elem 68 (01000100)


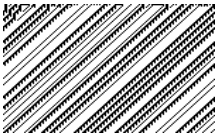
elem 69 (01000101)


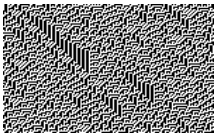
elem 70 (01000110)


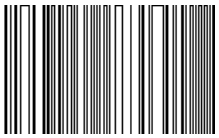
elem 71 (01000111)


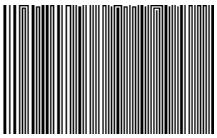
elem 72 (01001000)

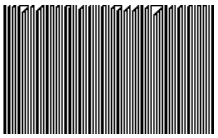

elem 73 (01001001)


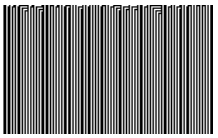
elem 74 (01001010)


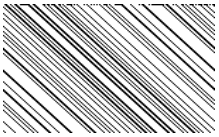
elem 75 (01001011)


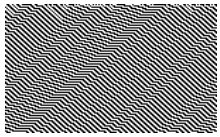
elem 76 (01001100)


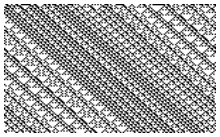
elem 77 (01001101)


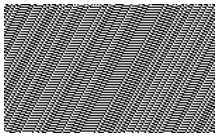
elem 78 (01001110)


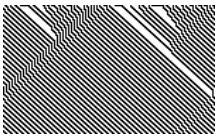
elem 79 (01001111)


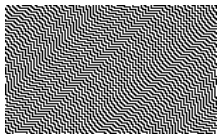
elem 80 (01010000)


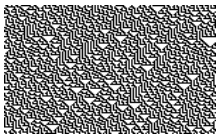
elem 81 (01010001)


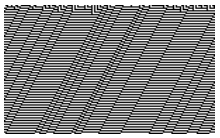
elem 82 (01010010)


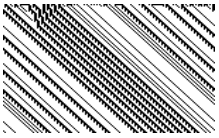
elem 83 (01010011)


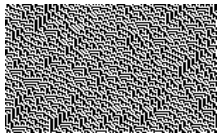
elem 84 (01010100)


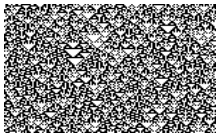
elem 85 (01010101)


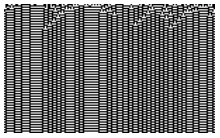
elem 86 (01010110)


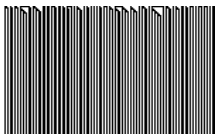
elem 87 (01010111)


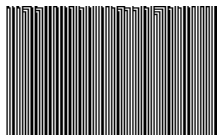
elem 88 (01011000)


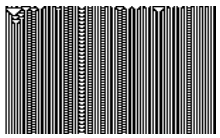
elem 89 (01011001)


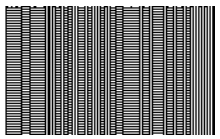
elem 90 (01011010)


elem 91 (01011011)


elem 92 (01011100)


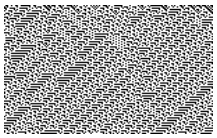
elem 93 (01011101)


elem 94 (01011110)


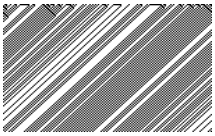
elem 95 (01011111)


elem 96 (01100000)

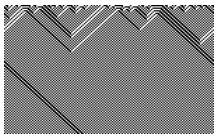
elem 97 (01100001)



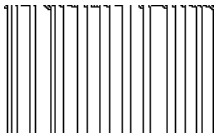
elem 98 (01100010)



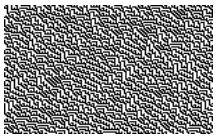
elem 99 (01100011)



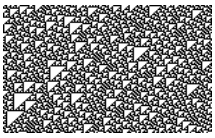
elem 100 (01100100)



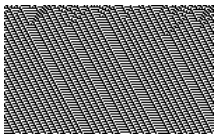
elem 101 (01100101)



elem 102 (01100110)

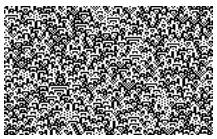


elem 103 (01100111)

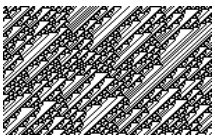


elem 104 (01101000)

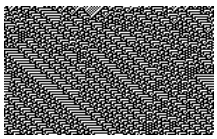
elem 105 (01101001)



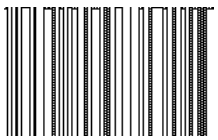
elem 106 (01101010)



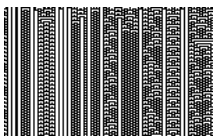
elem 107 (01101011)



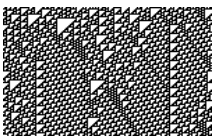
elem 108 (01101100)



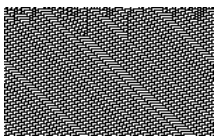
elem 109 (01101101)



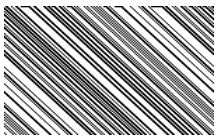
elem 110 (01101110)



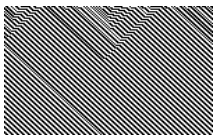
elem 111 (01101111)



elem 112 (01110000)

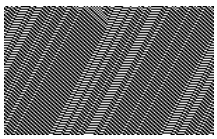


elem 113 (01110001)

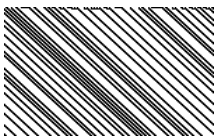


elem 114 (01110010)

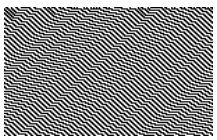
elem 115 (01110011)



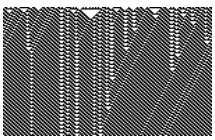
elem 116 (01110100)



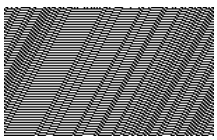
elem 117 (01110101)



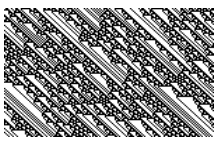
elem 118 (01110110)



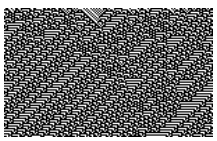
elem 119 (01110111)



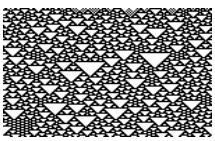
elem 120 (01111000)



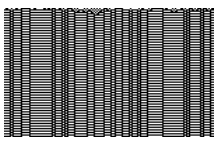
elem 121 (01111001)



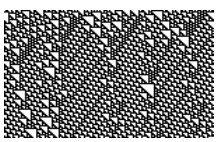
elem 122 (01111010)



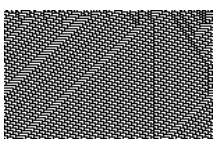
elem 123 (01111011)



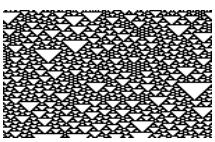
elem 124 (01111100)



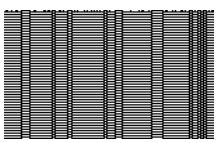
elem 125 (01111101)



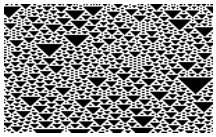
elem 126 (01111110)

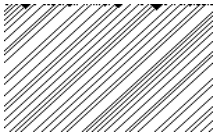


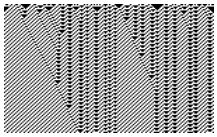
elem 127 (01111111)

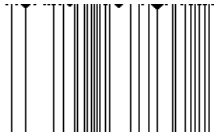


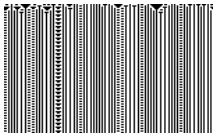
elem 128 (10000000)

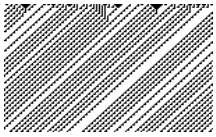
elem 129 (10000001)


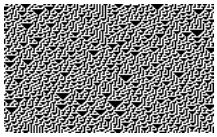
elem 130 (10000010)


elem 131 (10000011)


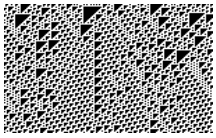
elem 132 (10000100)


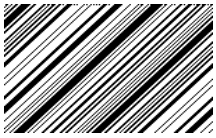
elem 133 (10000101)


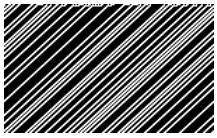
elem 134 (10000110)


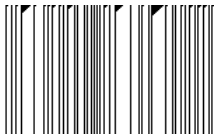
elem 135 (10000111)


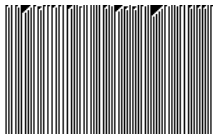
elem 136 (10001000)

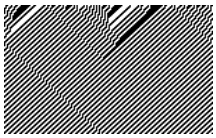
elem 137 (10001001)


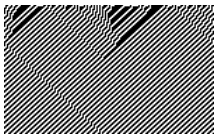
elem 138 (10001010)


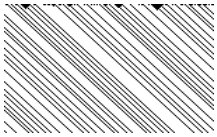
elem 139 (10001011)


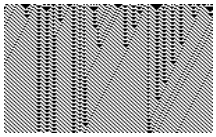
elem 140 (10001100)


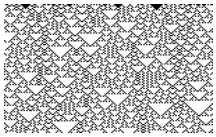
elem 141 (10001101)


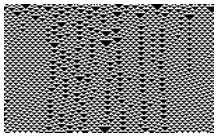
elem 142 (10001110)


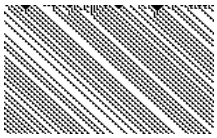
elem 143 (10001111)


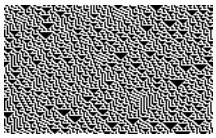
elem 144 (10010000)


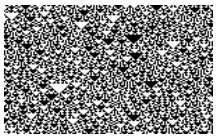
elem 145 (10010001)


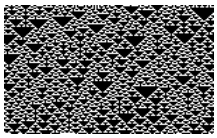
elem 146 (10010010)


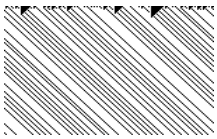
elem 147 (10010011)


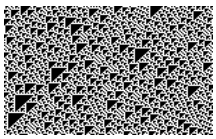
elem 148 (10010100)


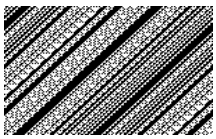
elem 149 (10010101)


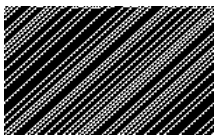
elem 150 (10010110)


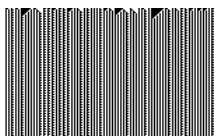
elem 151 (10010111)


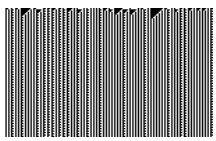
elem 152 (10011000)


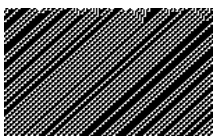
elem 153 (10011001)


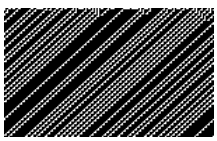
elem 154 (10011010)


elem 155 (10011011)


elem 156 (10011100)


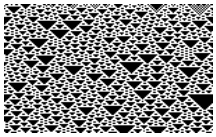
elem 157 (10011101)


elem 158 (10011110)


elem 159 (10011111)


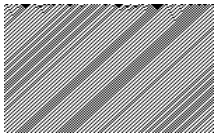
elem 160 (10100000)

elem 161 (10100001)



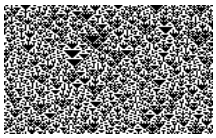
elem 162 (10100010)

elem 163 (10100011)

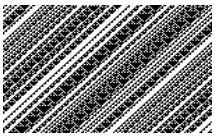


elem 164 (10100100)

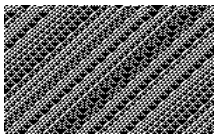
elem 165 (10100101)



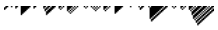
elem 166 (10100110)



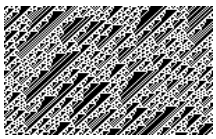
elem 167 (10100111)



elem 168 (10101000)



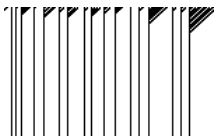
elem 169 (10101001)



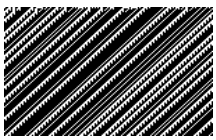
elem 170 (10101010)

elem 171 (10101011)

elem 172 (10101100)

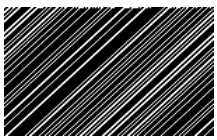


elem 173 (10101101)

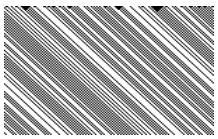


elem 174 (10101110)

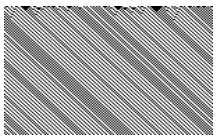
elem 175 (10101111)



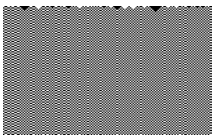
elem 176 (10110000)



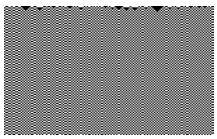
elem 177 (10110001)



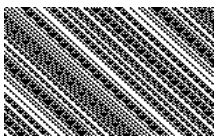
elem 178 (10110010)



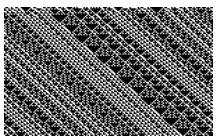
elem 179 (10110011)



elem 180 (10110100)

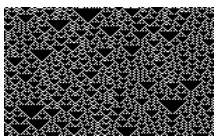


elem 181 (10110101)

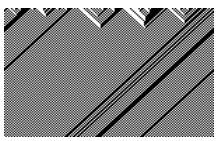


elem 182 (10110110)

elem 183 (10110111)



elem 184 (10111000)



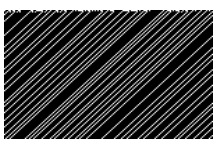
elem 185 (10111001)

elem 186 (10111010)

elem 187 (10111011)

elem 188 (10111100)

elem 189 (10111101)

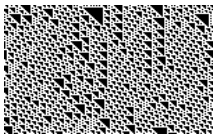


elem 190 (10111110)

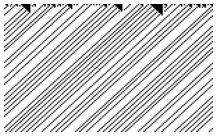
elem 191 (10111111)

elem 192 (11000000)

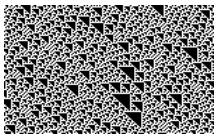
elem 193 (11000001)



elem 194 (11000010)

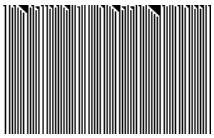


elem 195 (11000011)

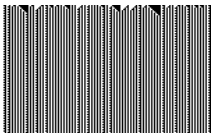


elem 196 (11000100)

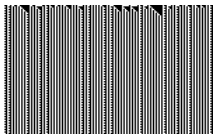
elem 197 (11000101)



elem 198 (11000110)

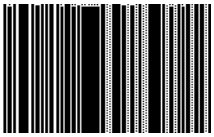


elem 199 (11000111)

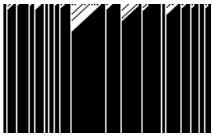


elem 200 (11001000)

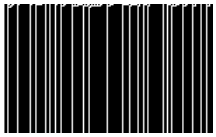
elem 201 (11001001)



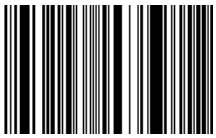
elem 202 (11001010)



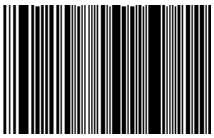
elem 203 (11001011)



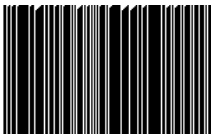
elem 204 (11001100)



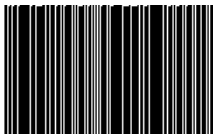
elem 205 (11001101)



elem 206 (11001110)

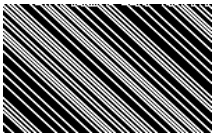


elem 207 (11001111)

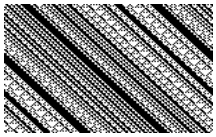


elem 208 (11010000)

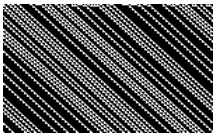
elem 209 (11010001)



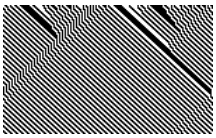
elem 210 (11010010)



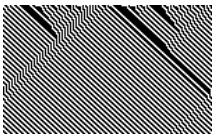
elem 211 (11010011)



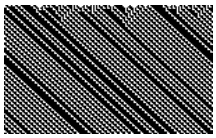
elem 212 (11010100)



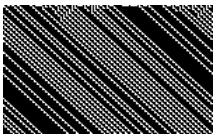
elem 213 (11010101)



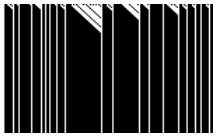
elem 214 (11010110)



elem 215 (11010111)

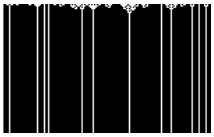


elem 216 (11011000)



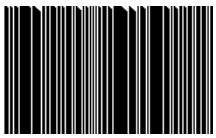
elem 217 (11011001)

elem 218 (11011010)

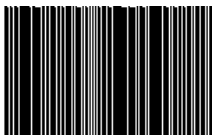


elem 219 (11011011)

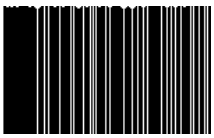
elem 220 (11011100)



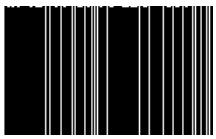
elem 221 (11011101)



elem 222 (11011110)

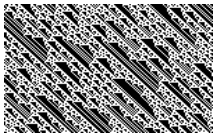


elem 223 (11011111)

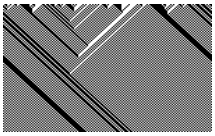


elem 224 (11100000)

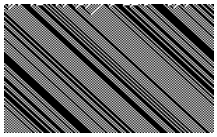
elem 225 (11100001)



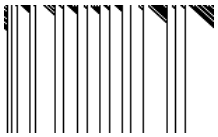
elem 226 (11100010)



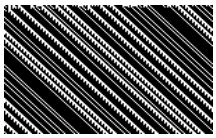
elem 227 (11100011)



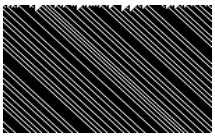
elem 228 (11100100)



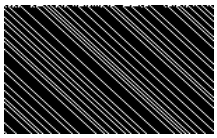
elem 229 (11100101)



elem 230 (11100110)



elem 231 (11100111)



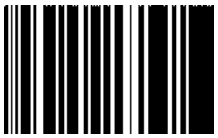
elem 232 (11101000)

elem 233 (11101001)

elem 234 (11101010)

elem 235 (11101011)

elem 236 (11101100)

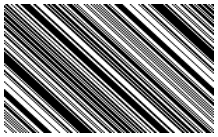


elem 237 (11101101)

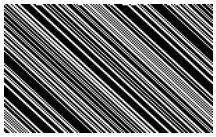
elem 238 (11101110)

elem 239 (11101111)

elem 240 (11110000)



elem 241 (11110001)

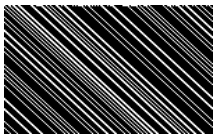


elem 242 (11110010)

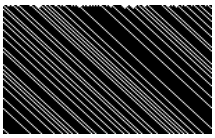
elem 243 (11110011)

elem 244 (11110100)

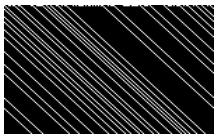
elem 245 (11110101)



elem 246 (11110110)



elem 247 (11110111)



elem 248 (11111000)

elem 249 (11111001)

elem 250 (11111010)

elem 251 (11111011)

elem 252 (11111100)

elem 253 (11111101)

elem 254 (11111110)

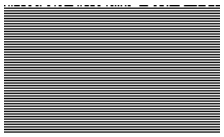
elem 255 (11111111)

C.2 Totalistic $k = 2$ $r = 3$ CA

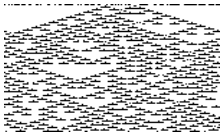
The images on the following pages provide examples of the evolution of the 256 totalistic CA (see section 2.2 for definition) that have 2 states per cell and a radius of 3 (neighborhood size of 7). The CA have periodic boundary conditions, are 150 cells wide, 100 time steps are shown, and all begin from the same initial condition, which was randomly generated with each cell having an equal probability of being in state 0 or 1.

tot 0 (00000000)

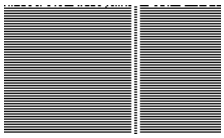
tot 1 (00000001)



tot 2 (00000010)

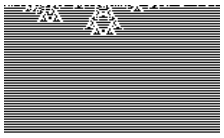


tot 3 (00000011)

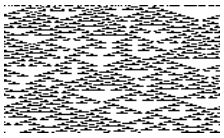


tot 4 (00000100)

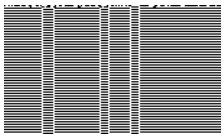
tot 5 (00000101)



tot 6 (00000110)

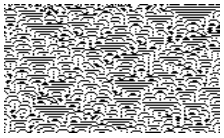


tot 7 (00000111)

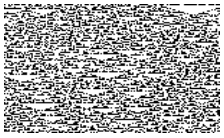


tot 8 (00001000)

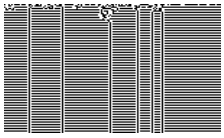
tot 9 (00001001)



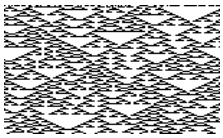
tot 10 (00001010)



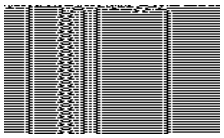
tot 11 (00001011)



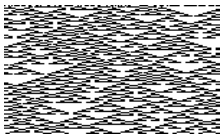
tot 12 (00001100)



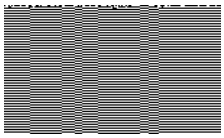
tot 13 (00001101)



tot 14 (00001110)

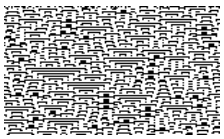


tot 15 (00001111)

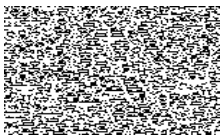


tot 16 (00010000)

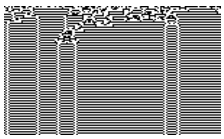
tot 17 (00010001)



tot 18 (00010010)

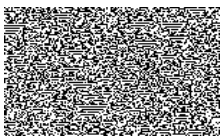


tot 19 (00010011)

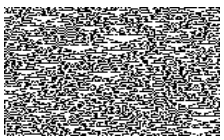


tot 20 (00010100)

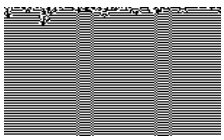
tot 21 (00010101)



tot 22 (00010110)

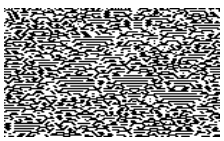


tot 23 (00010111)

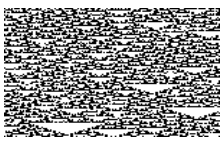


tot 24 (00011000)

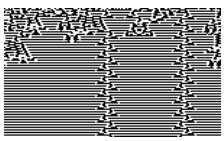
tot 25 (00011001)



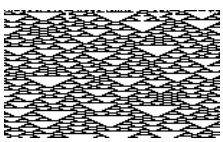
tot 26 (00011010)



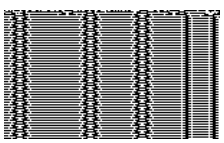
tot 27 (00011011)



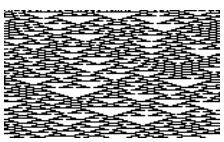
tot 28 (00011100)



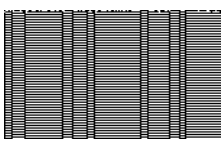
tot 29 (00011101)



tot 30 (00011110)

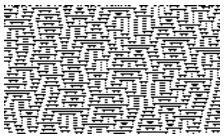


tot 31 (00011111)

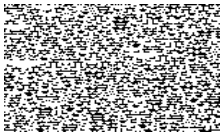


tot 32 (00100000)

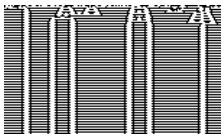
tot 33 (00100001)



tot 34 (00100010)

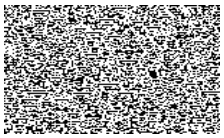


tot 35 (00100011)



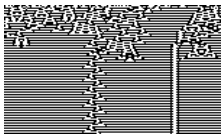
tot 36 (00100100)

tot 37 (00100101)

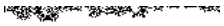


tot 38 (00100110)

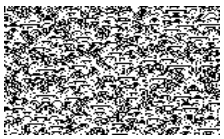
tot 39 (00100111)



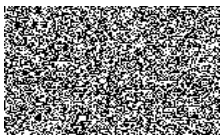
tot 40 (00101000)



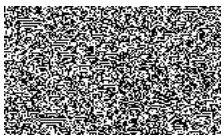
tot 41 (00101001)



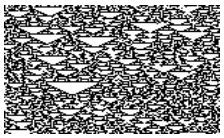
tot 42 (00101010)



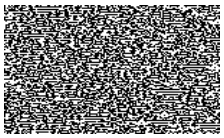
tot 43 (00101011)



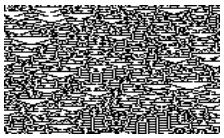
tot 44 (00101100)



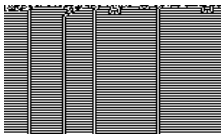
tot 45 (00101101)



tot 46 (00101110)

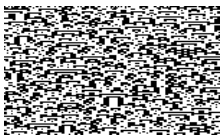


tot 47 (00101111)

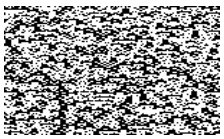


tot 48 (00110000)

tot 49 (00110001)



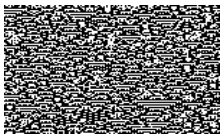
tot 50 (00110010)



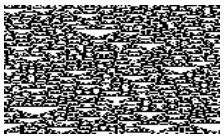
tot 51 (00110011)

tot 52 (00110100)

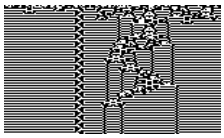
tot 53 (00110101)



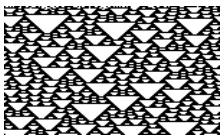
tot 54 (00110110)



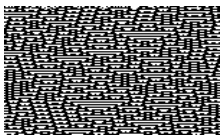
tot 55 (00110111)



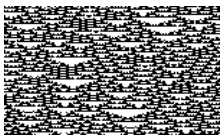
tot 56 (00111000)



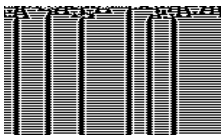
tot 57 (00111001)



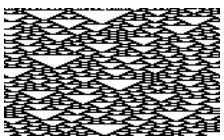
tot 58 (00111010)



tot 59 (00111011)

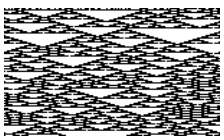


tot 60 (00111100)

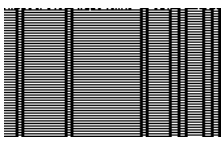


tot 61 (00111101)

tot 62 (00111110)

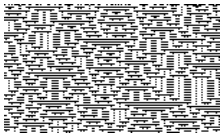


tot 63 (00111111)

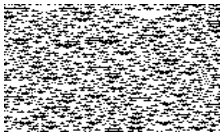


tot 64 (01000000)

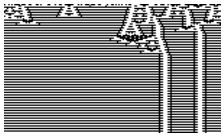
tot 65 (01000001)



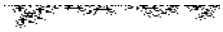
tot 66 (01000010)



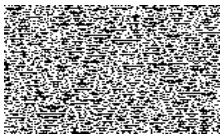
tot 67 (01000011)



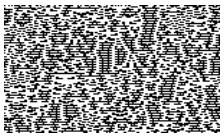
tot 68 (01000100)



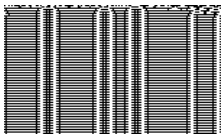
tot 69 (01000101)



tot 70 (01000110)

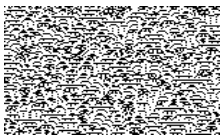


tot 71 (01000111)

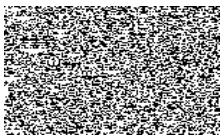


tot 72 (01001000)

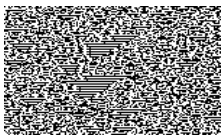
tot 73 (01001001)



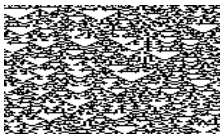
tot 74 (01001010)



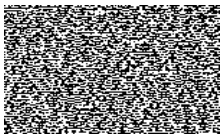
tot 75 (01001011)



tot 76 (01001100)

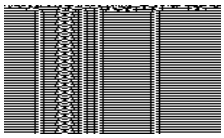


tot 77 (01001101)



tot 78 (01001110)

tot 79 (01001111)

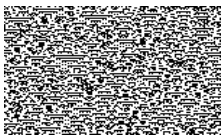


tot 80 (01010000)

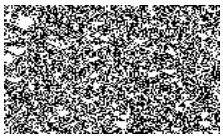
tot 81 (01010001)

tot 82 (01010010)

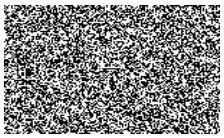
tot 83 (01010011)



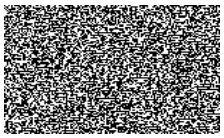
tot 84 (01010100)



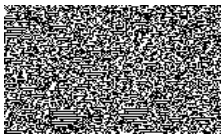
tot 85 (01010101)



tot 86 (01010110)

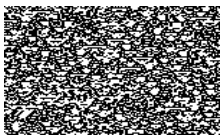


tot 87 (01010111)



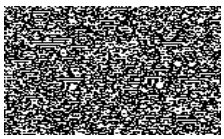
tot 88 (01011000)

tot 89 (01011001)

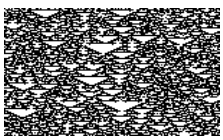


tot 90 (01011010)

tot 91 (01011011)



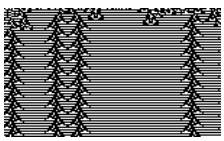
tot 92 (01011100)



tot 93 (01011101)

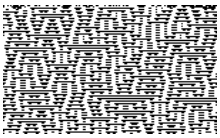
tot 94 (01011110)

tot 95 (01011111)

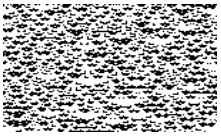


tot 96 (01100000)

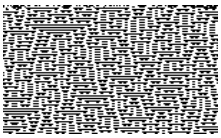
tot 97 (01100001)



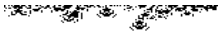
tot 98 (01100010)



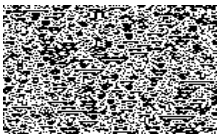
tot 99 (01100011)



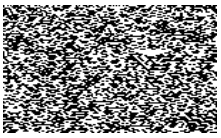
tot 100 (01100100)



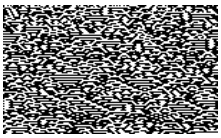
tot 101 (01100101)



tot 102 (01100110)

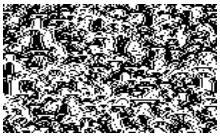


tot 103 (01100111)



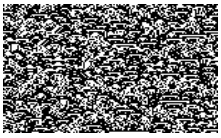
tot 104 (01101000)

tot 105 (01101001)

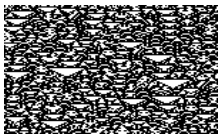


tot 106 (01101010)

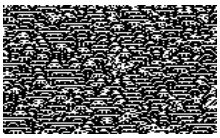
tot 107 (01101011)



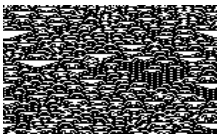
tot 108 (01101100)



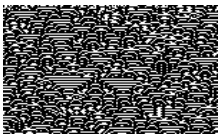
tot 109 (01101101)



tot 110 (01101110)



tot 111 (01101111)

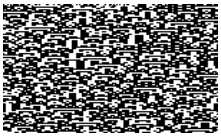


tot 112 (01110000)

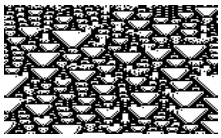
tot 113 (01110001)

tot 114 (01110010)

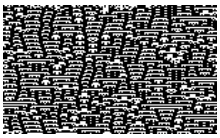
tot 115 (01110011)



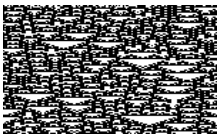
tot 116 (01110100)



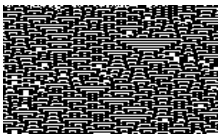
tot 117 (01110101)



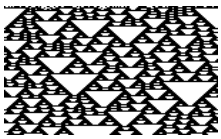
tot 118 (01110110)



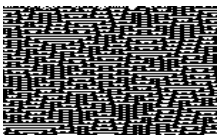
tot 119 (01110111)



tot 120 (01111000)

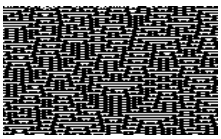


tot 121 (01111001)

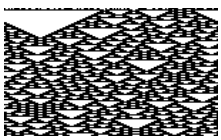


tot 122 (01111010)

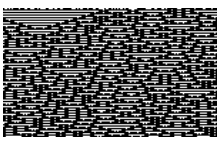
tot 123 (01111011)



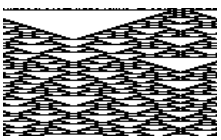
tot 124 (01111100)



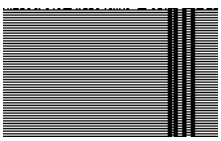
tot 125 (01111101)



tot 126 (01111110)

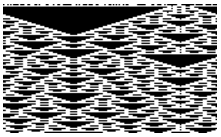


tot 127 (01111111)



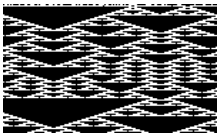
tot 128 (10000000)

tot 129 (10000001)



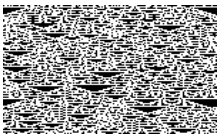
tot 130 (10000010)

tot 131 (10000011)

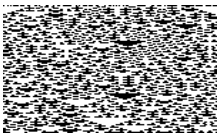


tot 132 (10000100)

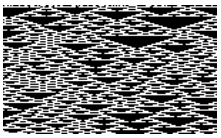
tot 133 (10000101)



tot 134 (10000110)

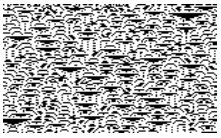


tot 135 (10000111)

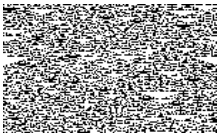


tot 136 (10001000)

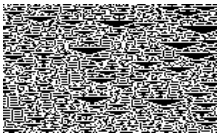
tot 137 (10001001)



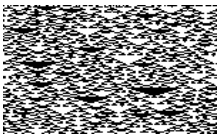
tot 138 (10001010)



tot 139 (10001011)

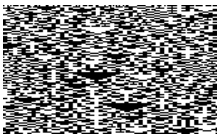


tot 140 (10001100)

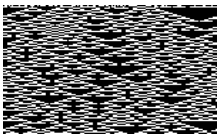


tot 141 (10001101)

tot 142 (10001110)

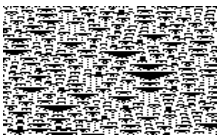


tot 143 (10001111)

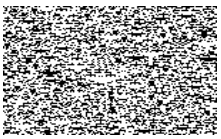


tot 144 (10010000)

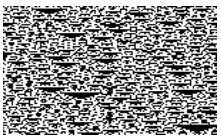
tot 145 (10010001)



tot 146 (10010010)

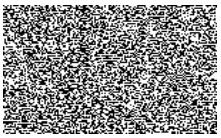


tot 147 (10010011)

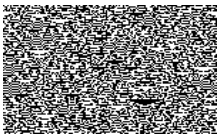


tot 148 (10010100)

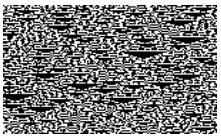
tot 149 (10010101)



tot 150 (10010110)

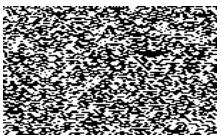


tot 151 (10010111)

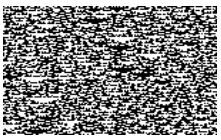


tot 152 (10011000)

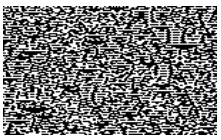
tot 153 (10011001)



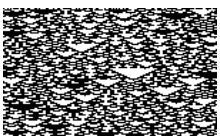
tot 154 (10011010)



tot 155 (10011011)

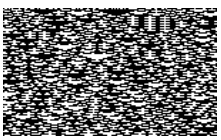


tot 156 (10011100)

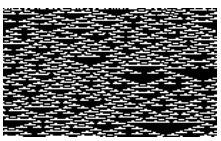


tot 157 (10011101)

tot 158 (10011110)

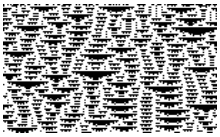


tot 159 (10011111)

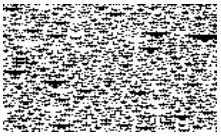


tot 160 (10100000)

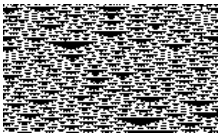
tot 161 (10100001)



tot 162 (10100010)

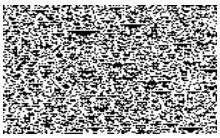


tot 163 (10100011)

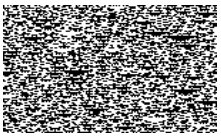


tot 164 (10100100)

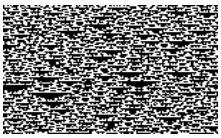
tot 165 (10100101)



tot 166 (10100110)



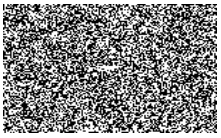
tot 167 (10100111)



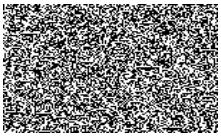
tot 168 (10101000)

tot 169 (10101001)

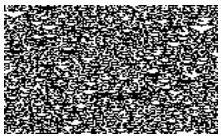
tot 170 (10101010)



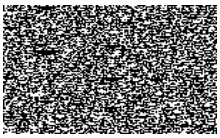
tot 171 (10101011)



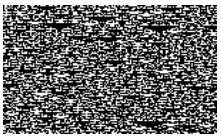
tot 172 (10101100)



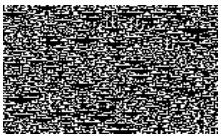
tot 173 (10101101)



tot 174 (10101110)



tot 175 (10101111)



tot 176 (10110000)

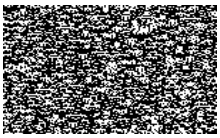
tot 177 (10110001)

tot 178 (10110010)

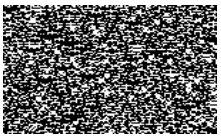
tot 179 (10110011)

tot 180 (10110100)

tot 181 (10110101)

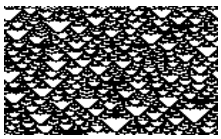


tot 182 (10110110)

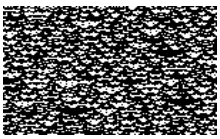


tot 183 (10110111)

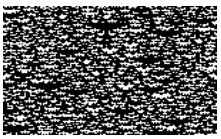
tot 184 (10111000)



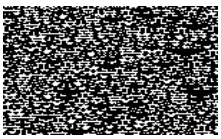
tot 185 (10111001)



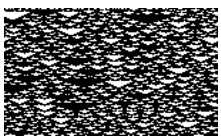
tot 186 (10111010)



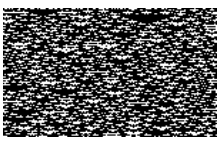
tot 187 (10111011)



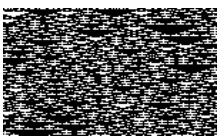
tot 188 (10111100)



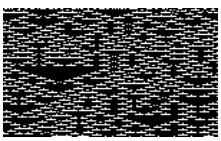
tot 189 (10111101)



tot 190 (10111110)

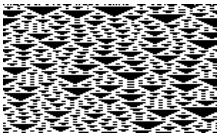


tot 191 (10111111)

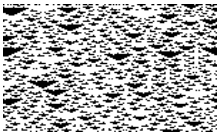


tot 192 (11000000)

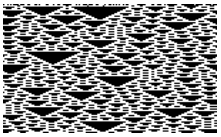
tot 193 (11000001)



tot 194 (11000010)

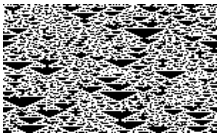


tot 195 (11000011)

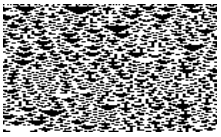


tot 196 (11000100)

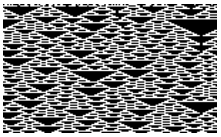
tot 197 (11000101)



tot 198 (11000110)

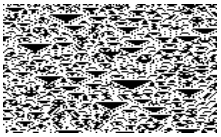


tot 199 (11000111)

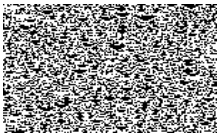


tot 200 (11001000)

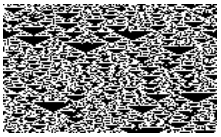
tot 201 (11001001)



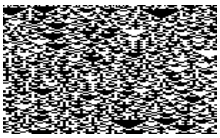
tot 202 (11001010)



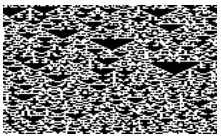
tot 203 (11001011)



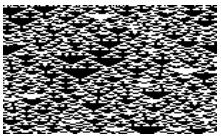
tot 204 (11001100)



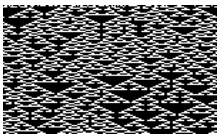
tot 205 (11001101)



tot 206 (11001110)



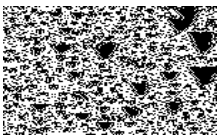
tot 207 (11001111)



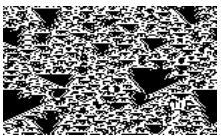
tot 208 (11010000)

tot 209 (11010001)

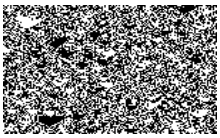
tot 210 (11010010)



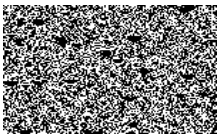
tot 211 (11010011)



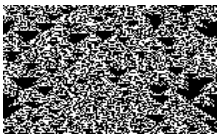
tot 212 (11010100)



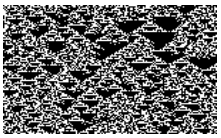
tot 213 (11010101)



tot 214 (11010110)



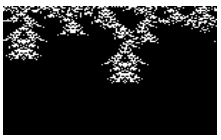
tot 215 (11010111)



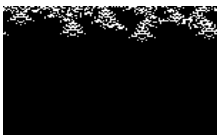
tot 216 (11011000)

tot 217 (11011001)

tot 218 (11011010)



tot 219 (11011011)



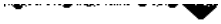
tot 220 (11011100)

tot 221 (11011101)

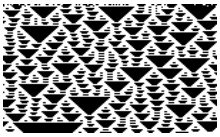
tot 222 (11011110)

tot 223 (11011111)

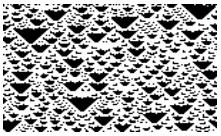
tot 224 (11100000)



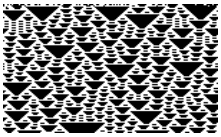
tot 225 (11100001)



tot 226 (11100010)

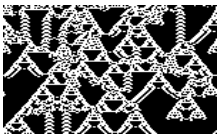


tot 227 (11100011)

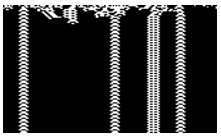


tot 228 (11100100)

tot 229 (11100101)



tot 230 (11100110)



tot 231 (11100111)

tot 232 (11101000)

tot 233 (11101001)

tot 234 (11101010)

tot 235 (11101011)

tot 236 (11101100)

tot 237 (11101101)

tot 238 (11101110)

tot 239 (11101111)

tot 240 (11110000)

tot 241 (11110001)

tot 242 (11110010)

tot 243 (11110011)

tot 244 (11110100)

tot 245 (11110101)

tot 246 (11110110)

tot 247 (11110111)

tot 248 (11111000)

tot 249 (11111001)

tot 250 (11111010)

tot 251 (11111011)

tot 252 (11111100)

tot 253 (11111101)

tot 254 (11111110)

tot 255 (11111111)

Appendix D

MATLAB Source

D.1 Source Index

D.2 Source Code

```
===== File "absActivity.m":
function aa = absActivity( rule, m )

%@@ Calculates absolute activity parameter for a CA rule

% Returns absolute activity parameter defined by Oliveira et al.
% similar to Langton's activity parameter, but defines activity relative to
% the configuration of the neighborhoods.

% rule - binary CA rule
```

```
% m - size of neighborhood (size of neighborhood must be odd, m = 2r + 1, where
%                           r is the neighborhood radius)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% absActivity.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% vectorized - no loops

[ row ruleSize ] = size( rule );
c = ceil(m/2); %center of neighborhood

% calculate number of differences
ruleBin = dec2binV( fliplr( 0:ruleSize-1 ) )';
diffs = abs( repmat( rule, m, 1 ) - ruleBin );
foldedDiffs = [ diffs(c,:); bitor( diffs(1:c-1,:), flipud( diffs(c+1:m,:) ) ) ];
numDiffs = sum(sum(foldedDiffs));

% calculate normalized parameter (find minimum and maximum possible values)

% for efficiency, just use precalculated min and max for elementary rules
% use general case for any other neighborhood size
if( m == 3 )
    aa = (numDiffs - 6) / 8;
else
    diffsZero = ruleBin;
    diffsOne = ~ruleBin;
    foldedDiffsZero = [ diffsZero(c,:); bitor( diffsZero(1:c-1,:), ...
                    flipud( diffsZero(c+1:m,:) ) ) ];
    foldedDiffsOne = [ diffsOne(c,:); bitor( diffsOne(1:c-1,:), ...
                    flipud( diffsOne(c+1:m,:) ) ) ];
    minDiffs = sum( min( sum( foldedDiffsZero ), sum( foldedDiffsOne ) ) );
    maxDiffs = sum( max( sum( foldedDiffsZero ), sum( foldedDiffsOne ) ) );

    aa = (numDiffs - minDiffs) / (maxDiffs - minDiffs);
end
===== End of file "absActivity.m"


===== File "actProp.m":
function ap = actProp( rule, m )

%@@ Calculates activity propigation parameter for a CA rule

% Returns activity propigation parameter defined by Oliveira et al.
% combines ideas of neighborhood dominance and sensitivity

% rule - binary CA rule

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% actProp.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ row ruleSize ] = size( rule );
```

```
c = ceil(m/2); %center of neighborhood

ruleBin = dec2binV( fliplr( 0:ruleSize-1 ) )';
ruleSums = sum( ruleBin );
negSums = [];
negRules = [];
for i = 1:m
    ruleBinNeg = ruleBin;
    ruleBinNeg(i,:) = ~ruleBinNeg(i,:);
    negSums(i,:) = sum( ruleBinNeg );
    negRules(i,:) = rule( fliplr( bin2decV(ruleBinNeg')+1 ) );
end

% calculate non-normalized activity propigation
ruleZeroDom = zeros(1,ruleSize);
ruleZeroDom( find( ruleSums < c & rule == 1 ) ) = 1;
ruleOneDom = zeros(1,ruleSize);
ruleOneDom( find( ruleSums >= c & rule == 0 ) ) = 1;

negZeroDom = zeros(m,ruleSize);
negZeroDom( find( negSums < c & negRules == 1 ) ) = 1;
negOneDom = zeros(m,ruleSize);
negOneDom( find( negSums >= c & negRules == 0 ) ) = 1;

truths = repmat( ( ruleZeroDom | ruleOneDom ), m, 1 ) ...
           & ( negZeroDom | negOneDom );
ap = sum(sum(truths));

% normalize
ap = ap / (ruleSize*m);
===== End of file "actProp.m"


===== File "allCharts.m":
function [] = allCharts( dir )

%@@ Displays a bar chart for each paramter

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% allCharts.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

vals = testElementary([1 1 1 1 1 1 1]);
barChartParam( vals(1,:), '\lambda - Activity', [ dir 'discrimL.eps' ] )
barChartParam( vals(2,:), 'Z', [ dir 'discrimZ.eps' ]  )
barChartParam( vals(3,:), '\mu - Sensitivity', [ dir 'discrimM.eps' ]  )
barChartParam( vals(4,:), 'Absolute Activity', [ dir 'discrimAA.eps' ] )
barChartParam( vals(5,:), 'Neighborhood Dominance', [ dir 'discrimND.eps' ] )
barChartParam( vals(6,:), 'Activity Propigation', [ dir 'discrimAP.eps' ] )
barChartParam( vals(7,:), '\upsilon - Incompressability', ...
               [ dir 'discrimU.eps' ] )
===== End of file "allCharts.m"


===== File "allParams.m":
function vals = allParams( rules, m )
```

```
%@@ Returns all parameter values for each of the rules given.

% rules - binary CA rule(s)
% m - neighborhood size

% vals - each row is a CA rule, each column a parameter

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% allParams.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ numRules, ruleLen ] = size(rules);
for i = 1:numRules
    r = rules(i,:);
    vals(i,:) = [ lambda(r), ...
                  z(r,m), ...
                  mu(r,m), ...
                  absActivity(r,m), ...
                  neighDom(r,m), ...
                  actProp(r,m), ...
                  upsilon(r,m) ];
end
===== End of file "allParams.m"


===== File "barChartParam.m":
function [] = barChartParam( paramValues, pTitle, fileName )

%@@ Plot class bar charts for each of the parameters

% Displays a bar chart for the 6 Li-Packard dynamic classes of the parameter
% values given by paramValues for the 256 elementary CA

% will save a color eps version of figure to fileName (no file is saved
% if fileName is '')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% barChartParam.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ c1, c2, c3, c4, c5, c6 ] = classes;
X = sort(unique(paramValues));
[ r numValues ] = size(X);
Y = zeros(numValues, 6);

for i = 1:numValues
    for j = 1:6
        Y(i,j) = sum( paramValues(eval(['c' num2str(j)])+1) == X(i) );
    end
end

figure;
%bar( X, Y, 1.5 );
```

```
bar( X, Y, 'stacked' );
%colormap(lines);
%axis( [ -.1 1.1 0 max(max(Y)) ] );
xlabel('Parameter Value');
set( gca, 'XTick', X );
set( gca, 'XTickLabel', sprintf('%0.2g|', X) );
ylabel('Number of Elementary CA');
legend('Null','Fixed Point','Two-Cycle','Periodic','Complex','Chaotic');
title(pTitle);

if(~strcmp(fileName,''))
    print( '-depsc', fileName );
end
===== End of file "barChartParam.m"


===== File "base2decV.m":
function dec = base2decV ( baseNum, base )

%@@ Convert a base n vector to a decimal number

% Returns a matrix containing the deciaml equivalents of base n number(s).
% Similar to MATLAB built-in base2dec, but takes a matrix of numbers in the
% range [0 n) instead of a string

% baseNum - a 1D or 2D matrix of numbers in range [0 n). If baseNum is 1D a
% single decimal number is returned. If baseNum is 2D, a 1D vector of decimal
% numbers is returned

% base - the base of the given number, in the range [2 10]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% base2decV.m
% Dan Kunkle
% November 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ r, c ] = size( baseNum );
pow = base .^ ( (ones(r,1) * (c-[1:c]) ) );
dec = sum( baseNum .* pow, 2 )';
===== End of file "base2decV.m"


===== File "bin2decV.m":
function dec = bin2decV ( bin )

%@@ Convert a binary vector to a decimal number

% Returns a matrix containing the deciaml equivalents of binary number(s).
% Similar to MATLAB built-in bin2dec, but takes a matrix of 1s and 0s instead
% of a string

% bin - a 1D or 2D matrix of 1s and 0s. If bin is 1D a single decimal number is
%       returned. If bin is 2D, a 1D vector of decimal numbers is returned

% example: bin2decV( [ 1 0 1 0 ] ) returns 10
%          bin2decV( [ 1 0 1 0; 0 1 0 1 ] ) returns [ 10 5 ]
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% bin2decV.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ r, c ] = size( bin );
pow = pow2( (ones(r,1) * (c-[1:c]) ) );
dec = sum( bin .* pow, 2 )';
===== End of file "bin2decV.m"


===== File "bin2gray.m":
function gray = bin2gray( bin )

%@@ Converts binary numbers to a gray codes

% bin - each row is a binary number (vector of 1s and 0s)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% bin2gray.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ row col ] = size( bin );
gray(:,1) = bin(:,1);
gray(:,2:col) = xor( [ bin(:,1:col-1) ], bin(:,2:col) );
===== End of file "bin2gray.m"


===== File "ca1d.m":
function timeSpace = ca1d ( initM, m, rule, timeSteps )

%@@ Simulates a one-dimensional, k=2, CA

% initM: the initial CA matrix
% m:     size of neighborhood (ONLY WORK FOR 3, 5 and 7 FOR NOW)
% rule: rule governing the state changes of each cell in the CA (all cells
%       have same rules). Rule must be of length 2^(2^m) where m is the size
%       of the neighborhood.
%       example: n = 3, rule is t7,t6,...,t0 where t7 corresponds to
%                neighborhood (111), t6 for (110), ..., t0 for (000)
% timeSteps: number of time steps to execute the CA for (includes init state)
%
% returns -- a 2D matrix representing the evolution of the CA over time.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ca1d.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% for now, just do it the easier way with n = 3, 5 or 7

timeSpace = initM;

% size of CA
```

```
[ x, n ] = size( initM );
[ x, ruleSize ] = size( rule );

% left3, left2, left1, center, right1, right2, right3
l3 = [ n-2:n 1:n-3 ];
l2 = [ n-1:n 1:n-2 ];
l1 = [ n 1:n-1 ];
c  = [ 1:n ];
r1 = [ 2:n 1 ];
r2 = [ 3:n 1:2 ];
r3 = [ 4:n 1:3 ];

for i = 2:timeSteps
    % find neighborhoods (based on size, m )
    if ( m == 3 )
        neighbors = [ timeSpace(i-1, l1)' timeSpace(i-1, c)' ...
                      timeSpace(i-1, r1)' ];
    elseif ( m == 5 )
        neighbors = [ timeSpace(i-1, l2)' timeSpace(i-1, l1)' ...
                      timeSpace(i-1, c)' timeSpace(i-1, r1)' ...
                      timeSpace(i-1, r2)' ];
    elseif ( m == 7 )
        neighbors = [ timeSpace(i-1, l3)' timeSpace(i-1, l2)' ...
                      timeSpace(i-1, l1)' timeSpace(i-1, c)'...
                      timeSpace(i-1, r1)' timeSpace(i-1, r2)' ...
                      timeSpace(i-1, r3)' ];
    end

    % update based on rule
    timeSpace(i,:) = rule( ruleSize - bin2decV( neighbors ) );
end

%TESTING
%iptsetpref( 'ImshowBorder', 'tight' );
%imshow( ~timeSpace, [0 1] );
===== End of file "ca1d.m"


===== File "ca1dk.m":
function timeSpace = ca1dk ( initM, m, k, rule, timeSteps )

%@@ Simulates a one-dimensional CA with num. states k in the range [0 10]

% initM: the initial CA matrix
% m:     size of neighborhood (ONLY WORK FOR 3, 5 and 7 FOR NOW)
% k:     the number of states (colors)
% rule: rule governing the state changes of each cell in the CA (all cells
%       have same rules). Rule must be of length 2^(2^m) where m is the size
%       of the neighborhood.
%       example: n = 3, rule is t7,t6,...,t0 where t7 corresponds to
%                neighborhood (111), t6 for (110), ..., t0 for (000)
% timeSteps: number of time steps to execute the CA for (includes init state)
%
% returns -- a 2D matrix representing the evolution of the CA over time.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ca1dk.m
% Dan Kunkle
```

```
% November 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% for now, just do it the easier way with n = 3, 5 or 7

timeSpace = initM;

% size of CA
[ x, n ] = size( initM );
[ x, ruleSize ] = size( rule );

% left3, left2, left1, center, right1, right2, right3
l3 = [ n-2:n 1:n-3 ];
l2 = [ n-1:n 1:n-2 ];
l1 = [ n 1:n-1 ];
c  = [ 1:n ];
r1 = [ 2:n 1 ];
r2 = [ 3:n 1:2 ];
r3 = [ 4:n 1:3 ];

for i = 2:timeSteps
    % find neighborhoods (based on size, m )
    if ( m == 3 )
        neighbors = [ timeSpace(i-1, l1)' timeSpace(i-1, c)' ...
                      timeSpace(i-1, r1)' ];
    elseif ( m == 5 )
        neighbors = [ timeSpace(i-1, l2)' timeSpace(i-1, l1)' ...
                      timeSpace(i-1, c)' timeSpace(i-1, r1)' ...
                      timeSpace(i-1, r2)' ];
    elseif ( m == 7 )
        neighbors = [ timeSpace(i-1, l3)' timeSpace(i-1, l2)' ...
                      timeSpace(i-1, l1)' timeSpace(i-1, c)'...
                      timeSpace(i-1, r1)' timeSpace(i-1, r2)' ...
                      timeSpace(i-1, r3)' ];
    end

    % update based on rule
    timeSpace(i,:) = rule( ruleSize - base2decV( neighbors, k ) );
end

%TESTING
%iptsetpref( 'ImshowBorder', 'tight' );
%imshow( ~timeSpace, [0 1] );
===== End of file "ca1dk.m"


===== File "caScreenSaver.m":
function [] = caScreenSaver()

%@@ Saves sets of CA pictures for use with screen saver

% Saves some pretty pictures to file for use with my screen saver

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% caScreenSaver.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
base = '/tmp/pics/';

eRules = [ 147 110 129 ]; % elementary rules
eRulesBin = dec2binV( eRules, 8 );
tRules = [ 20 25 88 116 140 229 157 ]; % totalistic k=2 r=3 rules
tRulesBin = tot2full( dec2binV( tRules, 8 ) );

numEach = 5; % number of images of each rule to make
h = 640;  % height
w = 960; % width

[r numE ] = size(eRules);
[r numT ] = size(tRules);
for i = 1:numEach

    map = [ 0 0 0; 1 1 1 ];
    for j = 1:numE
        pic = ca1d( rand(1,w) > .5, 3, eRulesBin(j,:), h);
        fn = [ base 'e' num2str( eRules(j) ) '_' num2str(i) '.tiff' ];
        imwrite(pic+1, map, fn, 'tiff');
    end

    for j = 1:numT
        pic = ca1d( rand(1,w) > .5, 7, tRulesBin(j,:), h);
        fn = [ base 't' num2str( tRules(j) ) '_' num2str(i) '.tiff' ];
        imwrite(pic+1, map, fn, 'tiff');
    end

end
===== End of file "caScreenSaver.m"


===== File "classes.m":
function [ null, fixedPoint, twoCycle, periodic, complex, chaotic ] ...
           = classes()

%@@ Li-Packard class for each elementary CA

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% classes.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

null = [0, 255, 8, 64, 239, 253, 32, 251, 40, 96, 235, 249, 128, 254, ...
        136, 192, 238, 252, 160, 250, 168, 224, 234, 248 ];

fixedPoint = [2, 16, 191, 247, 4, 223, 10, 80, 175, 245, 12, 68, 207, 221, ...
              13, 69, 79, 93, 24, 66, 189, 231, 34, 48, 187, 243, 36, 219, ...
              42, 112, 171, 241, 44, 100, 203, 217, 46, 116, 139, 209, ...
              56, 98, 185, 227, 57, 99, 58, 114, 163, 177, 72, 237, ...
              76, 205, 77, 78, 92, 141, 197, 104, 233, 130, 144, 190, 246, ...
              132, 222, 138, 174, 208, 244, 140, 196, 206, 220, 152, 188, ...
              194, 230, 162, 176, 186, 242, 164, 218, 170, 240, 172, 202, ...
              216, 228, 184, 226, 200, 236, 204, 232 ];

twoCycle = [1, 127, 3, 17, 63, 119, 5, 95, 6, 20, 159, 215, 7, 21, 31, 87, ...
```

```
            9, 65, 111, 125, 11, 47, 81, 117, 14, 84, 143, 213, 15, 85, ...
            19, 55, 23, 25, 61, 67, 103, 27, 39, 53, 83, 28, 70, 157, 199, ...
            29, 71, 33, 123, 35, 49, 59, 115, 37, 91, 38, 52, 155, 211, ...
            43, 113, 50, 179, 51, 74, 88, 173, 229, 108, 201, ...
            134, 148, 158, 214, 142, 212, 156, 198, 178 ];

periodic = [26, 82, 167, 181, 41, 97, 107, 121, 62, 118, 131, 145, 94, 133, ...
            154, 166, 180, 210 ];

complex = [54, 147, 110, 124, 137, 193 ];

chaotic = [18, 183, 22, 151, 30, 86, 135, 149, 45, 75, 89, 101, 60, 102, ...
           153, 195, 73, 109, 90, 165, 105, 106, 120, 169, 225, 122, 161, ...
           126, 129, 146, 182, 150 ];

%TESTING
%nullSize = size(null)
%fixedPointSize = size(fixedPoint)
%twoCycleSize = size(twoCycle)
%periodicSize = size(periodic)
%complexSize = size(complex)
%chaoticSize = size(chaotic)

%union = [ null fixedPoint twoCycle periodic complex chaotic ];
%unionSize = size( union )
%uniqueUnion = unique( union )
%uniqueUnionSize = size( uniqueUnion )
===== End of file "classes.m"


===== File "classesCombined.m":
function c = classesCombined()

%@@ Li-Packard class for each elementary CA in one matrix

% Returns a 1x256 vector specifying the Li-Packard class of each of the 256
% elementary CA. 1 = null, 2 = fixed point, 3 = two-cycle, 4 = periodic,
% 5 = complex, 6 = chaotic

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% classesCombined.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% get indicies for each of the six classes
[ null, fixedPoint, twoCycle, periodic, complex, chaotic ] = classes;
c = zeros(1,256);
c(null+1) = 1;
c(fixedPoint+1) = 2;
c(twoCycle+1) = 3;
c(periodic+1) = 4;
c(complex+1) = 5;
c(chaotic+1) = 6;
===== End of file "classesCombined.m"


===== File "classesCombinedTot.m":
```

```
function c = classesCombinedTot()

%@@ Li-Packard class for each totalistic k=2 r=3 CA in one matrix

% Returns a 1x256 vector specifying the Li-Packard class of each of the 256
% totalistic k=2 r=3 CA. 1 = null, 2 = fixed point, 3 = two-cycle, 4 = periodic,
% 5 = complex, 6 = chaotic

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% classesCombinedTot.m
% Dan Kunkle
% June 2003
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% get indicies for each of the six classes
[ null, fixedPoint, twoCycle, periodic, complex, chaotic ] = classesTot;
c = zeros(1,256);
c(null+1) = 1;
c(fixedPoint+1) = 2;
c(twoCycle+1) = 3;
c(periodic+1) = 4;
c(complex+1) = 5;
c(chaotic+1) = 6;
===== End of file "classesCombinedTot.m"


===== File "classesTargets.m":
function c = classesTargets()

%@@ Li-Packard class of each elementary CA in NN target form

% Returns a 6x256 vector specifying the Li-Packard class of each of the 256
% elementary CA. Each column represents a CA, each row a class. There is a 1 in
% each cell where the CA rule and the classification match, a 0 other wise.
% Rows and classes match up as follows: 1 = null, 2 = fixed point, 3 =
% two-cycle, 4 = periodic, 5 = complex, 6 = chaotic

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% classesTargets.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% get indicies for each of the six classes
[ null, fixedPoint, twoCycle, periodic, complex, chaotic ] = classes;
c = zeros(6,256);
c(1,null+1) = 1;
c(2,fixedPoint+1) = 1;
c(3,twoCycle+1) = 1;
c(4,periodic+1) = 1;
c(5,complex+1) = 1;
c(6,chaotic+1) = 1;
===== End of file "classesTargets.m"


===== File "classesTargetsTot.m":
function c = classesTargetsTot()
```

```
%@@ Li-Packard class of each totalistic k=2 r=3 CA in NN target form

% Returns a 6x256 vector specifying the Li-Packard class of each of the 256
% totalistic k=2 r=3 CA. Each column represents a CA, each row a class.  There
% is a 1 in each cell where the CA rule and the classification match, a 0 other
% wise.  Rows and classes match up as follows: 1 = null, 2 = fixed point, 3 =
% two-cycle, 4 = periodic, 5 = complex, 6 = chaotic

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% classesTargets.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% get indicies for each of the six classes
[ null, fixedPoint, twoCycle, periodic, complex, chaotic ] = classesTot;
c = zeros(6,256);
c(1,null+1) = 1;
c(2,fixedPoint+1) = 1;
c(3,twoCycle+1) = 1;
c(4,periodic+1) = 1;
c(5,complex+1) = 1;
c(6,chaotic+1) = 1;
===== End of file "classesTargetsTot.m"


===== File "classesTot.m":
function [ null, fixedPoint, twoCycle, periodic, complex, chaotic ] ...
            = classesTot()

%@@ Listing of Li-Packard class for each totalistic k=2 r=3 CA

% This classification was done by hand (and eye), which may or may not be more
% reliable than the trained NN.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% classesTot.m
% Dan Kunkle
% June 2003
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

null = [0, 255, 4, 223, 32, 251, 36, 219, 64, 253, 68, 221, 72, 237, 80, ...
        245, 96, 249, 100, 217, 124, 193, 128, 254, 132, 222, 136, 238, ...
        144, 246, 160, 250, 164, 218, 192, 252, 196, 220, 200, 236, 208, ...
        244, 224, 248];

fixedPoint = [8, 239, 16, 247, 40, 235, 48, 243, 104, 233, 112, 241, 168, ...
              234, 176, 242, 216, 228, 232, 240];

twoCycle = [1, 127, 3, 63, 7, 31, 11, 47, 15, 23, 27, 39, 35, 59, 61, 67];

periodic = [2, 191, 5, 95, 6, 159, 13, 79, 19, 55, 24, 231, 29, 71, 62, 131 ...
            126, 129, 152, 230];

complex = [88, 229];

chaotic = [9, 111, 10, 175, 12, 207, 14, 143, 17, 119, 18, 183, 20, 215, ...
           21, 87, 22, 151, 25, 103, 26, 167, 28, 199, 30, 135, 33, 123, ...
```

```
          34, 187, 37, 91, 38, 155, 41, 107, 42, 171, 43, 44, 203, 45, 75, ...
          46, 139, 49, 115, 50, 179, 51, 52, 211, 53, 83, 54, 147, 56, ...
          227, 57, 99, 58, 163, 60, 195, 65, 125, 66, 189, 69, 93, 70, ...
          157, 73, 109, 74, 173, 76, 205, 77, 78, 141, 81, 117, 82, 181, ...
          84, 213, 85, 86, 149, 89, 101, 90, 165, 92, 197, 94, 133, 97, ...
          121, 98, 185, 102, 153, 105, 106, 169, 108, 201, 110, 137, 113, ...
          114, 177, 116, 209, 118, 145, 120, 225, 122, 161, 130, 190, 134, ...
          158, 138, 174, 140, 206, 142, 146, 182, 148, 214, 150, 154, 166, ...
          156, 198, 162, 186, 170, 172, 202, 178, 180, 210, 184, 226, 188, ...
          194, 204, 212];

%TESTING
%nullSize = size(null)
%fixedPointSize = size(fixedPoint)
%twoCycleSize = size(twoCycle)
%periodicSize = size(periodic)
%complexSize = size(complex)
%chaoticSize = size(chaotic)

%union = [ null fixedPoint twoCycle periodic complex chaotic ];
%unionSize = size( union )
%uniqueUnion = unique( union )
%uniqueUnionSize = size( uniqueUnion )
===== End of file "classesTot.m"


===== File "classify.m":
function [class, percentCor, percentCorByClass] = ...
    classify( net, type, fileName, pretty )

%@@ Classify totalistic k=2 r=3 CA using a trained NN

% Returns classifications of the 256 totalistic k=2 r=3 CA using the given
% neual net.

% net - pre-trained neural network
% type - either 'elem' or 'tot' for elementary CA or totalistc k=2 r=3 CA
% fileName - if a file name is given the results will be saved
% pretty - if pretty=1 data will be saved with extra info

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% classify.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if( strcmp(type, 'elem') )
    vals = readElementaryParams'; % parameter values for each elementary CA
    vals = vals(1:7,:);
    classByHand = classesCombined; % hand constucted classification of elem
else
    vals = readTotParams'; % parameter values for each tot k=2 r=3 CA
    classByHand = classesCombinedTot;% hand constucted classification of tot
end
class = sim( net, vals )'; % simulate network
[ numInputs numTrials ] = size( vals );

%count how many net got right
```

```
count = 0;
numPerClass = zeros(1,6);
classCount = zeros(1,6);
classRound = round(class*100);
for i = 1:numTrials
    %maximum net output value
    maxOutVal = max(classRound(i,:));
    %set of outputs with maximum value
    maxOuts = find(classRound(i,:) == maxOutVal);

    numPerClass(classByHand(i)) = numPerClass(classByHand(i)) + 1;
    if( find(maxOuts == classByHand(i)) )
        count = count + 1;
        classCount(classByHand(i)) = classCount(classByHand(i)) + 1;
    end
end
percentCor = count/numTrials;
percentCorByClass = classCount./numPerClass;

% save results
if( ~strcmp(fileName, '') )
    fid = fopen( fileName, 'w' );
    linePre = '';
    for i = 1:numTrials
        if( pretty )
            linePre = [num2str(i-1) ' & '];
        end
        fprintf( fid, linePre, '' );
        for j = 1:6
            if class(i,j) > .3
                template = '\\textbf{%3.2f}';
            else
                template = '%3.2f';
            end
            if classByHand(i) == j
                template = ['\\cc{' template '}'];
            end
            if (j < 6)
                template = [template ' & '];
            end
            fprintf( fid, [template], class(i,j) );
        end
        fprintf( fid, '\\\\ \\hline \n', '' );
    end
    fclose(fid);
end
===== End of file "classify.m"


===== File "clusterDist.m":
function [intra, inter] = clusterDist( classPoints )

%@@ Mean intra and inter-cluster distances for classes of points

% classPoints - a cell array containing matricies of n-dimensional points
%               partitioned into classes. Each matrix in the cell array
%               represents a class, each column in the matrix is a point,
%               and each row is a dimension
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% clusterDist.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ rows numClasses ] = size( classPoints ); % number of classes
[ numDims cols ] = size( cell2mat(classPoints(1)) ); % number of dimensions

centroids = zeros( numDims, numClasses ); % centroids of each class
intraDists = zeros( 1, numClasses ); % mean dists from points to centroid

% find centroids and intra-cluster distances for each class
for i = 1:numClasses
    [ rows numPoints ] = size( cell2mat(classPoints(i)) );
    centroids(:,i) = mean(cell2mat(classPoints(i)), 2);
    allIntraDists = dist([centroids(:,i) cell2mat(classPoints(i))]);
    intraDists(i) = mean(allIntraDists(2:numPoints,1));
end

% mean intra-cluster distance over all classes
intra = mean(intraDists);

% find mean inter-cluster distance over all classes (mean centroid distance)
allInterDists = dist(centroids);
lowerTri = tril(ones(numClasses,numClasses), -1); % 1's in lower triangle
inter = mean(allInterDists(find(lowerTri))); % mean of lower triangle of dists
===== End of file "clusterDist.m"


===== File "dec2baseV.m":
function baseNum = dec2baseV( dec, base, digits )

%@@ Convert decimal numbers to base n vector form

% Returns a matrix containing the base n equivalents of decimal number(s)
% Similar to MATLAB built-in dec2base, but returns a matrix of numbers in the
% range [0 n) instead of a string

% dec - a single or vector of decimal numbers
% base - the base of the resulting number
% digits - the minimum number of digits in the resulting number

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% dec2baseV.m
% Dan Kunkle
% November 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n = max(1,round(log2(max(dec)+1)/log2(base))); % number of digits needed for
                                               % largest dec
while any(base.^n <= dec)
   n = n + 1;
end
if nargin == 3
   n = max(n,digits); % number of digits in baseNum
end
```

```
dec = dec(:); % make dec a column vector
baseNum = rem( floor( dec * base.^( 1-n:0 ) ), base );
===== End of file "dec2baseV.m"


===== File "dec2binV.m":
function bin = dec2binV( dec, digits )

%@@ Convert decimal numbers to binary-vector form

% Returns a matrix containing the binary equivalents of decimal number(s)
% Similar to MATLAB built-in dec2bin, but returns a matrix of 1s and 0s
% instead of a string

% dec - a single or vector of decimal numbers
% digits - the minimum number of digits in the resulting binary number
% bin - a vector of 0's and 1's representing a binary number. If there are
%       multiple decimal numbers given each row of bin is another binary
%       number

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% dec2binV.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ f, e ] = log2( max( dec ) ); % the number of digits needed to represent the
                               % the largest number
if (nargin<2)
  digits=1; % Need at least one digit even for 0.
end
n = max( digits, e ); %number of digits in bin
dec = dec(:); % make dec a column vector
bin = rem( floor( dec * pow2( 1-n:0 ) ), 2 );
===== End of file "dec2binV.m"


===== File "diffPattern.m":
function dp = diffPattern( initM, m, rule, timeSteps, numFlips )

%@@ Generate CA difference patterns

% Returns the difference pattern between the evolution of a rule and the
% evolution of the same rule with the inital state modified slightly.

% initM: the initial CA matrix
% m:     size of neighborhood (ONLY WORK FOR 3, 5 and 7 FOR NOW)
% rule: rule governing the state changes of each cell in the CA (all cells
%       have same rules). Rule must be of length 2^(2^m) where m is the size
%       of the neighborhood.
%       example: n = 3, rule is t7,t6,...,t0 where t7 corresponds to
%                neighborhood (111), t6 for (110), ..., t0 for (000)
% timeSteps: number of time steps to execute the CA for (includes init state)
% numFlips: the number of bits in the inital matrix to be inverted. All flips
%           are adjacent and occur in the middle of the initial configuration
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% diffPattern.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ rows, n ] = size( initM );

initM1 = initM;
initM2 = initM;

% flip bits
mp = ceil( n/2 ); %mid-point
nL = ceil((numFlips-1)/2); %number of bits to flip left of center
nR = floor((numFlips-1)/2); %number of bits to flip right of center
initM2(mp-nL:mp+nR) = ~initM2(mp-nL:mp+nR);

timeSpace1 = ca1d( initM1, m, rule, timeSteps );
timeSpace2 = ca1d( initM2, m, rule, timeSteps );

dp = (timeSpace1 ~= timeSpace2);
===== End of file "diffPattern.m"


===== File "diffPatternOverlay.m":
function dpo = diffPatternOverlay( initM, m, rule, timeSteps, numFlips )

%@@ Overlay CA difference patterns on original CA evolution

% Returns the difference pattern between the evolution of a rule and the
% evolution of the same rule with the inital state modified slightly. Overlayed
% on the original evolution. Orignal pattern is represented by 0 and 0.5
% values, difference pattern is represented by 1 values.

% initM: the initial CA matrix
% m:     size of neighborhood (ONLY WORK FOR 3, 5 and 7 FOR NOW)
% rule: rule governing the state changes of each cell in the CA (all cells
%       have same rules). Rule must be of length 2^(2^m) where m is the size
%       of the neighborhood.
%       example: n = 3, rule is t7,t6,...,t0 where t7 corresponds to
%                neighborhood (111), t6 for (110), ..., t0 for (000)
% timeSteps: number of time steps to execute the CA for (includes init state)
% numFlips: the number of bits in the inital matrix to be inverted. All flips
%           are adjacent and occur in the middle of the initial configuration


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% diffPatternOverlay.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ rows, n ] = size( initM );

initM1 = initM;
initM2 = initM;

% flip bits
```

```
mp = ceil( n/2 ); %mid-point
nL = ceil((numFlips-1)/2); %number of bits to flip left of center
nR = floor((numFlips-1)/2); %number of bits to flip right of center
initM2(mp-nL:mp+nR) = ~initM2(mp-nL:mp+nR);

timeSpace1 = ca1d( initM1, m, rule, timeSteps );
timeSpace2 = ca1d( initM2, m, rule, timeSteps );

dp = (timeSpace1 ~= timeSpace2);
dpo = timeSpace1/3;
dpo(find(dp==1)) = 1;
===== End of file "diffPatternOverlay.m"


===== File "equivTot.m":
function eq = equivTot( display )

%@@ Returns a cell array of all equiv. groups of k=2 r=3 tot. CA

% display - if == 1 will display to command line

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% equivTot.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

equivGroups = zeros(1,256) - 1; % -1 if only rule i is in group, -2 if rule i
                                % is in a diff. group, any other number secifies
                                % other rule in group

rules = dec2binV(0:255);
for i = 1:256
    negatedRule = bin2decV( negate( rules(i,:) ) );
    if( (negatedRule ~= i-1) && ( i-1 < negatedRule ) )
        equivGroups(i) = negatedRule;
        equivGroups(negatedRule+1) = -2;
    end
end

% construct cell array
eq = {};
eqCount = 1;
for i = 1:256
    otherRule = equivGroups(i);
    if( otherRule ~= -2 )
        if( otherRule == -1 )
            eq(eqCount) = { [ i-1 ] };
        else
            eq(eqCount) = { [ i-1 otherRule ] };
        end
        eqCount = eqCount + 1;
    end
end

% display
if(display)
    disp( 'EQUIVALENT RULE GROUPS - TOTALISTIC K=2 R=3' );
```

```
    for i = 1:eqCount-1
        disp( mat2str(cell2mat(eq(i))) );
    end
end
===== End of file "equivTot.m"


===== File "equivTotPretty.m":
function [] = equivTotPretty( filename )

%@@ Prints a pretty LaTeX table of tot k=2 r=3 CA rule dynamics

% filename - prints to filename if given, to screen otherwise

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% equivTot.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

equivGroups = equivTot(0);
numGroups = length(equivGroups);
classNums = classesCombinedTot;
classNames = {'Null', 'Fixed Point', 'Two-Cycle', 'Periodic', 'Complex', ...
              'Chaotic'};

% display
if(~strcmp(filename, ''))
    fid = fopen( filename, 'w' );
else
    fid = 1;
end

columns = 3;
rows = ceil(numGroups/columns);
for i = 1:rows
    for j = 1:columns
        groupNum = (j-1)*rows + i;
        if groupNum <= numGroups
            group = equivGroups{groupNum};
            numInGroup = length(group);
            r1 = num2str(group(1));
            if numInGroup > 1
                r2 = num2str(group(2));
            else
                r2 = '';
            end
            cl = classNames{classNums(group(1)+1)};
            fprintf(fid, ['\\textbf{' r1 '} & ' r2 ' & ' cl]);
        else
            fprintf(fid, '& &');
        end
        if j < columns
            fprintf(fid, ' & ');
        end
    end
    fprintf(fid, '\\\\ \\hline \n');
end
```

```
===== End of file "equivTotPretty.m"


===== File "getClass.m":
function class = getClass( rule, shortName )

%@@ Returns the name of the class of the given CA

% rule - decimal index of elementary CA rule
% shortName - if shortName = 1 a one or two character abbreviation is used

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% getClass.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classes = classesCombined;
classNum = classes( rule+1 );
if ~shortName
  switch classNum
    case 1
      class = 'Null';
    case 2
      class = 'Fixed-Point';
    case 3
      class = 'Two-Cycle';
    case 4
      class = 'Periodic';
    case 5
      class = 'Complex';
    case 6
      class = 'Chaotic';
  end
else
  switch classNum
    case 1
      class = 'N';
    case 2
      class = 'FP';
    case 3
      class = 'TC';
    case 4
      class = 'P';
    case 5
      class = 'CO';
    case 6
      class = 'CH';
  end
end
===== End of file "getClass.m"


===== File "gray2bin.m":
function bin = gray2bin( gray )

%@@ Convert gray code numbers to a binary numbers
```

```
% gray - each row is a gray code (vector of 1s and 0s)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% gray2bin.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ row col ] = size( gray );
bin = rem(cumsum(gray,2),2);
===== End of file "gray2bin.m"


===== File "grayGen.m":
function grayCodes = grayGen( maxVals )

%@@ Generates a generalized gray code

% maxVals - a vector specifying the maximum value for each digit

% Adapted from Dah-Jyh Guan, "Generalized Gray Codes with Applications", 1998

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% grayGen.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% possibly try to vectorize this

[ r numBits ] = size(maxVals);
numCodes = prod(maxVals);
grayCodes = zeros(numCodes, numBits);
n = [ maxVals 2 ];
g = zeros(1,numBits+1); %last gray code
u = ones(1,numBits+1); %keep track of incrementing or decrementing bits

count = 0;
while (g(numBits+1)==0)
    count = count + 1;
    grayCodes(count,:) = g(1,1:numBits);

    i = 1;
    k = g(1)+u(1);
    while( (k>=n(i) | (k<0) ) )
       u(i) = -u(i);
       i = i + 1;
       k = g(i) + u(i);
    end
    g(i) = k;
end
===== End of file "grayGen.m"


===== File "grid.m":
function g = grid( M, n )

%@@ Increases size of matrix and adds grid around each original cell
```

```
% M - original matrix
% n - scaling factor

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% grid.m
% Dan Kunkle
% November 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ h w ] = size( M );
g = pixelRep( M, n, n );
gridRows = mod( 0:h*n, n );
gridRows = find( gridRows == 0 );
gridCols = mod( 0:w*n, n );
gridCols = find( gridCols == 0 );
g(gridRows,:) = 0.5;
g(:,gridCols) = 0.5;
===== End of file "grid.m"


===== File "hex2binV.m":
function bin = hex2binV( hex )

%@@ Convert hexidecimal strings to binary vectors

% Returns a matrix containing the binary equivalents of hexidecimal number(s)

% hex - a matrix, each row being a hex string

% bin - a vector of 0's and 1's representing a binary number. If there are
%       multiple hexidecimal numbers given each row of bin is another binary
%       number

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% hex2binV.m
% Dan Kunkle
% June 2003
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[numHex, hexLen] = size(hex);
binList = dec2binV(0:15);
for i = 1:numHex
    hexRule = hex(i,:);
    bin(i,:) = reshape(binList(hex2dec(hexRule(:))'+1,:)',1,hexLen*4);
end
===== End of file "hex2binV.m"


===== File "increaseM.m":
function newRule = increaseM( rule, m, newM )

%@@ Convert a rule of neigh. size m to one with a larger neigh.

% Returns fully specified binary CA rule(s) when given rule(s)

% rule - original rule
% m - neighborhood size of original rule
```

```
% newM - neightborhood size of new rule

%both m and newM should be odd

% newRule - a rule equivilent to the original rule specified for m=newM

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% increaseM.m
% Dan Kunkle
% June 2003
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[numRules, c] = size(rule);
mDiff = newM - m;
newNeigh = dec2binV(0:2^newM-1);
centerNeigh = newNeigh(:,1+(mDiff/2):newM-(mDiff/2));
centerNeighDec = bin2decV(centerNeigh);
for i = 1:numRules
    newRule(i,:) = rule(i,centerNeighDec+1);
end
===== End of file "increaseM.m"


===== File "initStructs.m":
function [] = initStructs( rule, m, minS, maxS, hideZeros, maxZeroString, ...
                           glideDist )

%@@ A simple search procedure for stable structures in CA

% tests given rule for initial structures arising from any initial
% configuration with n or fewer cells. Shows figures for visual examination.
% *Only tests all configurations for symmetric CA rules*

% rule - binary CA rule
% m - neighborhood size
% minS - minimum configuration size
% maxS - maximum configuration size
% hideZeros - hide evolutions that have all 0's on last step
% maxZeroString - maximum number of contiguous zeros config can have
% glideDist - distance to either side of center which must contain non-zero
%             cells to be valid (tend to find gliders when using larger
%             values of glideDist)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% initStructs.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

d = 50;  % distance on each side of configuration
t = 100; % time steps for each CA
zeroString = sprintf( '%u', zeros(1,maxZeroString+1) );

% construct inital state which includes all possible configurations with
% less than n cells (exclude reflections of configurations)

%figure;
%subplot( 1, 1, 1, 'replace' ); % so figure stays same size
```

```
initM = [ zeros(1,d) 1 zeros(1,d) ];
evo = ca1d( initM, m, rule, t );
if( sum(evo(t,:)) > 0 )
    imshow( ~evo, [0 1] );
    pause;
end
initM = [ zeros(1,d) 1 1 zeros(1,d) ];
evo = ca1d( initM, m, rule, t );
if( sum(evo(t,:)) > 0 )
    imshow( ~evo, [0 1] );
    pause;
end
for i = minS-2:maxS-2
    % find starting number based on maxZeroString
    if( maxZeroString+1 < i )
        initBin = repmat( [ zeros(1,maxZeroString) 1 ], 1, ...
                          ceil(i/maxZeroString+1) );
        initBin = initBin(1:i);
        start = bin2decV( initBin );
    else
        start = 0;
    end

    for j = start:(2^i)-1
        bin = dec2binV( j, i );
        % if bin contains more than maxZeroStrings in a row don't consider it
        if( isempty( findstr( zeroString, sprintf( '%u', bin ) ) ) )
            refDec = bin2decV( fliplr( bin ) );
            if( j <= refDec )
                initM = [ zeros(1,d) 1 bin 1 zeros(1,d) ];
                evo = ca1d( initM, m, rule, t );
                if( sum(evo(t,:)) > 0 )
                    [ rows iWidth ] = size( initM );
                    glideColL = ceil(iWidth/2) - glideDist;
                    glideColR = ceil(iWidth/2) + glideDist;
                    if ( sum(evo(:,glideColL) > 0) || ...
                         sum(evo(:,glideColR) > 0) )
                        figure( 'Position', [100 100 800 600] );
                        subplot( 1, 1, 1, 'replace' );
                        imshow( ~evo, [0 1] );
                        %beep;
                        %pause;
                    end
                end
            end
        end
    end
end

beep;
===== End of file "initStructs.m"


===== File "lambda.m":
function l = lambda( rule )

%@@ Calculates lambda (activity) parameter for a CA rule
```

```
% Returns Langton's lambda parameter for a CA rule (the ratio of neighborhoods
%         yeilding a 1 on next step to total number of neighborhoods)

% rule - binary CA rule

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% lambda.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ row col ] = size( rule );
l = 1 - abs( 2 * sum(rule)/col - 1 );
===== End of file "lambda.m"


===== File "lexrule2grayrule.m":
function gr = lexrule2grayrule( lex )

%@@ Convert lexicographic rule ordering to gray code ordering

% converts a CA rule in lexicographical ordering into a rule in
% gray code ordering

% lex - binary CA rule in lexicographical order

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% lexrule2grayrule.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ c ruleSize ] = size( lex );
neighBin = dec2binV( fliplr( 0:ruleSize-1 ) );
neighGray = bin2gray( neighBin );
indexGray = bin2decV( neighGray ) + 1;
gr = lex( indexGray );
===== End of file "lexrule2grayrule.m"


===== File "lexrule2sumrule.m":
function sr = lexrule2sumrule( lex )

%@@ Convert lexicographic rule ordering to sum ordering

% converts a CA rule in lexicographical ordering into a rule in
% sum ordering

% lex - binary CA rule in lexicographical order

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% lexrule2sumrule.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ c ruleSize ] = size( lex );
neighSum = sum( dec2binV( 0:ruleSize-1 )' );
```

```
[ sorted index ] = sort( neighSum );
sr = lex( index );
===== End of file "lexrule2sumrule.m"


===== File "lexrule2symrule.m":
function [ sym, neg, ref  ] = lexrule2symrule( lex, m )

%@@ Convert lexicographic rule ordering to symmetric neigh. ordering

% converts a CA rule in lexicographical ordering into a rule in
% symmetric neighborhood ordering, which has three parts
% (1) symmetric neighborhoods
% (2) non-symmetric neighborhoods ordered so a rule and its negated rule
%     yield reversed bit orders
% (3) non-symmetric neighborhoods ordered so a rule and its reflected rule
%     yield reversed bit orders

% lex - binary CA rule in lexicographical order
% m - size of neighborhood

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% lexrule2symrule.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if( m == 3 )
    % if elementary rule just use shortcut, otherwise use general case
    sym = lex( [ 1 3 6 8 ] );
    neg = lex( [ 2 4 5 7 ] );
    ref = lex( [ 2 4 7 5 ] );
else

[ c ruleSize ] = size( lex );
usedNeigh = zeros(ruleSize,2);
neighBin = dec2binV( fliplr( 0:ruleSize-1 ) );

sym = [];
neg = [];
ref = [];
symcount = 0;
negcount = 0;
refcount = 0;

numSym = pow2( ceil(m/2) );
numNonSym = ruleSize - numSym;

for i = 1:ruleSize
    neigh = neighBin(i,:);
    if ( ~( usedNeigh(i,1) || usedNeigh(i,2) ) )
    %neighborhood has not been seen before, mark it and store it
        if( sum( neigh == fliplr( neigh )) == m )
            %neighborhood is symmetric
            symcount = symcount + 1;
            sym(symcount) = lex(i);
            %mark neighborhood for
            usedNeigh(i,:) = [ 1 1 ];
```

```
        else
            %neighborhood is non-symmetric
            negcount = negcount + 1;
            neg(negcount) = lex(i);
            neg(numNonSym-negcount+1) = lex(ruleSize-bin2decV(~neigh));
            refcount = refcount + 1;
            ref(refcount) = lex(i);
            ref(numNonSym-refcount+1) = lex(ruleSize-bin2decV(fliplr(neigh)));
            %mark neighborhood, negated neighborhood, and reflected neighborhood
            usedNeigh(i,:) = [ 1 1 ];
            usedNeigh(ruleSize-bin2decV(~neigh),:) = [ 1 1 ];
            usedNeigh(ruleSize-bin2decV(fliplr(neigh)),:) = [ 1 1 ];
        end
    elseif( ~usedNeigh(i,1) )
        negcount = negcount + 1;
        neg(negcount) = lex(ruleSize-bin2decV(neigh));
        neg(numNonSym-negcount+1) = lex(ruleSize-bin2decV(~neigh));
        %mark neighborhood and negated neighborhood
        usedNeigh(i,1) = 1;
        usedNeigh(ruleSize-bin2decV(~neigh),1) = 1;
    elseif( ~usedNeigh(i,2) )
        refcount = refcount + 1;
        ref(refcount) = lex(ruleSize-bin2decV(neigh));
        ref(numNonSym-refcount+1) = lex(ruleSize-bin2decV(fliplr(neigh)));
        %mark neighborhood and reflected neighborhood
        usedNeigh(i,2) = 1;
        usedNeigh(ruleSize-bin2decV(~neigh),2) = 1;
    end
end

end %if
===== End of file "lexrule2symrule.m"


===== File "meanOverSets.m":
function means = meanOverSets(sets, paramList, stats)

%@@ Mean for set of params over subsets each is a member of

% sets - cell array of sets of parameters (vector of ints)
% paramList - list of parameters (ints) that appear in sets
% stats - matrix of statistical values

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% meanOverSets.m
% Dan Kunkle
% June 2003
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%length(sets) should be == to num rows in stats
[numSets, numStats] = size(stats);
numParams = length(paramList);
means = [paramList' zeros(numParams,numStats)];
counts = zeros(numParams,numStats);
for i = 1:numSets
    numParamsInSet = length(sets{i});
    means(sets{i},:) = means(sets{i},:) + ...
        repmat([0 stats(i,:)],numParamsInSet,1);
```

```
    counts(sets{i},:) = counts(sets{i},:) + ones(numParamsInSet,numStats);
end
means(:,2:numStats+1) = means(:,2:numStats+1) ./ counts;
===== End of file "meanOverSets.m"


===== File "meanfield.m":
function mf = meanfield( rule, m )

%@@ Calculates mean field parameters for a CA rule

% Returns normalized mean field parameters for a binary CA rule
% For a rule with a neighborhood size of size m there will be 2^(2^m) entires
% in the rule and m+1 mean field parameters, each will be in the range [0,1]

% rule - binary CA rule
% m - size of neighborhood

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% meanfield.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% size of rule
[ r c ] = size( rule );

% sums of bits in each neighborhood
% neighborhood with all 1's it leftmost, all 0's is right most
neighborhoodSums = fliplr( sum( dec2binV( 0:c-1 ), 2 )' );

% mean field parameters
for i = 1:m+1
    neighborhoods = find( neighborhoodSums == i-1 );
    [ r, num ] = size( neighborhoods );
    mf(i) = sum( rule( neighborhoods ) ) / num;
end
===== End of file "meanfield.m"


===== File "meanfieldInt.m":
function mfI = meanfieldInt( rule, m )

%@@ Calculates mean field parameters as integers

% Returns integer mean field parameters for a binary CA rule
% For a rule with a neighborhood size of size m there will be 2^(2^m) entires
% in the rule and m+1 mean field parameters, each will be in the range [0,1]

% rule - binary CA rule
% m - size of neighborhood

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% meanfieldInt.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% size of rule
[ r c ] = size( rule );

% sums of bits in each neighborhood
% neighborhood with all 1's it leftmost, all 0's is right most
neighborhoodSums = fliplr( sum( dec2binV( 0:c-1 ), 2 )' );

% mean field parameters
for i = 1:m+1
    neighborhoods = find( neighborhoodSums == i-1 );
    [ r, num ] = size( neighborhoods );
    mfI(i) = sum( rule( neighborhoods ) );
end
===== End of file "meanfieldInt.m"


===== File "mfChart.m":
function [] = mfChart( types )

%@@ Plots a number of views of mean field parameter space

% Makes a plot for the mean field parameters of the elementary rule space
% similar to that made by barChartParam for each of the other parameters

% types: a vector of bits specifying which types of graphs to generate
%   1 - Bar chart of all 64 mean field values, visited in lexicographic and
%       all binary-reflected Gray code orders
%   2 - Four bar charts, one for each mean field parameter
%   3 - Six sets of pie charts, one for each pair of mean field parameters

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% mfChart.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Mostly brute-forcing here with lots of loops
% Not the most elegent code ever, but shows pretty charts

%dynamic classes
[ c1, c2, c3, c4, c5, c6 ] = classes;

%mean field parameters
mfVals = [];
binRules = dec2binV( 0:255 );
for i = 1:256
    mfVals(i,:) = meanfieldInt( binRules(i,:), 3 );
end

if(types(1))
%organize data for bar chart
X = 1:64;
xLabels = '';
Y = zeros(64,6);
count = 0;
for i1 = 0:1
for i2 = 0:3
for i3 = 0:3
```

```
for i4 = 0:1
    count = count + 1;
    xLabels(count,:) = [ num2str(i1) num2str(i2) num2str(i3) num2str(i4) ];
    for j = 1:6
        matchMat = repmat( [ i1 i2 i3 i4 ], 256, 1 );
        correctVals = find( sum([mfVals==matchMat]') == 4 );
        [r num] = size( intersect(correctVals, eval(['c' num2str(j)])+1) );
        Y(count,j) = num;
    end
end
end
end
end

%show lexicographic ordering bar chart
figure;%('Position', [1 1 800 300]);
bar( X, Y, 'stacked' );
axis( [ 0 65 0 max(max(Y)) ] );
xlabel('Mean Field Index');
set( gca, 'XTick', X );
%set( gca, 'XTickLabel', '' )
ylabel('Number of Elementary CA');
legend('Null','Fixed Point','Two-Cycle','Periodic','Complex','Chaotic');
title('Mean Field Parameter In Lexicographic Order');

%re-order into Gray code and re-display
%try all 24 possible binary-reflected Gray codes
mfPerms = perms( 1:4 ); % all permutations of 4 mean field parameters
maxNums = mfPerms*10; %will be maximum numbers for each permutation
maxNums(find(maxNums==10)) = 2; %first mf param can be 0 or 1
maxNums(find(maxNums==20)) = 4; %second mf param can be 0,1,2, or 3
maxNums(find(maxNums==30)) = 4; %third mf param can be 0,1,2, or 3
maxNums(find(maxNums==40)) = 2; %fourth mf param can be 0 or 1

X2 = zeros(24,64);
Y2 = zeros(64,6,24);
for i = 1:24
    gc = grayGen( maxNums(i,:) );
    count = 0;
    for j = 1:64
        count = count + 1;
        index = gc(j,mfPerms(i,1)) + ...
                gc(j,mfPerms(i,2))*maxNums(i,mfPerms(i,1)) + ...
                gc(j,mfPerms(i,3))*prod(maxNums(i,mfPerms(i,1:2))) + ...
                gc(j,mfPerms(i,4))*prod(maxNums(i,mfPerms(i,1:3))) + 1;
        X2(i,j) = index;
        Y2(j,:,i) = Y(index,:);
    end
end

%TESTING
%X2
%unique(X2,'rows')
%size(unique(X2,'rows'))
%pause;

%show bar chart of each of 24 gray code order
```

```
figure;
for i = 1:24
    bar( X, Y2(:,:,i), 'stacked' );
    axis( [ 0 65 0 max(max(Y2(:,:,i))) ] );
    xlabel('Mean Field Index');
    set( gca, 'XTick', X );
    set( gca, 'XTickLabel', sprintf('%d|', X2(i,:)) )
    ylabel('Number of Elementary CA');
    legend('Null','Fixed Point','Two-Cycle','Periodic','Complex','Chaotic');
    mfOrdering =  [ 'n_' num2str(mfPerms(i,1)-1) ...
                    'n_' num2str(mfPerms(i,2)-1) ...
                    'n_' num2str(mfPerms(i,3)-1) ...
                    'n_' num2str(mfPerms(i,4)-1) ];
    title( [ 'Mean Field - Gray Code ' mfOrdering ] );

    pause %pause after each graph is displayed
end
end % TYPE 1

if(types(2))
    % X and Y bar chart values for each of 4 mean field parameter
    X0 = [1:2];
    Y0 = [];
    X1 = [1:4];
    Y1 = [];
    X2 = [1:4];
    Y2 = [];
    X3 = [1:2];
    Y3 = [];
    for j = 1:6 % for each class
        for i = 0:1 % for n0 and n3
            Y0(i+1,j) = sum( mfVals(eval(['c' num2str(j)])+1,1) == i );
            Y3(i+1,j) = sum( mfVals(eval(['c' num2str(j)])+1,4) == i );
        end
        for i = 0:3 % for n1 and n2
            Y1(i+1,j) = sum( mfVals(eval(['c' num2str(j)])+1,2) == i );
            Y2(i+1,j) = sum( mfVals(eval(['c' num2str(j)])+1,3) == i );
        end
    end

    baseFName = '../results/mfCharts/bar_';
    % show bar charts
    for i = 0:3
        figure;
        iStr = num2str(i);
        bar( eval(['X' iStr]), eval(['Y' iStr]), 'stacked' );
        xlabel('Parameter Value');
        set( gca, 'XTick', eval(['X' iStr]) );
        set( gca, 'XTickLabel', sprintf('%d|', eval(['X' iStr])-1) );
        ylabel('Number of Elementary CA');
        legend('Null','Fixed Point','Two-Cycle','Periodic','Complex','Chaotic');
        ptitle = [ 'Mean Field Parameter n_' iStr ];
        title(ptitle);
        %save plot
        fName = [ baseFName 'n' num2str(i) ];
        print( '-depsc', fName );
    end
```

```
end % TYPE 2

if(types(3))
    %show six sets of pie charts
    %nothing fancy, just done with loops "by hand"

    %each set of pie charts is represented by a 3-D matrix, the first two
    %dimensions are the two mean field parameters and the third is the set
    %of six Li-Packard classes.
    n0n1 = zeros(4,2,6);
    n0n3 = zeros(2,2,6);
    n1n2 = zeros(4,4,6);

    % n0n1
    for i = 0:3 %n1
        for j = 0:1 %n0
            for k = 1:6 %classes
                classStr = num2str(k);
                n0n1(i+1,j+1,k) = sum( ...
                    (mfVals(eval(['c' classStr])+1, 2) == i) & ...
                    (mfVals(eval(['c' classStr])+1, 1) == j) );
            end
        end
    end

    % n0n3
    for i = 0:1 %n3
        for j = 0:1 %n0
            for k = 1:6 %classes
                classStr = num2str(k);
                n0n3(i+1,j+1,k) = sum( ...
                    (mfVals(eval(['c' classStr])+1, 4) == i) & ...
                    (mfVals(eval(['c' classStr])+1, 1) == j) );
            end
        end
    end

    % n1n2
    for i = 0:3 %n1
        for j = 0:3 %n2
            for k = 1:6 %classes
                classStr = num2str(k);
                n1n2(i+1,j+1,k) = sum( ...
                    (mfVals(eval(['c' classStr])+1, 2) == i) & ...
                    (mfVals(eval(['c' classStr])+1, 3) == j) );
            end
        end
    end

    % show pie charts (save to files)
    warning off MATLAB:divideByZero;

    baseFName = '../results/mfCharts/pie_';

    figure;
    %n0n1
    for i = 0:3 %n1
```

```
        for j = 0:1 %n0
            fName = [ baseFName 'n0n1_' num2str(j) num2str(i) ];
            %subplot(4,2,i*2+j+1);
            pie(n0n1(i+1,j+1,:));
            textObjs = findobj(gca,'Type','text');
            set(textObjs,{'String'},{''});
            print( '-depsc', fName );
        end
    end

    figure;
    %n0n3
    for i = 0:1 %n3
        for j = 0:1 %n0
            fName = [ baseFName 'n0n3_' num2str(j) num2str(i) ];
            %subplot(2,2,i*2+j+1);
            pie(n0n3(i+1,j+1,:));
            textObjs = findobj(gca,'Type','text');
            set(textObjs,{'String'},{''});
            print( '-depsc', fName );
        end
    end

    figure;
    %n1n2
    for i = 0:3 %n1
        for j = 0:3 %n2
            fName = [ baseFName 'n1n2_' num2str(i) num2str(j) ];
            %subplot(4,4,i*4+j+1);
            pie(n1n2(i+1,j+1,:));
            textObjs = findobj(gca,'Type','text');
            set(textObjs,{'String'},{''});
            print( '-depsc', fName );
        end
    end

    warning on MATLAB:divideByZero;

end % TYPE 3
===== End of file "mfChart.m"


===== File "mu.m":
function result = mu( rule, m )

%@@ Calculates mu (sensitivity) parameter for a CA rule

% Returns Binder's sensitivity parameter mu for binary CA with neighborhood
% size m

% rule - binary CA rule
% m - neighborhood size

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% mu.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
[ row ruleSize ] = size( rule );
ruleBin = dec2binV( 0:ruleSize-1 );

% may be able to vectorize this further later, use a for loop for now
count = 0;
for i = 1:m
    flippedBin = ruleBin;
    flippedBin(:,i) = ~flippedBin(:,i);
    count = count + sum( abs( rule( bin2decV(flippedBin)+1 ) - rule ) );
end

result = count / (m*ruleSize);
===== End of file "mu.m"


===== File "nchoosekAll.m":
function result = nchoosekAll( nk )

%@@ A generalized version of MATLAB's nchoosek function

% nk - a matrix of size 2xM, values in row 1 are the n, values in row 2
%      are the k.

% returns - a vector of size 1xM where result(i) = nchoosek( nk(1,i), nk(2,i) )
%           if 0 <= k <= n and 0 otherwise

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% nchoosekAll.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ row col ] = size( nk );

for i = 1:col
    n = nk(1,i);
    k = nk(2,i);
    if( (k >= 0) && (k <= n) )
        result(i) = nchoosek( n, k );
    else
        result(i) = 0;
    end
end
===== End of file "nchoosekAll.m"


===== File "negate.m":
function r = negate( rule )

%@@ Negate a CA rule, yields same behavior with white/black reversal

% Returns a rule that produces the same behavior as the given rule except that
% the black and white pixels are swapped

% rule - binary CA rule (or a matrix where each row is a CA rule).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% reflect.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

r = ~fliplr(rule);
===== End of file "negate.m"


===== File "neighDom.m":
function nd = neighDom( rule, m )

%@@ Calculates neighborhood dominance parameter for a CA rule

% Returns neighborhood dominance parameter defined by Oliveira et al.
% similar to absolute activity, doesn't dicriminate between center cell
% and perimeter cells of incresing distance.

% rule - binary CA rule

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% neighDom.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% vectorized - no loops

[ row ruleSize ] = size( rule );

ruleBin = dec2binV( fliplr( 0:ruleSize-1 ) )';
binSums = sum( ruleBin );
c = ceil(m/2); %center of neighborhood

% calculate non-normalized neighborhood dominance
coeffs1 = nchoosekAll( [ repmat( m, 1, ruleSize ); binSums + c ] );
coeffs2 = nchoosekAll( [ repmat( m, 1, ruleSize ); binSums - c ] );
zeroDom = zeros(1,ruleSize);
zeroDom( find( binSums < c & rule == 0 ) ) = 1;
oneDom = zeros(1,ruleSize);
oneDom( find( binSums >= c & rule == 1 ) ) = 1;

nd = sum( (coeffs1 .* zeroDom) + (coeffs2 .* oneDom) );

% normalize
chooseLow = nchoosekAll( [ repmat(m,1,c); 0:c-1 ] );
chooseHigh = nchoosekAll( [ repmat(m,1,c); [0:c-1] + c ] );
nd = nd / ( 2 * sum( chooseLow .* chooseHigh ) );
===== End of file "neighDom.m"


===== File "numIntersects.m":
function n = numIntersects( sets )

%@@ Number of intersecting elements in all pairs of sets

% Returns the number of intersecting elements between each pair of sets
% from the array of sets provided. Includes duplicate members in each set
```

```
% (not technically a _set_, but close).

% sets - a cell array of sets (matricies) where each row is an element

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% numIntersects.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ r numSets ] = size( sets );
pairs = nchoosek( 1:numSets, 2 );
[ numPairs c ] = size( pairs );
n = 0;
for i = 1:numPairs
    % find the set intersection
    s1 = sets{pairs(i,1)};
    s2 = sets{pairs(i,2)};
    inter = intersect( s1, s2, 'rows');
    % if there are duplicate members in either set count them too
    [ numInters c ] = size(inter);
    for j = 1:numInters
        n = n + ( sum(ismember(s1,inter(j,:),'rows')) * ...
                  sum(ismember(s2,inter(j,:),'rows')) );
    end
end
===== End of file "numIntersects.m"


===== File "numTotRules.m":
function [numGroupsByClass, numRulesByClass] = numTotRules()

%@@ Return the number of totalistic rule groups and rules in each class

% numGroupsByClass - number of behav. equiv. rule groups in each of 6 classes
% numRulesByClass - number of rules in each of 6 classes

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% numTotRules.m
% Dan Kunkle
% June 2003
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

equivGroups = equivTot(0);
numGroups = length(equivGroups);
classNums = classesCombinedTot;
[n, fp, tc, p, co, ch] = classesTot;
numRulesByClass = ...
    [length(n) length(fp) length(tc) length(p) length(co) length(ch)];
numGroupsByClass = zeros(1,6);
for i = 1:numGroups
    groupClassNum = classNums(equivGroups{i}(1)+1);
    numGroupsByClass(groupClassNum) = numGroupsByClass(groupClassNum) + 1;
end
===== End of file "numTotRules.m"


===== File "paramSubsetStats.m":
```

```
function [ cM, overlapM, intraM, interM, ratioM, nnPerfM ] = ...
    paramSubsetStats( display, fileName )

%@@ Calculate statistics for all subsets of parameters

% Returns a cell array listing all subsets of CA parameters along with the
% corresponding number of overlapping elements between all pairs of Li-Packard
% classes and clustering stats (intra and inter cluster distances and the ratio
% or intra/inter). Each row is a subset of parameters (2^n rows where n is the
% number of parameters).

% display - flag: 1 = display results on command line if fileName = ''
% fileName - if specified and display is set will output to file (LaTeX style)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% paramSubsetStats.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

paramVals = readElementaryParams; % all 11 params (last 4 are meanfield)
paramVals = paramVals(:,1:7); % don't use meanfield params
paramSubsets = powerSet( 1:7 ); % cell array of all subsets (less null set)
[ r numSubsets ] = size(paramSubsets);
[ null, fixedPoint, twoCycle, periodic, complex, chaotic ] = classes;
cM = paramSubsets';
overlapM = zeros(1,numSubsets);
intraM = zeros(1,numSubsets);
interM = zeros(1,numSubsets);
ratioM = zeros(1,numSubsets);
nnPerfM = zeros(1,numSubsets);

for i = 1:numSubsets
    paramSub = paramVals( :, cell2mat(cM(i,1)) );
    %overlap stats
    sets = { paramSub(null+1,:), ...
             paramSub(fixedPoint+1,:), ...
             paramSub(twoCycle+1,:), ...
             paramSub(periodic+1,:), ...
             paramSub(complex+1,:), ...
             paramSub(chaotic+1,:) };
    overlapM(i) = numIntersects( sets );
    %clusters stats
    % unfortunately, sets need to be transposed for clusterDist
    setsT = { cell2mat(sets(1))', cell2mat(sets(2))', ...
              cell2mat(sets(3))', cell2mat(sets(4))', ...
              cell2mat(sets(5))', cell2mat(sets(6))' };
    [ intra, inter ] = clusterDist(setsT); % intra and inter
    ratio = inter/intra;
    intraM(i) = intra;
    interM(i) = inter;
    ratioM(i) = inter/intra;
    %neural net training performance stats
    numRuns = 5; %average value of multiple runs to acount for stocasticity
    totalError = 0;
    for j = 1:numRuns
        [net, perf] = trainNetSubset(cM{i,1}, 'elem');
```

```
        totalError = totalError + perf;
    end
    nnPerfM(i) = totalError/numRuns;
end

if(display)
    if( strcmp( fileName, '' ) )
        for i = 1:numSubsets
            disp( [ mat2str(cell2mat(cM(i,1))) ' ' num2str(overlapM(i)) ...
                    ' ' num2str(intraM(i))  ' ' num2str(interM(i)) ...
                    ' ' num2str(ratioM(i)) ] );
        end
    else
        paramLabels = ...
            {'$\\lambda$', '$Z$', '$\\mu$', 'AA', 'ND', 'AP', '$\\upsilon$'}
        fid = fopen( fileName, 'w' );
        for i = 1:numSubsets
            % convert subset into string
            prefix = '';
            paramList = cell2mat(cM(i));
            [ rows numParams ] = size( paramList );
            for j = 1:numParams
                prefix = [ prefix paramLabels{paramList(j)} ' ' ];
            end
            vals = [ overlapM(i) intraM(i) interM(i) ratioM(i) nnPerfM(i) ];
            fprintf( fid, ...
            [prefix '& %u & %4.3f & %4.3f & %4.3f & %1.4f \\\\ \\hline \n'], ...
            vals );
        end
        fclose( fid );
    end
end
===== End of file "paramSubsetStats.m"


===== File "paramTestEquiv.m":
function nonEquiv = paramTestEquiv( params )

%@@ Finds equivalent rules that have different parameter values

% params - a list of 256 parameter calculations, one for each elementary CA,
%          ordered 00000000, 00000001, ..., 11111111

% Checks if all rules equivalent by negation and reflection have the same
% parameter values, as specified by the input params. Will return a list of
% equivalent rules that have different parameter values in the form:
%   rule paramValue1 equivRule paramValue2
% where rule is a CA rule, paramValue1 is the parameter value of rule, equivRule
% is a rule equivalent to rule, and paramValue2 is the parameter value of
% equivRule. Rules are specified by their index number.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% paramTestEquiv.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if( size(params) ~= 256 )
```

```
    disp('Must include 256 parameter values, one for each elementary CA');
    return;
end

nonEquiv = [];
numNonEquiv = 0;

rules = dec2binV( 0:255 );
neg = negate(rules);
ref = reflect(rules);
negRef = negate(ref);
negI = bin2decV(neg)+1;
refI = bin2decV(ref)+1;
negRefI = bin2decV(negRef)+1;

for i = 1:256
    if( params(i) ~= params(negI(i)) )
        numNonEquiv = numNonEquiv + 1;
        nonEquiv(numNonEquiv,:) = [ i params(i) negI(i) params(negI(i)) 1 ];
    end
    if( params(i) ~= params(refI(i)) )
        numNonEquiv = numNonEquiv + 1;
        nonEquiv(numNonEquiv,:) = [ i params(i) refI(i) params(refI(i)) 2 ];
    end
    if( params(i) ~= params(negRefI(i)) )
        numNonEquiv = numNonEquiv + 1;
        nonEquiv(numNonEquiv,:)=[ i params(i) negRefI(i) params(negRefI(i)) 3 ];
    end
end
===== End of file "paramTestEquiv.m"


===== File "patDiffFigs.m":
function [] = patDiffFigs( which )

%@@ Shows difference patterns for a rule from each Li-Packard class

% Saves pattern difference images for a representitive CA from each of the
% six Li-Packard classes

% which - which classes to generate images for. binary vector of length 6

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% .m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n = 400; % number of cells (width of CA)
t = 200; % number of time steps
initM = rand(1,n) > .5; % inital state
m = 3; % neighborhood size

rulesDec = [ 40 13 15 26 110 18 ]; % rules to use
rulesBin = dec2binV( rulesDec, 8 ); % binary

fnPre = '../results/patDiff/rule';
fnPost = [ '_' num2str(n) 'x' num2str(t) '.eps' ];
```

```
for i = 1:6
    if(which(i))
        figure;
        imshow( 1-diffPatternOverlay( initM, m, rulesBin(i,:), t, 1 ) );
        fn = [fnPre num2str(rulesDec(i)) fnPost];
        print( '-deps', fn );
    end
end
===== End of file "patDiffFigs.m"


===== File "pixelRep.m":
function pixRepM = pixelRep( M, n, m )

%@@ Increases size of matrix by pixel replication

% Creates a new matrix of size nH x mW, where H and W are the height and width
% of the given matrix. The new matrix is created by replicating each pixel
% element in the original image n*m times. (this can be thought of as a very
% simplistic zoom function in terms of images).

% M - original matrix
% n - vertical scaling factor
% m - horizontal scaling factor

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% pixelRep.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ height, width ] = size(M);
xi = ones(n, 1) * [1:height];
xi = reshape( xi, 1, height*n );
yi = ones(m, 1) * [1:width];
yi = reshape( yi, 1, width*m );
pixRepM = M( xi, yi );
===== End of file "pixelRep.m"


===== File "plotClusterStats.m":
function [] = plotClusterStats( c, overlap, intra, inter, ratio, fileName )

%@@ Plot cluster statistics for parameter subsets

% c - cell array of all parameter subsets
% overlap - number of overlaps for each parameter subset
% intra - mean intracluster distance for each param subset
% inter - mean intercluster distance for each param subset
% ratio - ratio of inter/intra cluster distances
% fileName - optional file name to save figures to ('' will not save)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plotClusterStats.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%[ c overlap intra inter ratio ] = paramSubsetStats(0, '');

% split stats into sets corresponding to the number of parameters (1 to 7)
numStats = zeros(1,7); % the number of subsets with 1to7 elements
statTotal = zeros(4,7); % sum of all 4 stats for 7 set sizes
statMean = zeros(4,7); % mean of all 4 stats for 7 set sizes
statMin = zeros(4,7) + inf; % max values of all 4 stats for 7 set sizes
statMax = zeros(4,7); % min values of all 4 stats for 7 set sizes

allStats = [ overlap; intra; inter; ratio ];
[ numSets cols ] = size( c );
for i = 1:numSets
    [ rows setSize ] = size( cell2mat(c(i)) );
    numStats(setSize) = numStats(setSize) + 1;
    statTotal(:,setSize) = statTotal(:,setSize) + allStats(:,i);
    statMin(:,setSize) = min( statMin(:,setSize), allStats(:,i) );
    statMax(:,setSize) = max( statMax(:,setSize), allStats(:,i) );
end
statMean = statTotal ./ repmat( numStats, 4, 1 );

% intra and inter clustering statistics
figure;
hold on;
plot( 1:7, statMean(2,:), '-ob', 'LineWidth',2 ); % intracluster distance mean
plot( 1:7, statMax(2,:), '--b' ); % intracluster distance max
plot( 1:7, statMin(2,:), '-.b' ); % intracluster distance min
plot( 1:7, statMean(3,:), '-xg', 'LineWidth',2 ); % intercluster distance mean
plot( 1:7, statMax(3,:), '--g' ); % intercluster distance max
plot( 1:7, statMin(3,:), '-.g' ); % intercluster distance min
legend( 'Mean Intra Dist', 'Max Intra Dist', 'Min Intra Dist', ...
        'Mean Inter Dist', 'Max Inter Dist', 'Min Inter Dist', 0 );
title( 'Clustering statistic for all parameter subsets' );
ylabel( 'Mean Euclidean Distance' );
xlabel( 'Parameter Subset Size' );
hold off;
if( ~strcmp( fileName, '' ) )
    print( '-depsc', [ fileName 'IntraInter.eps' ] );
end

% clustering ratio
figure;
hold on;
plot( 1:7, statMean(4,:), '-ob', 'LineWidth', 2 ); % ratio mean
plot( 1:7, statMax(4,:), '--b' ); % ratio max
plot( 1:7, statMin(4,:), '-.b' ); % ratio min
legend( 'Mean Cluster Ratio', 'Max Cluster Ratio', 'Min Cluster Ratio' );
title( 'Clustering statistic for all parameter subsets' );
ylabel( 'Inter Dist / Intra Dist' );
xlabel( 'Parameter Subset Size' );
hold off;
if( ~strcmp( fileName, '' ) )
    print( '-depsc', [ fileName 'Ratio.eps' ] );
end

% overlap
figure;
```

```
hold on;
plot( 1:7, statMean(1,:), '-ob', 'LineWidth', 2 ); % overlap mean
plot( 1:7, statMax(1,:), '--b' ); % overlap max
plot( 1:7, statMin(1,:), '-.b' ); % overlap min
legend( 'Mean Overlap', 'Max Overlap', 'Min Overlap' );
title( 'Overlap statistic for all parameter subsets' );
ylabel( 'Number of Class Overlaps' );
xlabel( 'Parameter Subset Size' );
set(gca, 'YScale', 'log' );
hold off;
if( ~strcmp( fileName, '' ) )
    print( '-depsc', [ fileName 'Overlap.eps' ] );
end
===== End of file "plotClusterStats.m"


===== File "powerSet.m":
function c = powerSet( elements )

%@@ Returns cell array of all possible subsets

% Returns the power set (all possible subsets) of the set of elements (vector)
% given. Does not include the empty set. Sets with more elements will appear
% first.

% elements - full set of elements

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% powerSet.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

c = {}; %cell array of subsets
ci = 1;
[ row numElements ] = size(elements);
for i = 1:numElements
    k = numElements - i + 1;
    subs = nchoosek( elements, k );
    % place each subset into cell array
    [ numSubs col ] = size(subs);
    for j = 1:numSubs
        c(ci) = { subs(j,:) };
        ci = ci + 1;
    end
end
===== End of file "powerSet.m"


===== File "randint.m":
function r = randint( mini, maxi, n )

%@@ returns a vector of random integers

% mini - minimum integer
% maxi - maximum integer
% n - number of integers
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% randint.m
% Dan Kunkle
% November 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

r = floor( mini + ( maxi - mini + 1 ) * rand(1,n) );
===== End of file "randint.m"


===== File "readElementaryParams.m":
function vals = readElementaryParams( )

%@@ Read parameter values from file

% Reads a matrix of all of the parameters for each elementary CA to a file
% each collumn of the data represents a parameter, each row represents one
% of the 256 elementary CA (in lexicograpical order)

% fileName - file name to read data from

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% readElementaryParams.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fileName = 'data/paramValues';

% read the data
fid = fopen( [ fileName '.data' ], 'r' );
vals = fscanf( fid, '%g', [ 11, 256 ] )';
fclose(fid);
===== End of file "readElementaryParams.m"


===== File "readTotParams.m":
function vals = readTotParams( )

%@@ Read totalistic CA parameters from file

% Reads a matrix of all of the parameters for each totalistic k=2 r=3 CA from a
% file. each collumn of the data represents a parameter, each row represents one
% of the 256 elementary CA (in lexicograpical order)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% readTotParams.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fileName = 'data/totParamValues';

% read the data
fid = fopen( [ fileName '.data' ], 'r' );
vals = fscanf( fid, '%g', [ 256, 7 ] );
fclose(fid);
===== End of file "readTotParams.m"
```

```
===== File "reflect.m":
function r = reflect( rule )

%@@ Reflect a CA rule, yields same behavior with mirror symmetry

% Returns a rule that produces the same behavior as the given rule except that
% the states of the CA will be reflected left-to-right

% rule - binary CA rule (or a matrix where each row is a CA rule)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% reflect.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ row ruleSize ] = size( rule );
i = fliplr( [ 0:ruleSize-1 ] );
binI = dec2binV( i );
binIrefl = fliplr( binI );
iRefl = bin2decV( binIrefl );
r( :, i+1 ) = rule( :, iRefl+1 );
===== End of file "reflect.m"


===== File "ruleMontage.m":
function [] = ruleMontage( rules, m, type, rows, cols, height, width, ...
                           H, W, fileName )

%@@ Shows figure of small periods of CA evolution using subplots.

% rules - the CA rules to show
% m - size of neighborhood
% type - the type of rules
%   1: elementary
%   2: totalistic
% rows - number of subplot rows
% cols - number of suplot columns
% height - height of each individual CA image
% width - width of each individual CA iamge
% H - height of figures (in inches)
% W - width of figures (in inches)
% fileName - the base file name to store EPS images to (no save if '')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ruleMontage.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (type==1)
    rulesBin = rules;
    label = 'elem';
elseif (type==2)
    rulesBin = tot2full(rules);
    label = 'tot';
end
```

```
[numRules ruleSize] = size(rules);
rulesDec = bin2decV( rules );
rulesBinStr = dec2bin( rulesDec );

initM = rand( 1, width ) > .5; %initial CA matrix, half black

iptsetpref( 'ImshowBorder', 'tight' );
%set( 0, 'Units', 'inches' );
figure( 'PaperPositionMode', 'auto', 'Units', 'inches', 'Position', [0 0 W H] );
plotsPerFig = rows*cols;
plotSpot = 1;
figNum = 1;
for i = 1:numRules
    plotName = [ label ' ' num2str(rulesDec(i)) ' (' rulesBinStr(i,:) ')' ];
    h = subplot( rows, cols, plotSpot );
    set( h, 'FontSize', 8 );
    imshow( ~ca1d( initM, m, rulesBin(i,:), height ), [0 1] );
    title(plotName);
    plotSpot = plotSpot + 1;
    if ( plotSpot > plotsPerFig )
        if( ~strcmp(fileName, '') )
            fname = [ fileName num2str(figNum) '.eps' ];
            print( '-depsc', fname );
        end
        plotSpot = 1;
        if( i ~= numRules )
            figure( 'PaperPositionMode', 'auto', 'Units', 'inches', ...
                    'Position', [0 0 W H] );
            figNum = figNum + 1;
        end
    end
end

%set( 0, 'Units', 'pixels' );
===== End of file "ruleMontage.m"


===== File "slideshow.m":
function [] = slideshow( rules, m, labels )

%@@ Shows a sequence of CA evolutions, for manual classification

% Shows a number of evolutions for each rule given, pausing between each rule.
% Used mainly to manually classify a set of rules.

% rules - rules to show
% m - neighborhood size for each rule
% labels - for each rule (cell array)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% slideshow.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

height = 200; % number of steps for each CA run
width = 150;  % width of each CA
rows = 2;     % number of rows of CA
```

```
cols = 5;     % number of columns of CA (total CA per rule = rows*cols)
[ numRules ruleSize ] = size(rules);

for i = 1:numRules
    for j = 1:(rows*cols)
        subplot( rows, cols, j, 'replace' );
        imshow( ~ca1d( rand(1,width) > .5, m, rules(i,:), height ) );
    end
    if( size(labels) > 0 )
        title( cell2mat( labels(i) ) );
    end
    pause;
end
===== End of file "slideshow.m"


===== File "structs.m":
function [] = structs( fileName )

%@@ Examples of persistent structures in complex CA

% an ad hoc "copy/paste" collection of interesting patterns in various CA

% fileName - if ~= '' will save figures to file

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% structs.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

d = 20; % space around each pattern
t = 100; % time steps to run for

%TOTALISTIC K=2 R=3

% Rule 88:
ruleDec = 88;
rule = tot2full( dec2binV( ruleDec, 8 ) );

% NON-MOVING
    s1 = { ...
        [ 1 1 1 ], ...
        [ 1 1 0 0 1 0 1 1 1 1 1 ], ...
        [ 1 1 0 0 1 1 0 1 1 1 1 1 ], ...
        [ 1 1 0 1 1 1 1 1 1 0 1 1 ] ...
    };

% GLIDERS
    s2 = {...
        [ 1 1 0 0 1 1 0 1 1 1 1 1 1 ], ...
        [ 1 1 0 0 1 1 1 1 ] ...
    };

all = [s1 s2];
init = zeros(1,d);
[ rows numConfigs ] = size(all);
for i = 1:numConfigs
```

```
    init = [ init cell2mat(all(i)) zeros(1,d) ];
end
[ rows width ] = size(init);
init = [ init zeros(1,500) ]; % pad to the right to avoid wrap around

% show
figure;
evo = ca1d(init, 7, rule, t );
imshow( ~evo(1:t,1:width) );
if( ~strcmp( fileName, '' ) )
    print( '-deps', [ fileName 'tot' num2str(ruleDec) '.eps' ] );
else
    title( [ 'Totalistic k=2 r=3 code ' num2str(ruleDec) ] );
end

% show interactions
figure;
[ rows numStable ] = size(s1);
[ rows numGlider ] = size(s2);
for i = 1:numGlider
    for j = 1:numStable
        subplot( 3, 3, (i-1)*numStable + j );
        init = [ zeros(1,d) cell2mat(s2(i)) zeros(1,d) cell2mat(s1(j)) ...
                 zeros(1,d) ];
        [ rows width ] = size(init);
        init = [ init zeros(1,500) ]; % pad to avoid wrap around interference
        evo = ca1d( init, 7, rule, t );
        imshow( ~evo(1:t,1:width) );
    end
end
% glider2glider interaction
init = [ zeros(1,d) cell2mat(s2(1)) zeros(1,d*2) fliplr(cell2mat(s2(2))) ...
         zeros(1,d) ];
[ rows width ] = size(init);
init = [ init zeros(1,500) ]; % pad to avoid wrap around interference
evo = ca1d( init, 7, rule, t );
subplot( 3, 3, 9 );
imshow( ~evo(1:t,1:width) );
if( ~strcmp( fileName, '' ) )
    print( '-deps', [ fileName 'tot' num2str(ruleDec) 'inter.eps' ] );
end

% Rule 100:
ruleDec = 100;
rule = tot2full( dec2binV( ruleDec, 8 ) );

% NON-MOVING
    s1 = { ...
        [ 1 0 0 1 1 1 1 1 1 0 0 1 ], ...
        [ 1 0 0 1 0 0 0 0 1 0 0 1 ], ...
        [ 1 0 0 0 1 0 1 1 0 1 0 0 0 1 ] ...
    };

% GLIDERS
    s2 = {...
        [ 1 0 0 1 1 1 0 1 1 0 0 1 1 ] ...
    };
```

```
all = [s1 s2];
init = zeros(1,d);
[ rows numConfigs ] = size(all);
for i = 1:numConfigs
    init = [ init cell2mat(all(i)) zeros(1,d) ];
end
[ rows width ] = size(init);
init = [ init zeros(1,500) ]; % pad to the right to avoid wrap around

% show
figure;
evo = ca1d(init, 7, rule, t );
imshow( ~evo(1:t,1:width) );
if( ~strcmp( fileName, '' ) )
    print( '-deps', [ fileName 'tot' num2str(ruleDec) '.eps' ] );
else
    title( [ 'Totalistic k=2 r=3 code ' num2str(ruleDec) ] );
end

% show interactions
figure;
[ rows numStable ] = size(s1);
[ rows numGlider ] = size(s2);
for i = 1:numGlider
    for j = 1:numStable
        subplot( numGlider, numStable, (i-1)*numStable + j );
        init = [ zeros(1,d) cell2mat(s2(i)) zeros(1,d) cell2mat(s1(j)) ...
                 zeros(1,d) ];
        [ rows width ] = size(init);
        init = [ init zeros(1,500) ]; % pad to avoid wrap around interference
        evo = ca1d( init, 7, rule, t );
        imshow( ~evo(1:t,1:width) );
    end
end
if( ~strcmp( fileName, '' ) )
    print( '-deps', [ fileName 'tot' num2str(ruleDec) 'inter.eps' ] );
end

% Rule 164:
ruleDec = 164;
rule = tot2full( dec2binV( ruleDec, 8 ) );

% NON-MOVING
    s1 = { ...
        [ 1 0 0 1 1 1 1 1 0 0 1 ], ...
        [ 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 ] ...
    };

% GLIDERS
    s2 = {...
        [ 1 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 1 ] ...
    };

all = [s1 s2];
init = zeros(1,d);
[ rows numConfigs ] = size(all);
```

```
for i = 1:numConfigs
    init = [ init cell2mat(all(i)) zeros(1,d) ];
end
[ rows width ] = size(init);
init = [ init zeros(1,500) ]; % pad to the right to avoid wrap around

% show
figure;
evo = ca1d(init, 7, rule, t );
imshow( ~evo(1:t,1:width) );
if( ~strcmp( fileName, '' ) )
    print( '-deps', [ fileName 'tot' num2str(ruleDec) '.eps' ] );
else
    title( [ 'Totalistic k=2 r=3 code ' num2str(ruleDec) ] );
end

% show interactions
figure;
[ rows numStable ] = size(s1);
[ rows numGlider ] = size(s2);
for i = 1:numGlider
    for j = 1:numStable
        subplot( 1, 3, (i-1)*numStable + j );
        init = [ zeros(1,d) cell2mat(s2(i)) zeros(1,d) cell2mat(s1(j)) ...
                 zeros(1,d) ];
        [ rows width ] = size(init);
        init = [ init zeros(1,500) ]; % pad to avoid wrap around interference
        evo = ca1d( init, 7, rule, t );
        imshow( ~evo(1:t,1:width) );
    end
end
% show interations at other spacings
subplot(1,3,3);
init = [ zeros(1,d) cell2mat(s2(1)) zeros(1,d+3) cell2mat(s1(2)) ...
    zeros(1,d) ];
[ rows width ] = size(init);
init = [ init zeros(1,500) ]; % pad to avoid wrap around interference
evo = ca1d( init, 7, rule, t );
imshow( ~evo(1:t,1:width) );
%figure;
%for i = 1:10
%    subplot( 2, 5, i );
%    init = [ zeros(1,d) cell2mat(s2(1)) zeros(1,d+i) cell2mat(s1(2)) ...
%        zeros(1,d) ];
%    [ rows width ] = size(init);
%    init = [ init zeros(1,500) ]; % pad to avoid wrap around interference
%    evo = ca1d( init, 7, rule, t );
%    imshow( ~evo(1:t,1:width) );
%end
if( ~strcmp( fileName, '' ) )
    print( '-deps', [ fileName 'tot' num2str(ruleDec) 'inter.eps' ] );
end

% Rule 216:
ruleDec = 216;
rule = tot2full( dec2binV( ruleDec, 8 ) );
```

```
% NON-MOVING
    s1 = { ...
        [ 1 1 1 ], ...
        [ 1 1 0 0 1 0 1 1 1 1 1 ] ...
        [ 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 ] ...
    };

% GLIDERS
    s2 = {...
        [ 1 1 1 1 1 1 1 0 1 1 1 ] ...
        [ 1 1 0 0 1 1 1 1 ], ...
    };

all = [s1 s2];
init = zeros(1,d);
[ rows numConfigs ] = size(all);
for i = 1:numConfigs
    init = [ init cell2mat(all(i)) zeros(1,d) ];
end
[ rows width ] = size(init);
init = [ init zeros(1,500) ]; % pad to the right to avoid wrap around

% show
figure;
evo = ca1d(init, 7, rule, t );
imshow( ~evo(1:t,1:width) );
if( ~strcmp( fileName, '' ) )
    print( '-deps', [ fileName 'tot' num2str(ruleDec) '.eps' ] );
else
    title( [ 'Totalistic k=2 r=3 code ' num2str(ruleDec) ] );
end

% show interactions
figure;
[ rows numStable ] = size(s1);
[ rows numGlider ] = size(s2);
for i = 1:numGlider
    for j = 1:numStable
        subplot( numGlider, numStable, (i-1)*numStable + j );
        init = [ zeros(1,d) cell2mat(s2(i)) zeros(1,d) cell2mat(s1(j)) ...
                 zeros(1,d) ];
        [ rows width ] = size(init);
        init = [ init zeros(1,500) ]; % pad to avoid wrap around interference
        evo = ca1d( init, 7, rule, t );
        imshow( ~evo(1:t,1:width) );
    end
end
if( ~strcmp( fileName, '' ) )
    print( '-deps', [ fileName 'tot' num2str(ruleDec) 'inter.eps' ] );
end
===== End of file "structs.m"


===== File "testClusterDist.m":
%@@ Script to test the working of clusterDist.m

if 0
% two classes ("+" shaped)
```

```
% intra = 1, inter = 3
class1 = [ 2 1 3 2;
           3 2 2 1 ];
class2 = [ class1(1,:)+3;
           class1(2,:) ];
plot( class1(1,:), class1(2,:), '+r' );
hold on;
plot( class2(1,:), class2(2,:), 'og' );
axis( [ 0 7 0 4 ] );
[ intra inter ] = clusterDist( { class1, class2 } )
end

if 1
% two classes (50 random points and 100 random points in 3D)
class1x = rand( 1, 50 );
class1y = rand( 1, 50 );
class1z = rand( 1, 50 );
class2x = rand( 1, 100 ) + 2;
class2y = rand( 1, 100 ) + 2;
class2z = rand( 1, 100 ) + 2;
class1 = [ class1x; class1y; class1z ];
class2 = [ class2x; class2y; class2z ];

plot3( class1x, class1y, class1z, '+r' );
hold on;
plot3( class2x, class2y, class2z, 'og' );

[ intra inter ] = clusterDist( { class1, class2 } )
end
===== End of file "testClusterDist.m"


===== File "testElementary.m":
function ev = testElementary( which )

%@@ Tests parameters with all of the elementary rules

% which - a vector from 1 to n, where n is the number of parameters that
%         specifies which parameters to caculate
%         example: [ 0 1 0 0 0 1 0 ] would calculate the 2nd and 6th parameters
% 1 - lambda: activity
% 2 - Z
% 3 - mu: sensitivity
% 4 - absolute activity
% 5 - neighborhood dominance
% 6 - activity propigation
% 7 - upsilon
% 8 - mean field (there are 4 mean field parameters, appear int the last 4 col)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% testElementary.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rules = dec2binV( 0:255 );
ev = [];
```

```
j = 1;
c = 1;
if (which(j))
for i = 1:256
    ev(c,i) = lambda( rules(i,:) );
end
c = c + 1;
end

j = j + 1;
if (which(j))
for i = 1:256
    ev(c,i) = z( rules(i,:), 3 );
end
c = c + 1;
end

j = j + 1;
if (which(j))
for i = 1:256
    ev(c,i) = mu( rules(i,:), 3 );
end
c = c + 1;
end

j = j + 1;
if (which(j))
for i = 1:256
    ev(c,i) = absActivity( rules(i,:), 3 );
end
c = c + 1;
end

j = j + 1;
if (which(j))
for i = 1:256
    ev(c,i) = neighDom( rules(i,:), 3 );
end
c = c + 1;
end

j = j + 1;
if (which(j))
for i = 1:256
    ev(c,i) = actProp( rules(i,:), 3 );
end
c = c + 1;
end

j = j + 1;
if (which(j))
for i = 1:256
    ev(c,i) = upsilon( rules(i,:), 3 );
end
c = c + 1;
end
```

```
j = j + 1;
if (which(j))
for i = 1:256
    ev(c:c+3,i) = meanfield( rules(i,:), 3 )';
end
c = c + 4;
end
===== End of file "testElementary.m"


===== File "testUpsilon.m":
function [ n1, n2, n3, n4, n5, n6 ] = testUpsilon( )

%@@ Test variants of upsilon parameter

% test for violation of the "equivalent rules have the same parameter value"
% requierment

% n1 through n6 are the non-equivalencies for each of the six variants

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% testUpsilon.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rulesBin = dec2binV( 0:255 );
params = zeros(1,256);
for i = 1:6
    for j = 1:256
        params(j) = upsilonType( rulesBin(j,:), 3, i );
    end
    eval( [ 'n' num2str(i) ' = paramTestEquiv(params);' ] );
end
===== End of file "testUpsilon.m"


===== File "tot2full.m":
function full = tot2full( tot )

%@@ Convert a totalistic rule to a fully specified rule with k = 2

% Returns fully specified binary CA rule(s) when given rule(s) in totalistic
% form. A totalistic rule is of length m+1 where m is the neighborhood size. The
% full rule is of size 2^m.

% tot - totalistic rule(s), each row is a rule

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% tot2full.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ rows, totLen ] = size( tot );
fullLen = 2^(totLen-1);
fullNeighs = dec2binV( fliplr( 0:fullLen-1 ) )';
```

```
neighSums = sum( fullNeighs );
full = zeros( rows, fullLen );
for i = 1:rows
    for j = 1:totLen
        s = totLen-j;
        full( i, find(neighSums==s) ) = tot(i,j);
    end
end
===== End of file "tot2full.m"


===== File "tot2fullk.m":
function full = tot2fullk( tot, k )

%@@ Convert a totalistic rule to a fully specified rule for any k

% Returns fully specified CA rule(s) when given rule(s) in totalistic
% form. A totalistic rule is of length m+1 where m is the neighborhood size. The
% full rule is of size k^m.

% tot - totalistic rule(s), each row is a rule
% k - number of states (colors)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% tot2full.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ rows, totLen ] = size( tot );
fullLen = k^((totLen-1)/(k-1));
fullNeighs = dec2baseV( fliplr( 0:fullLen-1 ), k )';
neighSums = sum( fullNeighs );
full = zeros( rows, fullLen );
for i = 1:rows
    for j = 1:totLen
        s = totLen-j;
        full( i, find(neighSums==s) ) = tot(i,j);
    end
end
===== End of file "tot2fullk.m"


===== File "totSlideShow.m":
%@@ Slide show of 136 equivalent k=2 r=3 totalistic rules

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% totSlideShow.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

totEq = equivTot(0);
[ rows numGroups ] = size(totEq);
rules = [];
labels = {};
for i = 1:numGroups
    m = cell2mat(totEq(i));
```

```
    rules(i) = m(1);
    labels(i) = { num2str(m(1)) };
end

slideshow( tot2full(dec2binV(rules,8)), 7, labels )
===== End of file "totSlideShow.m"


===== File "trainNetElementary.m":
function [net, tr] = trainNetElementary()

%@@ Train NN using elementary CA to classify into Li-Packard classes

% net - neural network trained to classify elementary CA into 6 Li-Packard
%       classes based on 8 parameters (mean field parameter is actually 4
%       parameters, bringing the total to 11).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% trainNetElementary.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% get parameter values for each of 11 parameters (each row a param)
paramVals = readElementaryParams';

% COMPENSATATIONS %%%
paramValsCOMP= paramVals(1:7,:); % without mean field values
% add more instances of the 2 elementary complex CA
numMore = 5;
moreRule110Param = repmat( paramValsCOMP(:,111), 1, numMore );
moreRule54Param = repmat( paramValsCOMP(:,55), 1, numMore );
paramValsCOMP = [ paramValsCOMP moreRule110Param moreRule54Param ];
%%%

% get classification of each of the 256 elementary CA
class = classesTargets;

% COMPENSATAIONS
classCOMP = class;
% compensate for rules 100(fixed point) = 74(two-cycle)
%                      155(two-cycle) = 181(periodic)
% parameter sets should be half in each class (.5 target)
%classCOMP([2 3], [101 75]) = 0.5;
%classCOMP([3 4], [156 182]) = 0.5;
% add more instances of the 2 elementary complex CA
moreRule110Target = repmat( class(:,111), 1, numMore );
moreRule54Target = repmat( class(:,55), 1, numMore );
classCOMP = [ classCOMP moreRule110Target moreRule54Target ];
%%%

%bipolar option
classBipolar = 2*class - 1;

% create and train feedforward net
% 7 inputs (parameters), between 0 and 1
% 6 ouputs (classes), between 0 and 1 (correct class should output 1, others 0)
net = newff( [ zeros(7,1) ones(7,1) ], [ 50, 50, 6 ], ...
```

```
            {'tansig', 'tansig', 'logsig'}, 'trainrp' );
net.trainParam.show = 50;
net.trainParam.epochs = 2000;
%net.trainParam.goal = 1e-3; %would be 1e-5, but rules 100,74,155,181 are 0.5
net.trainParam.goal = 1e-2;
[net, tr] = train( net, paramValsCOMP, classCOMP ); % TRAIN - without meanfield
===== End of file "trainNetElementary.m"


===== File "trainNetSubset.m":
function [net, perf] = trainNetSubset(paramSubset, type)

%@@ Train NN a given subset of parameters

% net - neural network trained to classify elementary CA into 6 Li-Packard
%       classes based on 8 parameters (mean field parameter is actually 4
%       parameters, bringing the total to 11).
% tr  - training results
% type - either "elem" or "tot" for training with either elementary CA or
%        totalistic k=2 r=3 CA

% paramSubset - list of subset of parameters to train on

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% trainNetSubset.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

numParams = length(paramSubset);

% get values for each parameter (each row a param)
% and get classification of each of the 256 elementary CA
if( strcmp(type, 'elem') )
    paramVals = readElementaryParams';
    paramVals = paramVals(paramSubset,:); % subset of parameters
    class = classesTargets;
else
    paramVals = readTotParams';
    paramVals = paramVals(paramSubset,:); % subset of parameters
    class = classesTargetsTot;
end

% create and train feedforward net
% numParam inputs, between 0 and 1
% 6 ouputs (classes), between 0 and 1 (correct class should output 1, others 0)
net = newff( [ zeros(numParams,1) ones(numParams,1) ], [ 30, 30, 6 ], ...
            {'tansig', 'tansig', 'logsig'}, 'trainrp' );
net.trainParam.show = NaN;
net.trainParam.epochs = 500;
net.trainParam.goal = 0;
[net, tr] = train( net, paramVals, class );
perf = tr.perf(net.trainParam.epochs + 1);
===== End of file "trainNetSubset.m"


===== File "trainTest.m":
function [net, outputs, trainPerf, testPerf, percentCor, percentCorByClass] ...
```

```
            = trainTest(type)

%@@ Train and test a NN with subsets of elem or tot CA

% type - either 'elem' or 'tot' for training with either elementary CA or
%        totalistic k=2 r=3 CA

% net - trained NN
% perf - NN performance (mean squared error)
% percentCor - percent of CA correctly classified in testing set
% percentCorByClass - percent of CA correctly classified, listed by class

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% trainTest.m
% Dan Kunkle
% June 2003
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

numParams = 7;
% get values for each parameter (each row a param)
% and get classification of each of the 256 elementary CA
if( strcmp(type, 'elem') )
    paramVals = readElementaryParams';
    paramVals = paramVals(1:numParams,:);
    class = classesTargets;
    classComb = classesCombined;
    [ null, fixedPoint, twoCycle, periodic, complex, chaotic ] = classes;
else
    paramVals = readTotParams';
    paramVals = paramVals(1:numParams,:);
    class = classesTargetsTot;
    classComb = classesCombinedTot;
    [ null, fixedPoint, twoCycle, periodic, complex, chaotic ] = classesTot;
end

% split into testing and training sets
% put half of rules into test, half into train (split even over 6 classes)
trainSet = [];
testSet = [];
classCell = { null, fixedPoint, twoCycle, periodic, complex, chaotic };
for i = 1:6
    numRules = length(classCell{i});
    half = ceil(numRules/2);
    randOrder = randperm(numRules);
    trainSet = [ testSet classCell{i}(randOrder(1:half)) ];
    testSet = [ trainSet classCell{i}(randOrder(half+1:numRules)) ];
end

%organize inputs and outputs into training and testing sets
trainParams = paramVals(:,trainSet+1);
trainClass = class(:,trainSet+1);
testParams = paramVals(:,testSet+1);
testClass = class(:,testSet+1);
testClassComb = classComb(testSet+1);

% create and train feedforward net
% numParam inputs, between 0 and 1
```

```
% 6 ouputs (classes), between 0 and 1 (correct class should output 1, others 0)
net = newff( [ zeros(numParams,1) ones(numParams,1) ], [ 30, 30, 6 ], ...
             {'tansig', 'tansig', 'logsig'}, 'trainrp' );
net.trainParam.show = NaN;
net.trainParam.epochs = 1000;
net.trainParam.goal = 1e-3;
[net, tr] = train( net, trainParams, trainClass );
trainPerf = tr.perf(length(tr.perf));

% test trained NN
[outputs, Pf, Af, errors, testPerf] = sim( net, testParams, [], [], testClass );
outputs = outputs';

%count how many net got right
numTrials = length(testSet);
count = 0;
numPerClass = zeros(1,6);
classCount = zeros(1,6);
classRound = round(outputs*100);
for i = 1:numTrials
    %maximum net output value
    maxOutVal = max(classRound(i,:));
    %set of outputs with maximum value
    maxOuts = find(classRound(i,:) == maxOutVal);

    numPerClass(testClassComb(i)) = numPerClass(testClassComb(i)) + 1;
    if( find(maxOuts == testClassComb(i)) )
        count = count + 1;
        classCount(testClassComb(i)) = classCount(testClassComb(i)) + 1;
    end
end
percentCor = count/numTrials;
percentCorByClass = classCount./numPerClass;
===== End of file "trainTest.m"


===== File "trainTestRuns.m":
function [percentCorElem, percentCorByClassElem, ...
          percentCorTot, percentCorByClassTot] = trainTestRuns(numRuns)

%@@ Perform multiple train/test runs of NN for elem and tot CA

% numRuns - number of train/test sessions to average stats over

% percentCor - percent of CA correctly classified in testing set
% percentCorByClass - percent of CA correctly classified, listed by class

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% trainTestRuns.m
% Dan Kunkle
% June 2003
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

percentCorElem = 0;
percentCorByClassElem = 0;
percentCorTot = 0;
percentCorByClassTot = 0;
for i = 1:numRuns
```

```
    [net, outputs, trainPerf, testPerf, percentCor, percentCorByClass] = ...
        trainTest('elem');
    percentCorElem = percentCorElem + percentCor;
    percentCorByClassElem = percentCorByClassElem + percentCorByClass;

    [net, outputs, trainPerf, testPerf, percentCor, percentCorByClass] = ...
        trainTest('tot');
    percentCorTot = percentCorTot + percentCor;
    percentCorByClassTot = percentCorByClassTot + percentCorByClass;
end
percentCorElem = percentCorElem / numRuns;
percentCorByClassElem = percentCorByClassElem / numRuns;
percentCorTot = percentCorTot / numRuns;
percentCorByClassTot = percentCorByClassTot / numRuns;
===== End of file "trainTestRuns.m"


===== File "uncomp.m":
function f = uncomp( bin )

%@@ The number of contiguous blocks in a binary array

% returns the number of contiguous blocks (of 1s and 0s) in the given vector
% (minus 1). Produces a value in the range [0,size of bin - 1]. This is a
% simple approximation of the "uncompressability" of the string

% bin - a vector of 1s and 0s representing a binary string

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% uncomp.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ row col ] = size( bin );
f = sum( bin(2:col) ~= bin(1:col-1) );
===== End of file "uncomp.m"


===== File "upsilon.m":
function up = upsilon( rule, m )

%@@ Calculates upsilon (incompressibility) parameter for a CA rule

% Returns my incompressibility parameter for a CA rule - basically the number
% of contiguous blocks in a particular ordering of the rule

% rule - binary CA rule
% m - neighborhood size

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% upsilon.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[sym neg ref] = lexrule2symrule(rule,m);
[ r symLen ] = size(sym);
```

```
[ r nsymLen ] = size(neg);
up = (uncomp(sym) + uncomp(neg) + uncomp(ref)) / (symLen+(2*nsymLen)-3);
===== End of file "upsilon.m"


===== File "upsilonAvg.m":
function up = upsilonAvg( rule, m )

%@@ Upsilon parameter averaged over all equivalent rules

% Returns incompressibility parameter for a CA rule averaged over all
% equivalent rules

% rule - binary CA rule
% m - neighborhood size

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% upsilonAvg.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[sym1 neg1 ref1] = lexrule2symrule(rule,m);
[sym2 neg2 ref2] = lexrule2symrule(negate(rule),m);
[sym3 neg3 ref3] = lexrule2symrule(reflect(rule),m);
[sym4 neg4 ref4] = lexrule2symrule(negate(reflect(rule)),m);
[ r symLen ] = size(sym1);
[ r nsymLen ] = size(neg1);
symAll = [];
symAll(1) = (uncomp(sym1) + uncomp(neg1) + uncomp(ref1)) / (symLen+2*nsymLen-3);
symAll(2) = (uncomp(sym2) + uncomp(neg2) + uncomp(ref2)) / (symLen+2*nsymLen-3);
symAll(3) = (uncomp(sym3) + uncomp(neg3) + uncomp(ref3)) / (symLen+2*nsymLen-3);
symAll(4) = (uncomp(sym4) + uncomp(neg4) + uncomp(ref4)) / (symLen+2*nsymLen-3);

% try concatenating each rule part and testing incompressibility
%symAll(1) = uncomp([ sym1 neg1 ref1]) / (symLen+2*nsymLen-1);
%symAll(2) = uncomp([ sym2 neg2 ref2]) / (symLen+2*nsymLen-1);
%symAll(3) = uncomp([ sym3 neg3 ref3]) / (symLen+2*nsymLen-1);
%symAll(4) = uncomp([ sym4 neg4 ref4]) / (symLen+2*nsymLen-1);

symU = mean(symAll);
%fix round-off errors, use four significant digits
symU = round(symU*1000)/1000;
up = symU;
===== End of file "upsilonAvg.m"


===== File "upsilonType.m":
function up = upsilonType( rule, m, type )

%@@ Calculate several different variants of the upsilon parameter

% Returns incompressibility parameter for a CA rule - basically the number
% of contiguous blocks in a particular ordering of the rule

% rule - binary CA rule
% m - neighborhood size
% type - allows six different types, indexed as 1, 2, 3, 4, 5, and 6
```

```
%   1: return uncompressability of normal rule
%   2: return uncompressability of gray code ordered rule
%   3: return uncompressability of sum ordered rule
%   4: return minimum of uncompressability of gray code and sum rules
%   5: return maximum of uncompressability of gray code and sum rules
%   6: return uncompressability of symmetric neighborhood ordered rule

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% upsilonType.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ row ruleSize ] = size( rule );

normU = uncomp( rule ) / (ruleSize-1);
grayU = uncomp( lexrule2grayrule(rule) ) / (ruleSize-1);
sumU = uncomp( lexrule2sumrule(rule) ) / (ruleSize-1);

switch type
    case {1}
        up = normU;
    case {2}
        up = grayU;
    case {3}
        up = sumU;
    case {4}
        up = min( grayU, sumU );
    case {5}
        up = max( grayU, sumU );
    case {6}
        up = upsilon(rule,m);
end
===== End of file "upsilonType.m"


===== File "upsilonTypeCharts.m":
function [] = upsilonTypeCharts()

%@@ Displays a class bar chart for each upsilon variant

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% upsilonTypeCharts.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rules = dec2binV( 0:255 );
upVals = zeros(6,256);
for i = 1:6
    for j = 1:256
        upVals(i,j) = upsilonType( rules(j,:), 3, i );
    end
end



%chart
```

```
dir = '../results/upsilonCharts/';
names(1,1) = {'lex.eps'};
names(1,2) = {'gray.eps'};
names(1,3) = {'sum.eps'};
names(1,4) = {'min.eps'};
names(1,5) = {'max.eps'};
names(1,6) = {'symm.eps'};
titles(1,1) = {'\upsilon - Incompressibility - Lexicographic Ordering'};
titles(1,2) = {'\upsilon - Incompressibility - Gray Code Ordering'};
titles(1,3) = {'\upsilon - Incompressibility - Sum Ordering'};
titles(1,4) = {'\upsilon - Incompressibility - min(gray,sum)'};
titles(1,5) = {'\upsilon - Incompressibility - max(gray,sum)'};
titles(1,6) = ...
        {'\upsilon - Incompressibility - Symmetric Neighborhood Ordering'};
for i = 1:6
    barChartParam(upVals(i,:), char(titles(1,i)), [ dir char(names(1,i)) ]);
end
===== End of file "upsilonTypeCharts.m"


===== File "writeElementaryParams.m":
function [] = writeElementaryParams( fileName )

%@@ Write parameter values for elementary CA to file

% Writes a matrix of all of the parameters for each elementary CA to a file
% each collumn of the data represents a parameter, each row represents one
% of the 256 elementary CA (in lexicograpical order)

% fileName - file name to write data to

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% writeElementaryParams.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

vals = testElementary( [ 1 1 1 1 1 1 1 1 ] );

% write the data
fid = fopen( [ fileName '.data' ], 'w' );
fprintf( fid, ...
 '%5.4f %5.4f %5.4f %5.4f %5.4f %5.4f %5.4f %5.4f %5.4f %5.4f %5.4f \n', vals );
fclose(fid);

% write an explanation of the data
fid = fopen( [ fileName '.README' ], 'w' );
fprintf( fid, '%s', ...
    [ 'Each row represents an elementary CA, from 0 to 255. ' ...
      'Each column is a parameter: lambda, Z, mu, absolute activity, ' ...
      'neighborhood dominance, activity propigation, upsilon ' ...
      '(incompressibility), meanfield n0, meanfield n1, meanfield n2,' ...
      'meanfield n3.' ] );
===== End of file "writeElementaryParams.m"


===== File "writeParamsPretty.m":
function [] = writeParamsPretty( fileName )
```

```
%@@ Write parameters with additional information

% Writes a matrix of all of the parameters for each elementary CA to a file
% each collumn of the data represents a parameter, each row represents one
% of the 256 elementary CA (in lexicograpical order)

% fileName - file name to write data to

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% writeParamsPretty.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

vals = testElementary( [ 1 1 1 1 1 1 1 0 ] )'; %don't write mean field

% write the data
fid = fopen( [ fileName '.data' ], 'w' );
[ numRows numCols ] = size(vals);
for i = 1:numRows
    linePrefix = [num2str(i-1) ' ' dec2bin(i-1, 8) ' ' getClass( i-1, 0 ) ' '];
    fprintf( fid, [linePrefix ...
     '%3.2f %3.2f %3.2f %3.2f %3.2f %3.2f %3.2f \n'],...
     vals(i,:) );
end
fclose(fid);
===== End of file "writeParamsPretty.m"


===== File "writeTotParams.m":
function [] = writeTotParams( fileName )

%@@ Write totalistic rule parameter values to file

% Writes a matrix of all of the parameters for each totalistic CA with k = 2
% and r = 3 to a file each collumn of the data represents a parameter, each row
% represents one of the 256 totalistic CA (in lexicograpical order)

% fileName - file name to write data to

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% writeTotParams.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

vals = allParams( tot2full( dec2binV( 0:255 ) ), 7 );

% write the data
fid = fopen( [ fileName '.data' ], 'w' );
fprintf( fid, '%5.4f %5.4f %5.4f %5.4f %5.4f %5.4f %5.4f \n', vals );
fclose(fid);

% write an explanation of the data
fid = fopen( [ fileName '.README' ], 'w' );
fprintf( fid, '%s', ...
    [ 'Each row represents a totalistic k=2 r=3 CA, from 0 to 255. ' ...
```

```
      'Each column is a parameter: lambda, Z, mu, absolute activity, ' ...
      'neighborhood dominance, activity propigation, upsilon ' ...
      '(incompressibility)' ] );
===== End of file "writeTotParams.m"


===== File "writeTotParamsPretty.m":
function [] = writeTotParamsPretty( fileName )

%@@ Write totalistic rule parameter values with extra info

% Writes a matrix of all of the parameters for each totalistic CA with k = 2
% and r = 3 to a file each collumn of the data represents a parameter, each row
% represents one of the 256 totalistic CA (in lexicograpical order)

% fileName - file name to write data to

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% writeTotParamsPretty.m
% Dan Kunkle
% October 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

vals = allParams( tot2full( dec2binV( 0:255 ) ), 7 );

% write the data
fid = fopen( [ fileName '.data' ], 'w' );
[ numRows numCols ] = size(vals);
for i = 1:numRows
    linePrefix = [num2str(i-1) ' ' dec2bin(i-1, 8) ' ' ];
    fprintf( fid, [linePrefix ...
     '%3.2f %3.2f %3.2f %3.2f %3.2f %3.2f %3.2f \n'],...
     vals(i,:) );
end
fclose(fid);
===== End of file "writeTotParamsPretty.m"


===== File "z.m":
function zp = z( rule, m )

%@@ Calculates Z parameter for a CA rule

% Returns Wuensche's Z parameter for a binary CA rule
% Z parameter = max( z_left, z_right ) : this is done by
% simply finding z_left twice, once with the normal rule and once with
% a the same rule with reflected neighborhoods

% rule - binary CA rule
% m - size of neighborhood

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% z.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

zp = max( zLeft( rule, m ), zLeft( reflect( rule ), m ) );
===== End of file "z.m"
```

```
===== File "zLeft.m":
function zL = zLeft( rule, m )

%@@ A component of the Z parameter

% Returns Wuensche's Z_left parameter for a binary CA rule
% complete Z parameter = max( z_left, z_right ) : this is done by z.m
% which simply executes z_left twice, once with the normal rule and once with
% a the same rule with reflected neighborhoods

% rule - binary CA rule
% m - size of neighborhood

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% z_left.m
% Dan Kunkle
% September 2002
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[ row ruleSize ] = size( rule );

zL = 0;
negProbs = [];

for i = 1:m
    n = 0;
    countBy = 2^(i-1);
    for j = 1:ruleSize/(2^i)
        start1 = (j-1)*(countBy*2)+1;
        end1 = start1+countBy-1;
        start2 = end1+1;
        end2 = start2+countBy-1;
        t1 = rule( start1 : end1 );
        t2 = rule( start2 : end2 );
        n = n + ( abs( sum(t1) - sum(t2) ) == countBy );
    end
    norm = n/(ruleSize/(2^i));
    zL = zL + ( norm * prod(negProbs) );
    negProbs(i) = 1 - norm;
end
===== End of file "zLeft.m"
```

Bibliography

[1] D. Andre, F. H. Bennett III, and J. R. Koza. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In Koza et al., editor, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 3–11, MIT Press, 9-11 1996.

[2] E. Berlekamp, J. H. Conway, and R. Guy. *Winning Ways for Your Mathematical Plays*. Academic Press, New York, NY, USA, 1982.

[3] P. M. Binder. A phase diagram for elementary cellular automata. *Complex Systems*, 7:241–7, 1993.

[4] E. F. Codd. *Cellular Automata*. Academic Press, New York, 1968.

[5] K. Culik II, L. P. Hurd, and S. Yu. Computation theoretic aspects of cellular automata. *Physica (D)*, 45:357–378, 1990.

[6] K. Culik II and S. Yu. Undecidability of CA classification schemes. *Complex Systems*, 2(2):177–190, April 1988.

[7] R. Das, J. P. Crutchfield, M. Mitchell, and J. E. Hanson. Evolving globally synchronized cellular automata. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 336–343, San Francisco, CA, 1995. Morgan Kaufmann.

[8] H. Demuth and M. Beale. *Neural Network Toolbox: For use with MATLAB: User's Guide*. The Mathworks, Natick, MA, USA, 2002.

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

[10] M. Gardner. Mathematical games: The fantastic combinations of John Conway's new solitaire game 'Life'. *Scientific American*, 223(4):120–123, October 1970.

[11] M. Gardner. Mathematical games: On cellular automata, self-reproduction, the Garden of Eden and the game of 'Life'. *Scientific American*, 224(2):112–117, February 1971.

[12] S. W. Golomb. Run-length encodings. *IEEE Trans. Inform. Theory*, IT-12:399–401, 1966.

[13] D. J. Guan. Generalized gray codes with applications. In *Proceedings of the National Science Council, Part A: Physical Science and Engineering*, pages 841–848, 1998.

[14] H. Gutowitz. Cellular automata and the sciences of complexity. *Complexity*, in press, 1996.

[15] R. W. Hamming. *Coding and information theory*. Prentice-Hall, Englewood Cliffs, New Jersey, 1980.

[16] H. Juille and J. B. Pollack. Coevolving the ideal trainer: Application to the discovery of cellular automata rules. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 519–527, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998. Morgan Kaufmann.

[17] C. G. Langton. Computation at the edge of chaos. *Physica D*, 42, 1990.

[18] W. Li. Phenomenology of nonlocal cellular automata. *Journal of Statistical Physics*, 68(5 / 6):829–882, September 1992.

[19] W. Li and N. Packard. The structure of the elementary cellular automata rule space. *Complex Systems*, 4(3):281–297, June 1990.

[20] W. Li, N. Packard, and C. G. Langton. Transition phenomena in CA rule space. *Physica D*, 45(1–3):77–94, 1990.

[21] H. V. McIntosh. *Linear Cellular Automata*. Universidad Autonoma de Puebla, Mexico, 1987.

[22] M. Mitchell, J. Crutchfield, and R. Das. Evolving cellular automata with genetic algorithms: A review of recent work, 1996.

[23] M. Mitchell, J. P. Crutchfield, and R. Das. Evolving cellular automata to perform computations. In T. Back, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, pages G1.6:1–9. Institute of Physics Publishing and Oxford University Press, Bristol, New York, 1997.

[24] G. M. B. Oliveira, P. P. B. de Oliveira, and N. Omar. Definition and application of a five-parameter characterization of one-dimensional cellular automata rule space. *Artificial Life*, 7(3):277–301, 2001.

[25] G. M. B. Oliveira, P. P. B. de Oliveira, and N. Omar. Searching for one-dimensional cellular automata in the absence of a priori information. *Lecture Notes in Computer Science*, 2159:262–??, 2001.

[26] G. M. B. Oliveira, P. P. B. Oliveira, and N. Omar. Guidelines for dynamics-based parameterization of one-dimensional cellular automata rule spaces. *Complexity*, 6:63–71, 2000.

[27] N. H. Packard. Adaptation towards the edge of chaos. In J. A. S. Kelso, A. J. Mandell, and M. F. Shlesinger, editors, *Dynamic Patterns in Complex Systems*, pages 293–301, Singapore, 1988. World Scientific.

[28] P. Sarkar. A brief history of cellular automata. *ACM Computing Surveys*, 32(1):80–107, March 2000.

[29] M. Sipper. Co-evolving non-uniform cellular automata to perform computations. *Physica D*, 92:193–208, 1996.

[30] J. v. Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, IL, USA, 1966.

[31] S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

[32] S. Wolfram. *Cellular Automata and Complexity: Collected Papers*. Addison-Wesley, 1994.

[33] S. Wolfram. *A new kind of science*. Wolfram Media, Champaign, IL, USA, 2002.

[34] A. Wuensche. Complexity in one-d cellular automata: Gliders, basins of attraction and the z parameter. Technical Report Santa Fe Institute Working Paper 93-03-014, 1994.

[35] A. Wuensche. Classifying cellular automata automatically: Finding gliders, filtering, and relating space-time patterns, attractor basins, and the Z parameter. *Complexity*, 4(3):47–66, 1999.

[36] A. Wuensche and M. Lesser. *The Global Dynamics of Cellular Automata*, volume Ref. Vol. 1 of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley, Reading, MA, 1992.