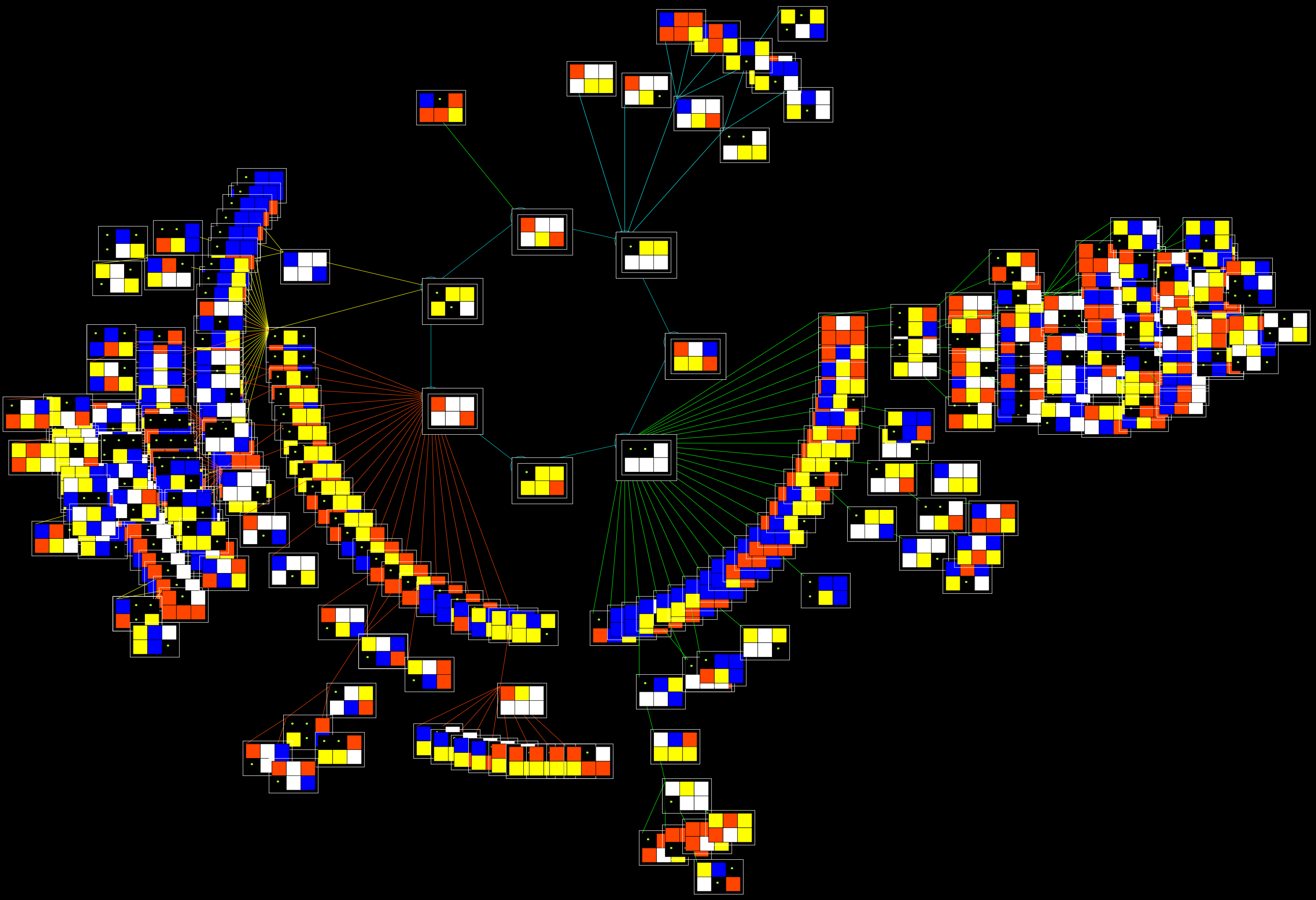
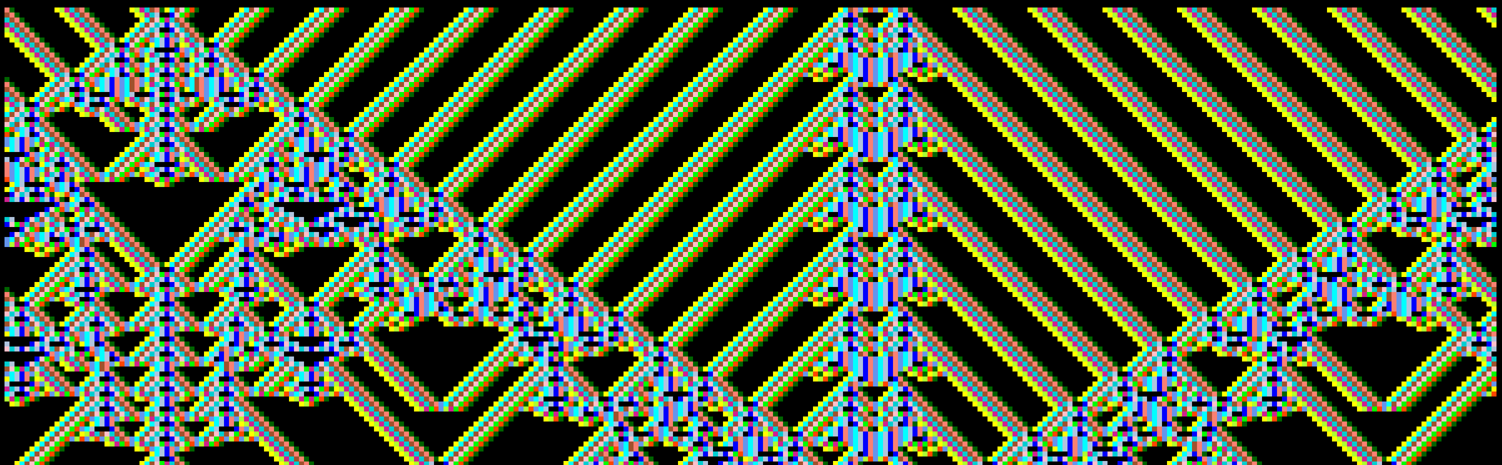


The DDLab Manual

Tools for researching cellular automata, Random Boolean Networks, multi-value discrete dynamical networks, and beyond

Second Edition for Multi-Value Neworks - June 2010

Andrew Wuensche – Discrete Dynamics Lab – www.ddlab.org



Intentionally Blank

Chapters

Intentionally Blank

Preface

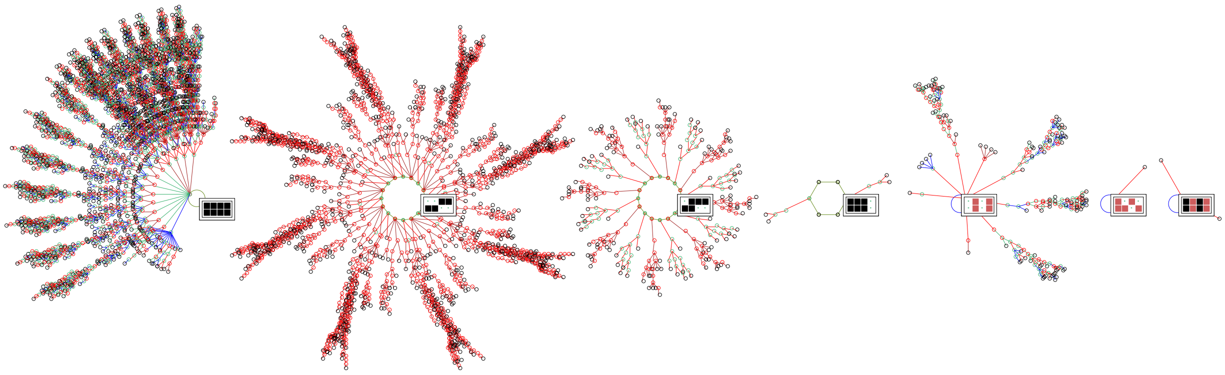


Figure 1: The basin of attraction field of a 3-value Cellular Automaton, $v = 3$, $k = 3$, $n = 8$. There are in fact 17 basins, but equivalent basins have been suppressed leaving just the 7 prototypes. One attractor state is shown for each basin. The rule 020120122021201201011011022.

Dynamics *on* networks, and *of* networks

Networks of sparsely inter-connected elements with discrete cell-states and updating in parallel are central to a wide range of natural and artificial phenomena drawn from many areas of science: from physics to biology to cognition, to social and economic organization; to parallel computation; to emergent self-organisation and artificial life; to complex systems of all kinds.

Dynamics *upon* or *on* a network are its changing patterns of cell-states, while the network "architecture", its connections and logic, remain fixed. DDLab studies these changing patterns and how they "fall" along trajectories into various attractors, revealing the fine detail of basins of attraction.

DDlab also deploys flexible methods for constructing and changing the network's architecture – the dynamics *of* the network – which of course has consequences for the dynamics *on* the network.

To give an oversimplified example, take a network of neurons in a brain; the pattern of firing and synaptic activity is the dynamics *on* the neural network sustaining automatic behavior, whereas network placticity, changes in synaptic micro-circuitry and connections between neurons, is the dynamics *of* the network supporting learning and lifetime adaptation.

Networks like this are sometimes called "decision-making" because each element "decides" how to change based on the signals arriving from other elements along its input connections, mediated by its own internal logic, and each element provides outputs to other elements making their "decisions".

The many decisions make a collective by simultaneous iteration and massive feedback creating the the possibillity of all sorts of "emergent" dynamic patterns.

We can unravel the dynamics and gain some insight into the basic principles by studying idealized networks were all aspects are discretized: value, space, and time, where dynamics *on* the network are simplified into discrete time-steps updating synchronously, in parallel, or in some partial order.

Cellular automata and random Boolean networks are the most common idealized dynamical networks that have been studied, and together with their applications support a very large body of literature. Applications include complex systems and emergent patterns[18, 19], collision-based computation[1], neural networks, genetic regulatory networks[12, 15, 9, 22, 25], and theories of networks in general. The idealized networks are interesting in themselves as mathematical/physical/dynamical systems. Because the dynamics resist analysis by classical mathematics, computer simulation, a kind of experimental mathematics, is unavoidable.

DDLab software

DDLab is able to construct these networks and investigate many aspects of their dynamical behavior. The software utilises interactive graphics and other methods to study cellular automata (CA), random Boolean networks (RBN)[12], and the general case of discrete dynamical networks (DDN), where the "Boolean" attribute is extended to multi-value. Together with hybrid networks, and networks of interacting sub-networks, there is a huge diversity of behaviour that can be investigated - mostly terra incognita.

There are currently versions of DDLab for Linux, Unix, Irix, Mac, Cygwin and DOS. The code is open source and written in C.

As well as generating space-time patterns in one, two or three dimensions, DDLab generates attractor basins, graphs that link network states according to their transitions, representing "global" dynamics[19], analogous to Poincaré's "phase portrait" which provided powerful insights in continuous dynamical systems. A key insight is that the dynamics on the networks converge, thus fall into a number of basins of attraction. The graphs, consisting of trees rooted on attractor cycles, represent the network's memory, its ability to hierarchically categorize its patterns of activation (state-space), as a function of the precise network architecture[22].

Relating this to space-time patterns in CA, high convergence implies order, low convergence implies disorder or chaos[19]. The most interesting emergent structures occur at the transition, sometimes called the "edge of chaos"[14, 27].

DDLab is an applications program, it does not require writing code. Network parameters and the graphics presentation can be flexibly set, reviewed and altered, including changes "on-the-fly". A wide variety of data, measures, analysis, and statistics are available. DDLab produces graphic images of all aspects of the data, the networks, attractor basins, and moving images of space-time patterns. Many of the images can be captured as PostScript files for publication. This manual provides a comprehensive guide.

Update: multi-value, totalistic rules, vector graphics

This edition of the manual is updated to cover the latest version of DDLab. Apart from many other new features, improvements and bug fixes, the significant changes from the old manual are as follows:

- DDLab is generalized for multi-value logic. Up to 8 values (or cell-states, or colors) are now possible, instead of just Boolean logic (two values: 0,1). Of course, with just 2 values selected, DDLab behaves as before[32]. The multi-value version opens up new possibilities for dynamical behavior and modeling.
- DDLab can be costrained to run "forward-only" for various types of totalistic rules which depend on just the totals of each value in a neighborhood, including outer-totalistic rules and reaction-diffusion rules. This reduces the relative sizes of lookup tables allowing larger neighborhoods (max-k=25, instead of 13 as before) at the cost of disabling basin of attraction functions. In 2d the neighborhoods are predefined to make hexagonal as well square lattices. Many interesting cellular automaton rules with "Life"-like and other complex dynamics can be found in totalistic multi-value rule-space, in 3d as well as 2d[33, 34].
- Most of DDLab's graphic output, including space-time patterns and attractor basins, the network-graph and the attractor jump-graph, can be captured as vector PostScript files, which is a preferred format for publication than the previously available bitmap images.

Acknowledgments

Many people have influenced DDLab by contributing ideas, suggesting new features, providing encouragement, criticism, and helping with programming. I reserve all the blame for its shortcomings. I would like to thank Mike Lesser, Grant Warrell, Crayton Walker, Chris Langton, Stuart Kauffman, Wentian Li, Pedro P.B. de Oliveira, Inman Harvey, Phil Husbands, Guillaume Barreau, Josh Smith, Raja Das, Christian Reidys, Brosl Hasslacher, Steve Harris, Simon Frazer, Burt Voorhees, John Myers, Roland Somogyi, Lee Altenberg, Andy Adamatzky, Mark Tilden, Rodney Douglas, Terry Bossomaier, Ed Coxon, Oskar Itzinger, Pietro di Fenizio, Pau Fernandez, Ricard Sole, Paolo Patelli, Jose Manuel Gomez Soto, Dave Burraston, and many other friends and colleagues (to whom I apologize for not listing). Also DDLab users who have provided feedback, and researchers and staff at the Santa Fe Institute in the 1990s

Intentionally Blank

Contents

List of Figures

List of Tables

Intentionally Blank

Chapter 1

Overview

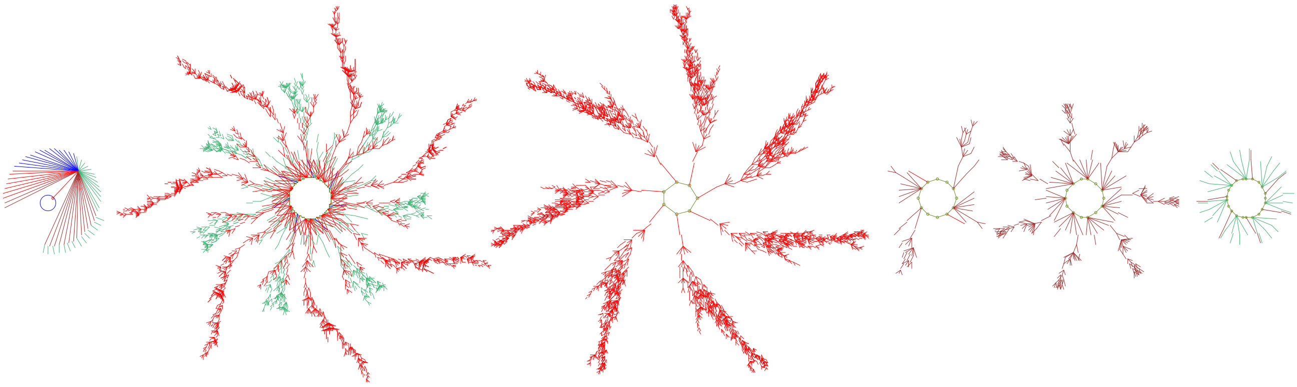


Figure 1.1: The basin of attraction field of a Cellular Automaton, $v = 2$, $k = 3$, $n = 14$. There are in fact 15 basins, but equivalent basins have been suppressed leaving just the 6 prototypes. State nodes have been omitted. This is one of the computationaly universal "elementary" rules, 193 (equivalent to the famous rule 110[17])
.

1.1 Introduction

DDLab is interactive graphics software, able to construct, visualize and manipulate a hierarchy of discrete systems: Cellular Automata, Random Boolean Networks, Discrete Dynamical Networks in general, intermediate or hybrid networks, and random maps — and investigate and visualize many aspects of the dynamics *on* the networks, both from the time-series perspective — space-time patterns, and from the state-space perspective — attractor basins, with interactive graphical methods, as well as data gathering, analysis, and statistics. "Attractor basins" refers to any type of state transition graph: basin of attraction fields, single basins, or subtrees.

Figure 1.2 gives a glimpse of the main themes in DDLab, and also the broad and slippery categories of the systems available (also figure 29.5),

- CA: Cellular Automata: a local neighborhood of k inputs (1D, 2D, 3D) and one rule (but possibly a mix of rules to extend the definition).

- RBN: Random Boolean Networks: random wiring of k inputs (but possibly with mixed k) and a mix of rules (but possibly just one rule).

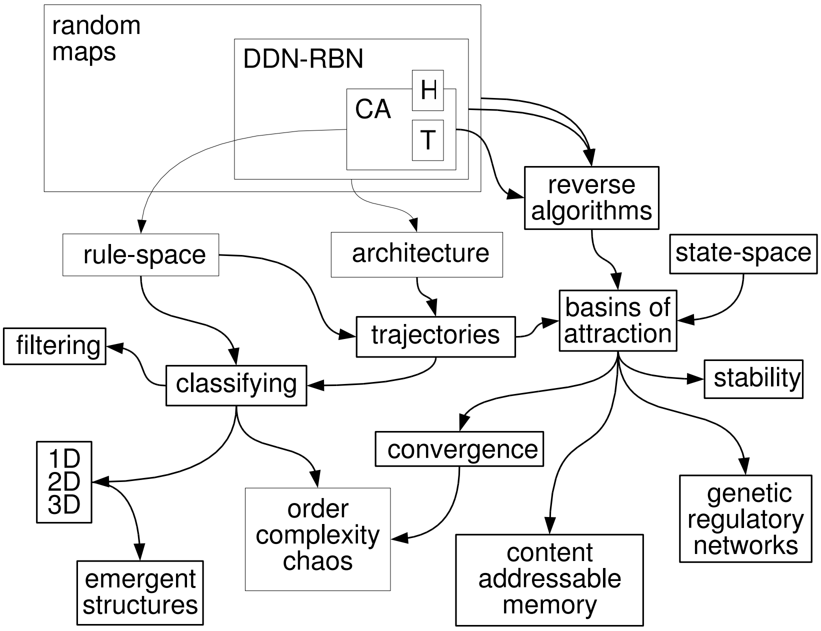


Figure 1.2: The various themes, methods, functions and applications in DDLab, loosely connected. *Top Left*: The expanding hierarchy of networks: CA → RBN/DDN → within the super-set of random maps (directed graphs with out-degree one), imposing decreasing constraints on the dynamics. There are also multiple sub-categories, for example totalistic rules (T), hybrids (H) and networks of networks.

- DDN: Discrete Dynamical Networks: as RBN, but allowing a value-range $v \geq 2$. Binary CA are a special case of RBN, and RBN and multi-value CA are special cases of DDN.

- Random maps: directed graphs with out-degree one, where each state in state-space is *assigned* a successor. CA, RBN and DDN, which are usually sparsely connected ($k \ll n$) are all special cases of random maps, but if fully connected ($k = n$), rule-mix CA and the rest are equivalent to random maps.

DDLab is used widely in research and education, and has been applied to study complexity, emergence and self-organization[19, 27, 34], in the behavior of bio-molecular networks such as neural[21, 23] and genetic[12, 15, 9, 25] networks, and in many other disparate areas. The results in [19]-[34] may be implemented with DDLab.

As well as scientific applications, DDLab's imagery has featured in art exhibitions[28], as the light show at raves, and has been applied for generative music[4].

1.2 Source code and platforms

DDLab is free shareware for personal users. Other users, including commercial users, companies, government agencies, research or educational institutions are required to register. The DDlab code is open source under the GNU General Public License (GPL) as published by the Free Software Foundation.

Compiled versions of DDLab are maintaned for UNIX based opperating systems, and also for DOS, as follows:

- Linux/PC: compiled in Ubuntu 6.06.
- UNIX/XWindows/Sun: compiled in SunOS 5.8. The libraries `libX11.so.6.1` and `libsunmath.so.1` need to be accessible in your system.
- MAC OSX with X11: compiled in Mac OSX 10.4.10.
- Irix/SGI: compiled for IRIX 6.5.27 with MIPSpro C 7.4.3m and -n32. Thanks to Oskar Itzinger.
- CygwinX/Windows/PC: this is a Linux environment running in MSWindows98 and above. Cygwin/X must be installed with the default packages, plus xorg-x11-devel, xorg-x11-fnts.
- DOS/PC: compiled with WatcomC version 11. MSWindows users will find CygwinX performs better than DOS, so DOS may not be maintained or supported to the same extent as the other platforms. There is no native MSWindows version of DDLab at present.

Ths manual covers all versions, which function in essentially the same way, though aspects of the graphics presentation might be slightly different. The manual will refer to the Linux/UNIX/MAC/Irix/CygwinX versions as Linux-like, as opposed to the DOS version.

DDLab is in the process of continual development. New features are being added in response to various research needs. Some aspects of this manual may be out of date or not applicable to particular platforms.

For information on the latest versions, downloading, installing, documentation, licensing, platforms, source code, updates, examples of output, applications, related publications, etc., consult one of the DDLab web sites below:

`www.ddlab.org`
`www.cogs.susx.ac.uk/users/andywu/ddlab.html` (Univ. of Sussex)
`http://uncomp.uwe.ac.uk/wuensche/ddlab.html` (Univ. of the West of England)

1.3 Discrete dynamical networks

A discrete dynamical network in DDLab can be imagined as a software simulation of a collection light bulbs which transmit information to each other about their color state (on/off for binary), and change color according to arriving signals. More abstractly, the network is made up of automatons, elements[1] or "cells", connected to each other by directed links or "wires", where a wire has an input and output terminal. A cell takes on a value (or color, or cell-state), and transmits this value down its output wires. Its value is updated as a function of the values on its input wires.

[1] The term "cell" (as in cellular automata) denotes a network "element", and should not be confused with a biological cell. In random Boolean network models of genetic regulatory networks, a network element represents a gene, whereas in neural network models, a network element represents a neuron.

Updating is usually done in parallel, synchronously, in discrete time-steps, but may also be sequential in a predetermined or random partial order. These dynamics are deterministic, but noise can be added.

This is the system in a nutshell. It remains to set up the network according to its various parameters,

- The "value-range", v: The range of values (cell-states) available to a cell. In other words, the number of possible internal states of the cell, or colors, or letters in its "alphabet". In early versions of DDLab this was limited to just 2 values (0,1), but can now be selected from 2 to 8.
- The system size, n: The number of network elements. $n \leq 31$ for basin of attraction fields, or $n \leq 65025$ otherwise.
- The network "geometry": How the elements are arranged in space: in a 1d, 2d or 3d lattice with axial dimensions i, j, h, or some other arrangement. The network geometry may have real meaning (depending on the "wiring scheme" below), or it may simply allow convenient indexing and representation.
- The connectivity: The number of input wires, *k*, to each cell, or the "*k*-mix" if *k* is not homogeneous. *k*, may vary from 0 to 13, or to 25 for totalistic rules. Maximum *k* is reduced for greater value-range *v*. The *k*-mix and "wiring scheme" (below) can be biased, for instance to make scale-free networks.
- The "wiring scheme": defining the location of the output terminals of each cell's input wires, the element's "neighborhood". Cellular automata have a homogeneous nearest neighbor (or next nearest etc.) local neighborhood throughout the network, in 1d, 2d or 3d. RBN and DDN may have a completely arbitrary wiring scheme (a "pseudo-neighborhood"). The wiring can be assigned at random, or may be biased - for example, by confining an element's pseudo-neighborhood close to itself. The wiring can be assigned to create a hybrid of CA/DDN, or a network of sub-networks in any combination.

  The wiring scheme defines the lattice and its boundary conditions. CA wiring usually requires "periodic boundary conditions" where lattice edges wrap around to their opposite edges.
- The "rule scheme": the rules or logical functions in the network. Each element applies a rule to its inputs to compute its output. Usually this is made into a look-up table, the "rule-table", listing the outputs of all possible input patterns. Cellular automata have a homogeneous rule scheme, the same rule throughout the network. DDN may have a completely arbitrary rule scheme. The rule scheme can be a CA/DDN hybrid. Rules and rule schemes can be biased in various ways.

  A special case of a rule scheme are the outer-totalistic rules where the rule to be applied depends on the cell value requiring *v* rules. Outer totalistic rules are used to implement reaction-diffusion and game-of-Life (and Life-like) rules.

DDlab is able to create and modify these networks, and graphically represent and analyze both the networks themselves, and the dynamics *on* the networks - the changing patterns made by the complex web of feedback.

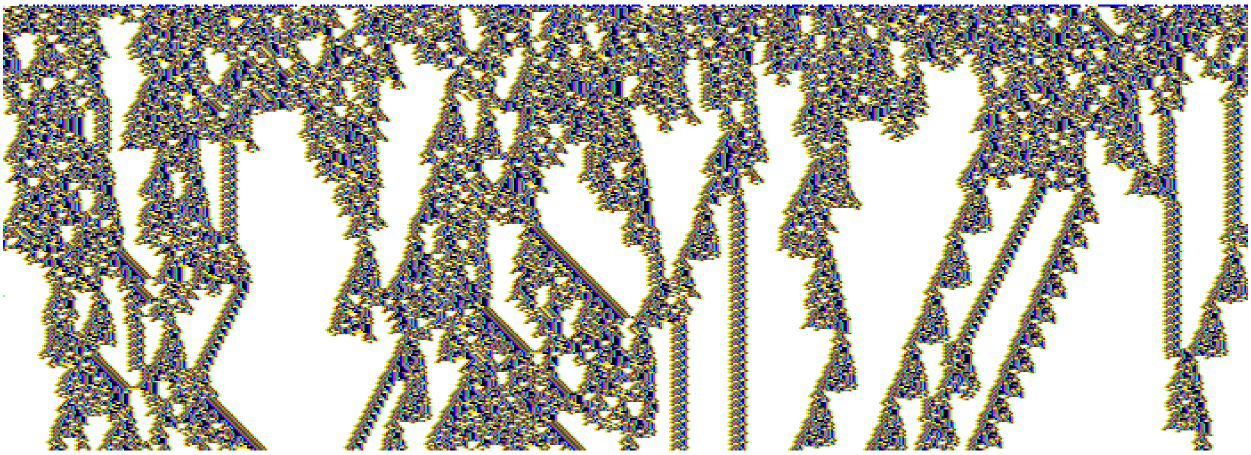


Figure 1.3: The space-time pattern of a 1d complex CA with interacting gliders. 308 time-steps from a random initial state. Value-range n=2, system size n=700, neighborhood size k=7, rule (hex) = 3b 46 9c 0e e4 f7 fa 96 f9 3b 4d 32 b0 9e d0 e0. Cells are colored/shaded according to neighborhood look-up instead of the value. Space is across and time down the page. The basin of attraction field for this rule for n=16 is shown figure 1.4.

Acronym glossary:

- CA: cellular automata: homogeneous wiring/rules; a local universal wiring template (for example: nearest neighbor) and a universal rule, the same rule everywhere.
- RBN: random Boolean networks: $v = 2$, arbitrary wiring and heterogeneous rules, Kauffman's famous model, but possibly with heterogeneous connectivity k.
- DDN: discrete dynamical networks: as RBN, but allowing for a value range $v > 2$ (CA and RBN are special cases of DDN).

1.4 Space-time patterns and attractor basins

DDLab has two alternative ways of looking at network dynamics. *Local* dynamics, running the network forwards, and *global* dynamics, which entails running the network backwards.

Running "backwards" (global dynamics) generates multiple predecessors rather than a trajectory of unique successors. This procedure reconstructs the branching sub-tree of ancestor patterns rooted on a particular state. States without predecessors will be disclosed, the so called "garden-of-Eden" states, the "leaves" of the sub-tree. Sub-trees, basins of attraction (with a topology of trees rooted on attractor cycles), or the entire basin of attraction field (referred to collectively as "attractor basins") can be displayed in real time as directed graphs (state transition graphs), with many alternative presentation options, and methods for gathering/analyzing data. The attractor basins of "random maps" may be generated, with or without some bias in the mapping.

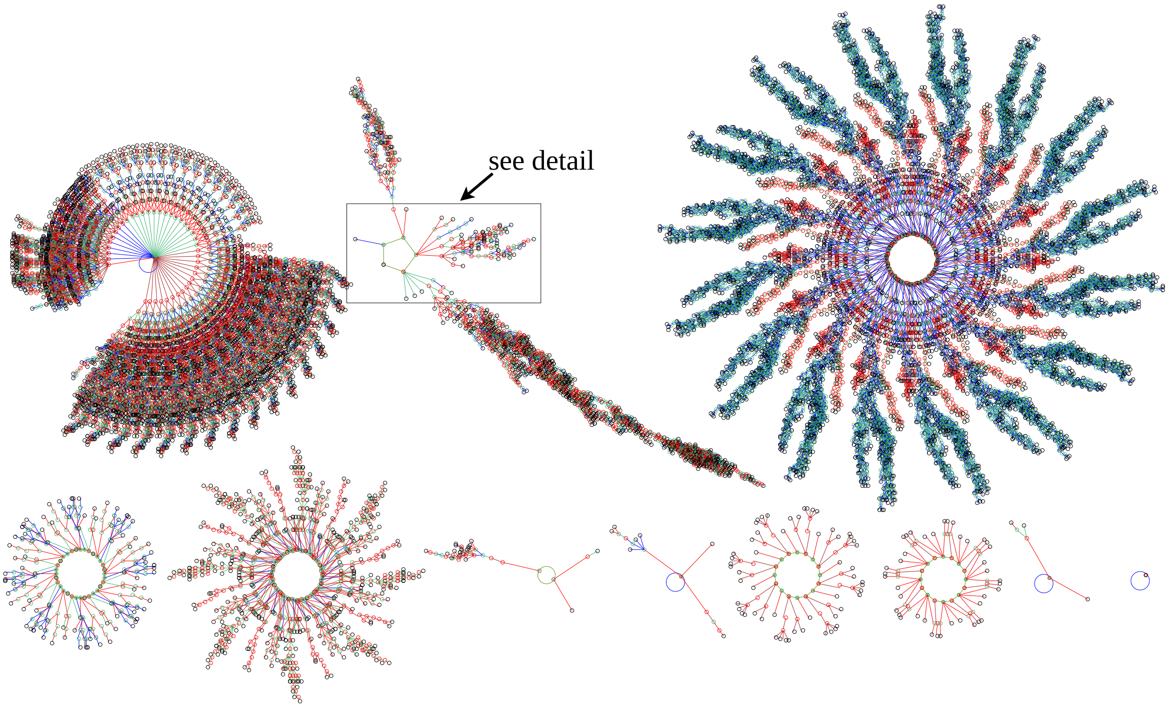


Figure 1.4: The basin of attraction field of a complex CA rule. An example of its space-time patterns are shown in figure 1.3. n=16, k=7. The 2^{16} = 65536 states in state space are connected into 89 basins of attraction. Only the 11 non-equivalent basins are shown, with symmetries characteristic of CA[19]. The period (p), percentage of state space in each basin type(s), and number of each type (t), of the biggest three basins (top row), are as follows: (1) p=1 s=15.7% t=1. (2) p=5 s=55.8% t=16. (3) p=192 s=22.9% t=1. The field's G-density=0.451, λ_{ratio}=0.938, Z=0.578.

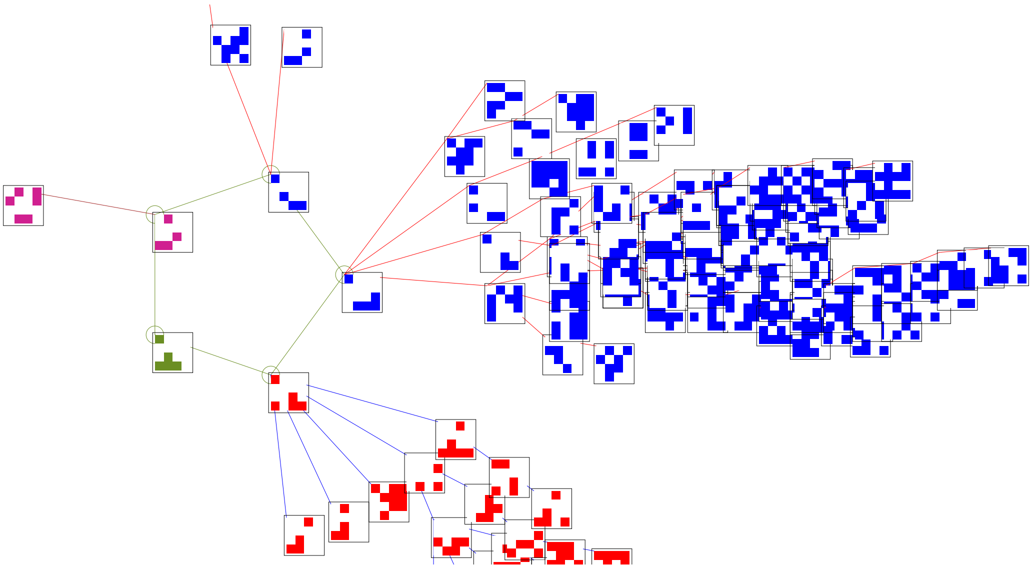


Figure 1.5: A detail of the 2nd basin of attraction in figure 1.4. The states are shown as 4×4 bit patterns.

1.4.1 Totalistic rules - forwards only

DDLab can be constrained to run "forwards only" for various types of totalistic rules which depend on just the totals of each value or color in a neighborhood, including outer-totalistic rules and reaction-diffusion rules. The lookup tables of totalistic rules are much smaller than full lookup tables, so larger neighborhoods are possible (max-k=25, instead of 13) at the cost of disabling basin of attraction functions.

1.5 Categorization

It can be argued that attractor basins represent the network's "memory" by their hierarchical categorization of state-space [20, 22]. Each basin is categorized by its attractor and each sub-tree by its root. Learning/forgetting algorithms allow attaching/detaching sets of states as predecessors of a given state by automatically mutating rules or changing connections. This allows sculpting the basin of attraction field to approach a desired scheme of hierarchical categorization. More generally, preliminary "inverse problem" or "reverse engineering" algorithms are included to find the network that satisfies a given set of transitions.

1.6 Network size limitations

Whereas large networks may be run forward to look at space-time patterns, or backward to look at subtrees, the size is limited when generating the entire basin of attraction field, given that state-space S grows exponentially with system size n ($S = v^n$, where v is the value-range).

DDLab's upper limit for basin of attraction fields is n=31 (lower in practice). Otherwise the limit is n=65025, based on the maximum size of a 2d network, 255×255, where n can be represented by an unsigned short int. This applies to space-time patterns (in 1d, 2d and 3d), and to single basins or sub-trees, though in practice much smaller sizes are appropriate for attractor basins (except in the case of subtrees for maximally chaotic CA "chain-rules"). Larger sizes may be tried, but may impose unacceptable time, memory or display constraints.

Networks with disordered (chaotic) dynamics, which have low in-degree or branchiness in their subtrees (such as CA chain-rules) allow backwards computation of much larger networks than for ordered dynamics which have high in-degree. For CA, rules giving chaotic dynamics have a high Z parameter, rules giving ordered dynamics have a low Z parameter[27].

For RBN and DDN, running backwards generally imposes a greater computational load than for CA.

1.7 Parameters and options

DDLab is an applications program, it does not require writing code. The network's parameters, and the graphics display and presentation, can be very flexibly set, reviewed and altered from DDLab's graphical user interface.

Changes can be made on-the-fly, including changes to rules, connections, current state, scale, and alternative presentations highlighting different properties of space-time patterns. Networks of whatever dimension can be interchangeably represented in 1d, 2d, and 3d. 2d dynamics can be shown with a time dimension (2d+time) in isometric projections.

The network architecture, states, data, and the screen image can be saved and loaded in a variety of tailor-made file formats, and most graphic output can be saved as vector PostScript files.

1.8 Measures and data

Various quantitative, statistical and analytical measures and data on both forward dynamics and attractor basin topology are available in DDLab, as well as various global parameters for rules and network architecture. The measures and data, shown graphically as well as numerically in most cases, include the following:

- Rule parameters: λ, P, Z.
- The frequency of canalizing inputs. This can be set to any arbitrary level.
- Various measures on forward dynamics such as pattern density, frozen islands, damage spread between two networks, the Derrida plot, rule-table lookup frequency (which allows "filtering"), input entropy, and the variance of the input entropy, which allows ordered, complex and chaotic rules to be classified automatically[27].
- Various global measures on the topology of attractor basins including garden-of-Eden density and in-degree frequency.

1.9 Contents summary

- Chapter 2 provides a descriptive summary of DDLab's functions.
- Chapter 3 describes how DDlab can be accessed.
- Chapter 4 describes DDLab's graphical user interface and gives some "quick start" examples. Its probably a good idea to try these right away to get the flavor of DDLab before reading on, or tackling the detailed reference manual.
- Chapters 5 to 35 contain the detailed reference manual.

For further background on the attractor basins of CA, RBN, DDN, and their implications, see references [19]-[33] in the bibliography, which are available at:

www.cogs.susx.ac.uk/users/andywu/publications

Chapter 2

Summary of DDLab functions

This chapter provides a descriptive summary of DDLab's functions.

2.1 DDLab's prompts

DDLab's graphical user interface (chapter 5) allows setting, viewing and amending network parameters, and various presentation and analysis functions, by responding to prompts or accepting defaults.

The prompts present themselves in a main sequence and also in a number of context dependent prompt windows. You can backtrack to previous prompts, and in some cases skip forward. A flashing cursor indicates the current prompt. "Return" (or the left mouse button) steps forward through the prompts, "q" (or the right mouse button) backtracks, or interrupts a running process (a run), such as space-time patterns or attractor basins being generated. There are also on-the-fly changes that can be made during a run.

2.2 Initial choices

Some initial choices in the prompt sequence set the stage for all subsequent DDLab operations, and cannot be amended later (or not easily) without backtracking. These include the following,

2.2.1 Totalistic rules, forwards-only, TFO-mode

There is a choise to constrain DDLab to run "forwards-only" for various types of totalistic rules and reaction diffusion rules (TFO-mode). This reduces memory load by cutting out full look-up tables and all attractor functions (prompts for these will not be displayed), and allows larger neighborhoods, k (max-k=25, instead of 13). Choosing TFO-mode can only be made at the first prompt (chapter 6).

2.2.2 Basin field or initial state

If DDLab is not constrained in TFO-mode as above in section 2.2.1, there is a further choice,

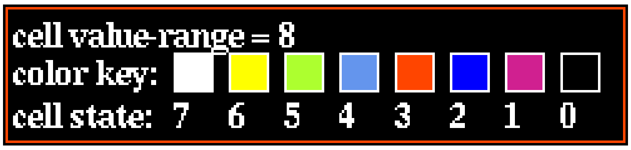
- either to show the basin of attraction field, which does not require an initial state (Field mode).
- or to show anything else which *does* require an initial state: space-time patterns, a single basin of attraction, or a subtree (Seed mode).

This choise can also be amended in at a later stage, in section 30.4.

2.3 Setting the value-range

The value-range v can be set from 2 to 8. If v=2 DDLab behaves as in the old binary version. Note that as v is increased, the size of max-k will decrease, but this also depends on whether DDLab was constrained to run "forwards-only" for totalistic rules (TFO-mode) in section 2.2.1. For example, for v=8 and unconstrained DDLab, max-k=4 to handle the large lookup table. In TFO-mode max k=11. Max-k for different values of v, for both cases, are given in section 2.5.

The selection of the value-range can only be made at this early stage in the program.



cell value-range = 8
color key: ⬜ 🟨 🟩 🟦 🟧 🟦 🟪 ⬛
cell state: 7 6 5 4 3 2 1 0

Figure 2.1: The cell value color key window (for a black background, v=8) that appears when the value-range is selected. The values themselves are indexed from 7 to 0. See figure 7.1 for all the color keys.

2.4 Setting the network size

The network size n for 1d is set early on in the prompt sequence, but this is superseeded if a 2d (i,j) or 3d (i,j,h) network is selected in a subsequent prompt window. Although the size of n for 1d can be increased or decreased by one cell on-the-fly when drawing space-time patterns (section 32.2), in general the network size can only be changed at these early prompts.

For space-time patterns, the upper limit of network size n_{Lim}=65025, based on the maximum size of a 2d network (i,j)=255×255, where n can be represented by an unsigned short int. This limit also applies for single basins and subtrees, though in practice much smaller sizes are appropriate, except when generating subtees for maximaly chaotic CA chain-rules (section16.11).

For basin of attraction fields, however, the upper limit of network size, n_{Lim}, is much smaller, and depends on the value-range v as set out in section 7.3.

2.5 The neighborhood k or k-mix

The size of the neighborhood k, the number of inputs each cell receives, can vary from 0 to k_{Lim}, which depends on the value-range v, and also on whether or not DDLab was constrained to run forwards-only for totalistic rules (TFO-mode, section 2.2.1 above). k_{Lim} for increasing value-range v, for both cases, is set out in (section 7.2).

k can be homogenious, or there can be a mix of k-values in the network. The k-mix may be set and modified in a variaty of ways, including defining the proportions of different k's to be allocated at random in the network, or a "scale-free" distribution, A k-mix may be saved/loaded from a file, but is also implicit in the wiring scheme (section 2.6 below).

2.6 Wiring

The network's wiring scheme (i.e. its connections) has pre-defined layouts for regular CA (for 1d, 2d and 3d) for each neighborhood size, k=1 to max-k (chapter 10). Note, however, that for 3d max-k=13, because 3d neighborhoods greater than 13 have not yet been defined. Regular 3d wiring defines a cubic lattice with periodic boundary conditions. In 2d, the pre-defined neighborhood layouts define a toroidal lattice which can be either square or hexagonal. The square lattice includes the 5 cell vonNeumann neighborhood and the 9 cell Moore neighborhood.

Wiring can also be set at random (non-local wiring), with a wide variety of constraints and biases, or by hand (chapter 12). The pre-defined neighborhoods in this case act as pseudo-neighborhoods to which the rule is applied.

A wiring scheme can be set and amended just for a predefined sub-network within the network, and may be saved/loaded from a file (chapter 19).

Non-local wiring can be constrained in various ways, including confinement within a local patch of cells with a set diameter in 1d, 2d and 3d. Part of the network only can be designated to accept a particular type of wiring scheme, for example rows in 2d and layers in 3d; the wiring can be biased to connect designated rows or layers.

The network parameters can be displayed and amended in a 1d, 2d or 3d graphic format, in a "spread sheet" (chapter 17), or as a network-graph which can be rearranged in various ways, including dragging nodes with the mouse (chapter 20).

2.7 Rules

The most general update logic or rule is expressed as a full lookup table will be refered to as "rcode". There are useful subsets of the general case, two types of totalistic rules, "kcode" and "tcode". The simplest, tcode, depends on the sum of values in the neighborhood. kcode depends on the frequency of each value (color) in the neighborhood. If k=2 tcode and kcode are identical.

In addition, both types of totalistic rules can be made into outer-totalistic rules (also called semi-totalistic), where a different rule applies for each value of the central cell; the game-of-life is one such rule. Outer-totalistic rules also allow implimentation of reaction-diffusion rules or excitable media[7].

As mentioned in section 2.2.1, DDLab can be constrained to run forwards only, for these various types of totalistic rules (TFO-mode), which allows greater $[v, k]$ networks than for a full lookup table. Transformations and mutations would then apply to just the restricted lookup table.

If DDLab remains unconstrained, the totalistic rules (but not outer-totalistic rules) can still be selected, but they will be transformed into the full lookup table (which allows attractor basins). Transformations and mutations will apply to this full lookup table. Within the full lookup table there are also subsets of rules that can be automatically selected at random, including symmetric rules, maximaly chaotic chain-rules, Altenberg rules, and others, and the rules can be biased by various parameters, *lambda*, Z, and canalizing inputs.

A network may have one homogenious rule, as for CA, or a rule-mix as for RBN and DDN. The rule-mix can be confined to a subset of pre-selected rules. Rules may be set and modified in a wide variety of ways, in decimal, hex, as a rule-table bit pattern, at random or loaded from a file. The "game-of-Life", "majority", and other predefined rules or rule biases can be selected (chapters 13, 14, 16).

A rule scheme can be set and amended just for a predefined sub-network within the network, and may be saved/loaded from a file (chapter 19).

Rules may be changed into their equivalents (by reflection and negative transformations), and transformed into equivalent rules with larger or smaller neighborhoods (chapter 18). Rules transformed to larger neighborhoods are useful to achieve finer mutations. Rule parameters λ and Z, and the frequency of canalizing inputs in a network can be set to any arbitrary level (chapter 15).

2.8 Initial network state, seed

An initial network state (a seed) is required to run a network forward and generate space-time patterns. A seed is also required to generate a single basin, by first running forward to find the attractor, then backward from each attractor state. A seed is required to generate a subtree, by simply running backwards from the seed. However, for most CA rules, most states in state-space have no predesessors (they are the leaves of a subtree, "garden-of-Eden" states), so from a random seed its usualy necessary to run forwards by a few steps to penetrate the subtree before running backwards, and an option is provided to do this (section 29.2).

A basin of attraction field does not require setting a seed, because appropriate seeds are automatically provided.

As in setting a rule, there are a wide variaty of methods for defining the seed (chapter 21), in decimal or hex, as a bit pattern in 1d, 2d or 3d, at random (with various constraints or biases), or loaded from a file (see chaper 35).

The bit pattern method is a mini paint program, using the mouse and keyboard to draw colors. Figure 2.2 shows the method applied to draw a portrait, which is then transformed by applying a CA rule for three time-steps.

2.9 Networks of sub-networks

Its possible to create a system of independent or weakly coupled sub-networks within the base network, either directly, or by saving smaller networks to a file, then loading them at appropriate positions in the base network (chapter 19). Thus a 2d network can be tiled with sub-networks, and 1d, 2d or 3d sub-networks can be inserted into a 3d base network.

The parameters of the sub-networks can be totally different from the base network, provided the base network is set up appropriately, with sufficent "memory" to accomadate the sub-network. For example, to load an DDN into a CA, the CA may need be set up as if it were an DDN. To load a mixed-k sub-network into single-k base network, k in the base network needs to be at least as big as the biggest k in the sub-network. Options are available to easily set up networks in this way. Once loaded, the wiring can be fine-tuned to interconnect the sub-networks.

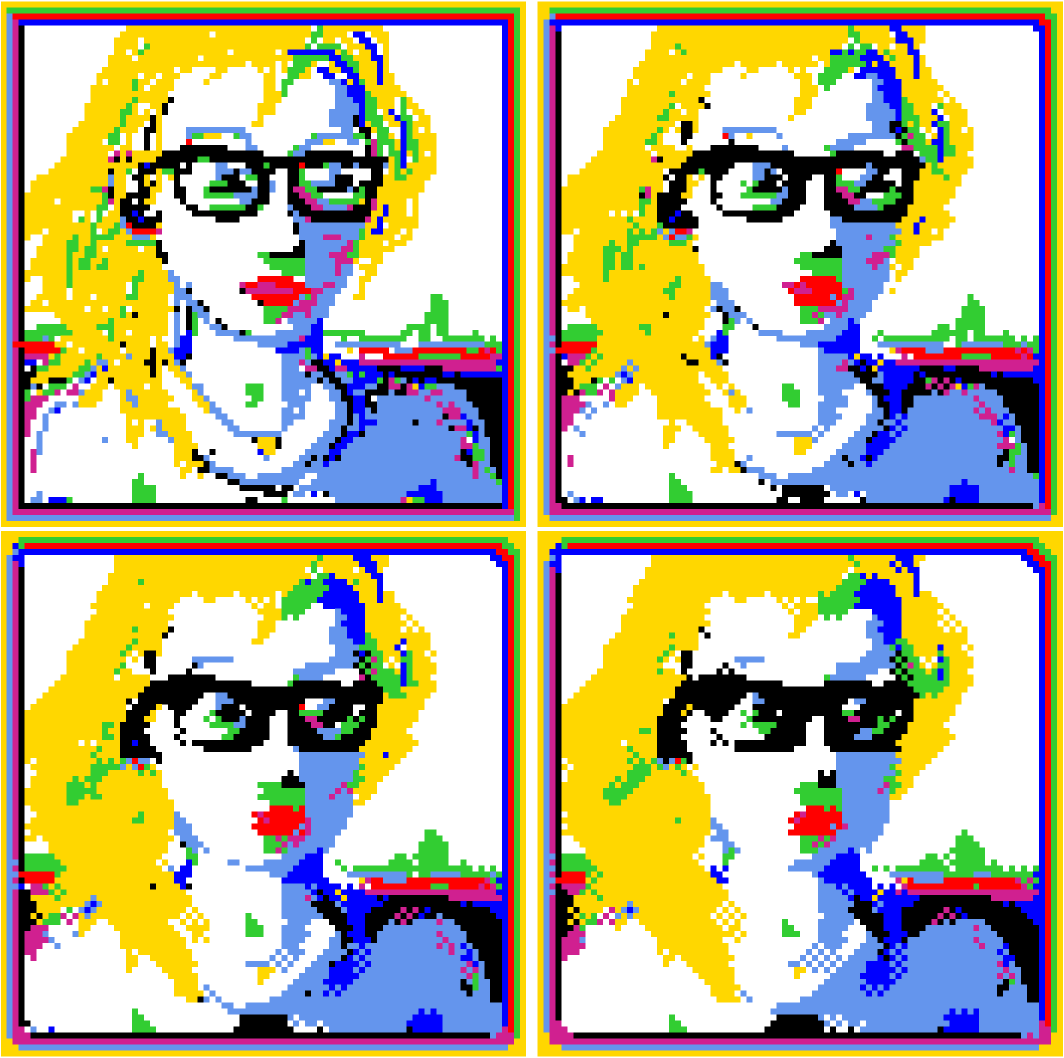


Figure 2.2: Drawing a 2d seed, 88 × 88, with the mouse and keyboard, v=8, k=4, shown top-left. The seed is then transformed by applying a CA rule majority rule for three time-steps.

A network can be automatically duplicated to create a total network made up of two identical sub-networks. This is useful to see the difference pattern (or damage spread) between two networks from similar initial states (section 31.16).

2.10 Presentation options

Many options are provided for the presentation of attractor basins and space-time patterns. Again, many of these settings can be changed "on the fly".

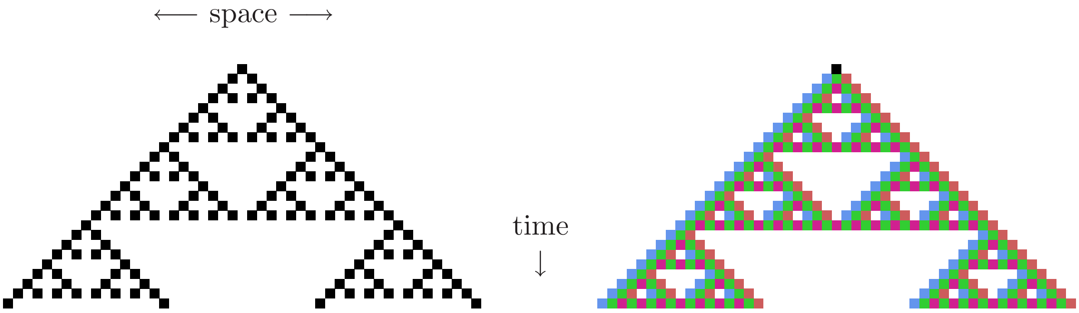


Figure 2.3: Space-time patterns of a 1d CA (v=2, $k = 3$, n=51, rule 90). 24 time-steps from an initial state with a single central 1. Two alternative presentations are shown. *Left*, cells by value. *Right*, cells colored according to their look-up neighbourhood.

2.10.1 Space-time patterns

chapters 31 and 32

A cell in a space-time pattern is coloured according to its value, or alternatively according to a predefined color depending on its neighborhood at the previous time step, the entry in the look-up table that determined the cell's value. A key press will toggle between the two. Space-time patterns can be filtered to suppress cells that updated according to the most frequently occuring neighborhoods, thus exposing "gliders" and other structures (section 32.10.5).

The presentation can be set to highlight cells that have not changed in the previous x generations, where x can be set to any value (section 32.10.1). The emergence of such frozen elements is associated with "canalizing inputs", and underlies Kauffman's RBN model of gene regulatory networks[12, 9].

1d space-time patterns are usual presented as successive time-steps scrolling vertically. 2d networks are presented as a "movie" of successive time-steps, but can also be displayed with a time dimension (2d+time) where successive time-steps are also scrolled, either vertically or diagonally, in isometric projections. 2d networks can be toggled between square and hexagonal layout. 3d networks are presented as a "movie" within a 3d "cage". The presentation of space-time patterns can be switched "on the fly" between 1d, 2d, 2d+time, and 3d, irrespective of their native dimensions. DDLab automatically unravels or bundles up the dimensions.

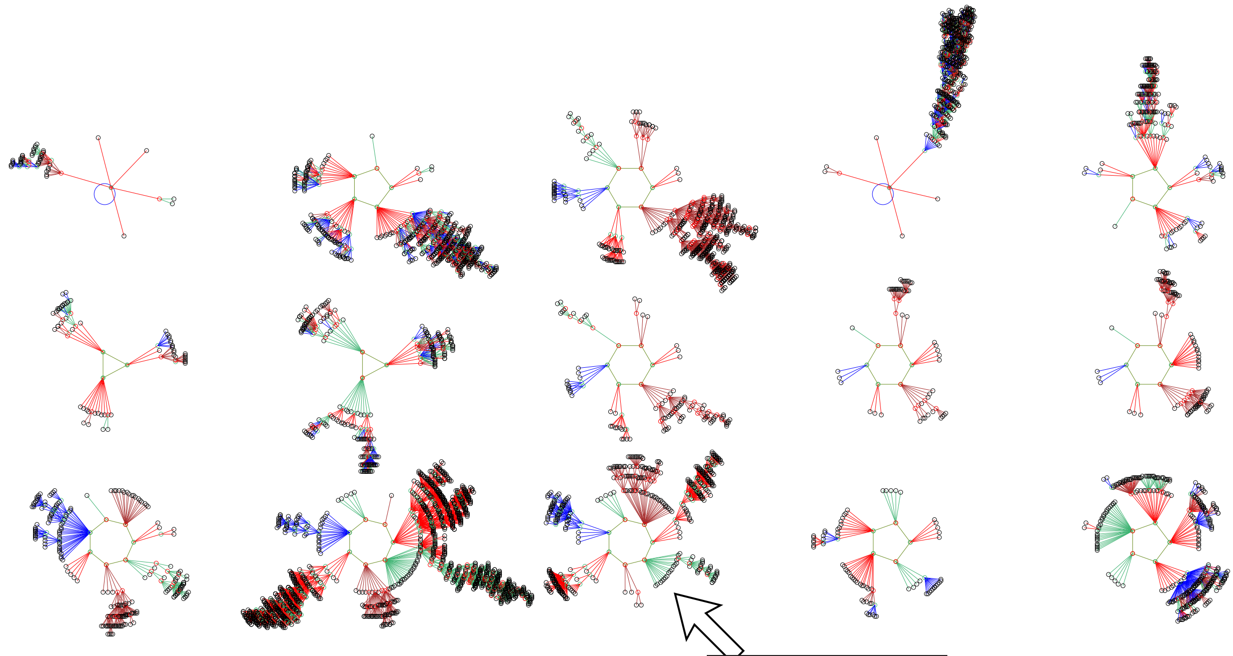
There are many other on-the-fly options, including changing the scale of space-time patterns, changing the seed, rule/s, wiring, and the size of 1d networks (chapter 32).

Concurrently with these standard presentations, space-time patterns can be displayed in a separate window according to the network-graph layout. This can be rearranged in many ways, including various default layouts. For example a 1d space-time pattern can be shown in a circular layout (section 32.15).

2.10.2 Attractor basins

chapters 24 to 30

Options for attractor basins allow the selection of the basin of attraction field, a single basin (from a selected seed), or a sub-tree (also from a seed). Because a random seed is likely to be a garden-of-Eden state, to generate sub-trees an option is offered to run the network forward a given number of



this basin shown in more detail in figure 2.5

Figure 2.4: The basin of attraction field of a random Boolean network, n=13, k=3 (also shown in the jump-graph, figure 20.3). The 2^{13} = 8192 states in state space are organized into 15 basins, with attractor periods ranging between 1 and 7. The number of states in each basin is: 68, 984, 784, 1300, 264, 76, 316, 120, 64, 120, 256, 2724, 604, 84, 428. Figure 2.5 shows the arrowed basin in more detail. Right: the network's architecture, its wiring/rule scheme.

cell	wiring	rule
12	10,1,7	86
11	6,2,9	4
10	10,10,12	196
9	2,10,4	52
8	5,6,8	234
7	12,5,12	100
6	1,9,0	6
5	5,7,5	100
4	4,11,7	6
3	8,12,12	94
2	11,6,12	74
1	6,5,9	214
0	12,9,6	188

steps to a new seed before running backward. This guarantees a sub-tree with at least that number of levels.

Options (and defaults) are provided for the layout of attractor basins, their size, position, spacing, and type of node display (as a spot, in decimal, hex or a 1d or 2d bit pattern, or none). Regular 1d and 2d CA produce attractor basins where sub-trees and basins are equivalent by rotational symmetry. This allows "compression" of basins (by default) into non-equivalent prototypes, though compression can be turned off. Attractor basins are generated for a given system size, or for a range of sizes. As attractor basins are generating, the reverse space-time pattern can be simultaniously displayed.

An attractor basin run can be set to pause to see data on each transient tree, each basin, or each field. Any combination of this data, including the complete list of states in basins and trees, can be saved to a file (chapter 27).

Normally a run will pause before the next "mutant" attractor basin, but this pause may be turned off to create a continuous demo of new attractor basins. A "screen-saver" demo option shows new basins continually growing at random positions (section 4.12 and 24.8).

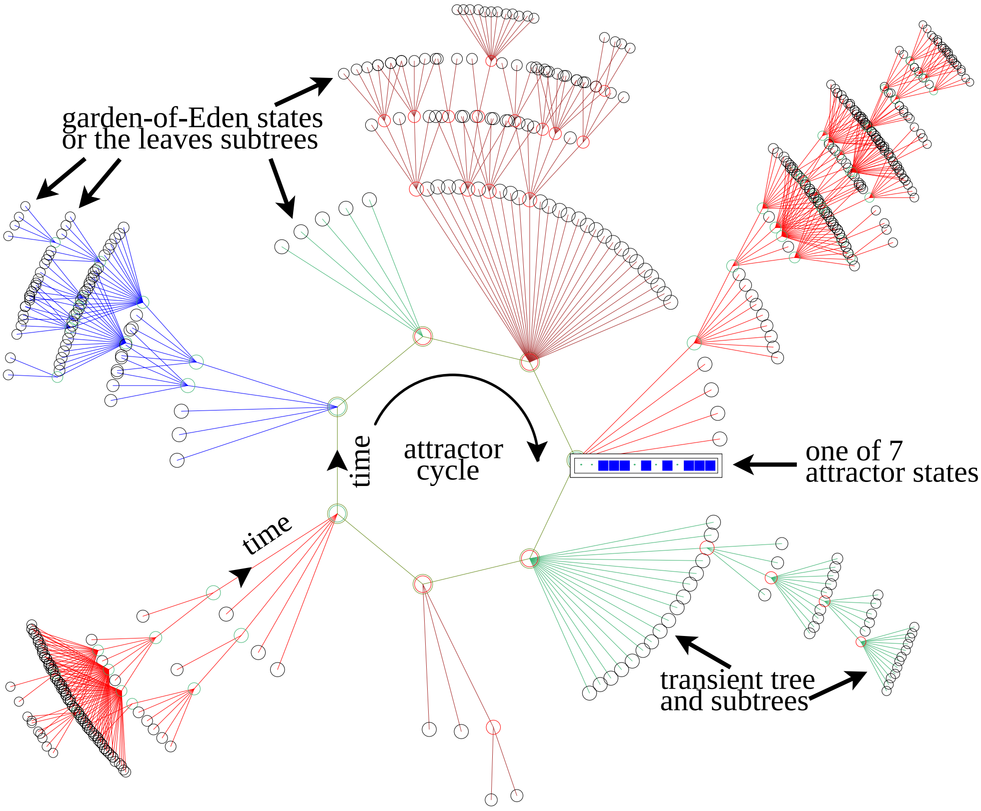


Figure 2.5: A basin of attraction (one of 15) of the random Boolean network (n=13, k=3) shown in figure 2.4. The basin links 604 states, of which 523 are garden-of-Eden states. The attractor period = 7, and one of the attractor states is shown in detail as a bit pattern. The direction of time is inwards from garden-of-Eden states to the attractor, then clock-wise.

2.10.3 Interrupting a run

At any time, a space-time pattern or attractor basin run can be interrupted to pause, save or print the screen image, change various parameters, or backtrack through options (chapters 32 and 30).

2.11 Graphics

chapter 5

In Linux-like versions, the DDLab window starts up at 925×694 or a smaller size automatically set to comfortably fit on the monitor. This can be resized, moved and iconised in the usual way. In DOS the graphics will start up at a resolution of 640×480 (VGA) but can be reset to higher resolutions, or initially with a program parameter. The default background color is black, but can be reset to white either from within the program or with a program parameter. The text size and spacing is set automatically according to the screen resolution, but can be resized.

2.12 Filing and Printing

DDLab allows filing a wide range of filetypes, including network parameters, data, the screen image and vector PostScript files. (sections 19 and 35). For compatability with DOS, filenames follow the DOS format, so a file name has up to eight characters starting with a letter (but not "q") plus a 3 character extension. For example `myfile25.dat`. In DDLab, only the first part of the filename is selected – without the extention (or a default filename can be accepted), the extension is added automatically to identify the file type.

2.12.1 Filing network parameters

Network parameters and states can be saved and loaded for the following: k-mix, wiring-schemes, rules, rule-schemes, wiring/rule schemes, and network states (chapter 19).

2.12.2 Filing data

Data on attractor basins, at various levels of detail (chapter 27) can be automatically saved. A file of "exhaustive pairs", made up of each state and its successor, can be created (section 29.7).

Various data including mean entropy and entropy variance of space-time patterns can be automaticaly generated and saved. This allows a sorted sample of CA rules to be created, discriminating between order, complexity and chaos (section 33). Complex rules, those featuring "gliders" or other large scale emergent structures, can be found automatically. Pre-assembled files of 1d and 2d CA rules sorted by this method[24] are provided with DDLab (section 32.5.1).

2.13 Vector PostScript images

Vector PostScript files can be generated for most DDLab graphics output: space-time patterns or snapshots (1d, 2d and 3d), attractor basins, the wiring graphic, the network-graph, and the attractor jump-graph. This is a new feature in DDLab described in chapter 36. Vector graphics is preferable for publication quality images. The methods work in both Linux-like systems and DOS.

Previous bitmap methods, below, are still available. Most of the figures in this manual where produced as vector PostScript files, others as bitimage PostScript files.

2.14 Bitmap images

The screen image can saved and loaded using an efficient compressed format only applicable within DDLab (section 5.5). Alternativly, in Linux-like systems, a program such as XView can be used to grab the DDLab window or part of it, and to save the image in many standard bitmap formats.

In DOS, to save/print the image in a standard format, use a "stay resident screen grabber". A number of specialist screen grabbers are available, and others are part of "paint" programs. Alternatively run DDLab as a DOS application in Microsoft Windows and use their "paint" screen grabber.

2.15 Printing the screen image

(section 5.6)

Bitmap methods are still available to print the screen image directly from DDLab. In Linux-like systems, the screen can be printed within DDLab as a bitmap PostScript file to a laser printer. Alternativly use XView to grab the bitmap image, in various formats.

In DOS, in theory the screen can be printed within DDLab, but only the following (now well out of date) printers have been tested: Epson MX-82 dot-matrix, Cannon BJC 4000 bubble jet. Printing to other printers is unpredictable.

In both Linux-like systems and DOS, images saved vector PostScript files can be printed in GhostView, or converted to .pdf file and printed in Adobe (Acroread).

2.16 Mutations

A wide variaty of network "mutations", as well as changes in presentation, can be be made, many on-the-fly while the simulation is running.

2.16.1 Running Forward

When running forward, key-press options allow changes to be made to the network and presentation on-the-fly (chapter 32). This includes "mutations" to wiring, rules, current state, and size. A number of 1d "complex" rules (with glider interactions) can be set for k = 5, 6 and 7 (section 32.5.1).

2.16.2 Running Backward

When running backward, and attractor basins are complete, a key press will regenerate the attractor basin of a mutant network. Various mutation options can be pre-set (chapter 28) including random bit-flips in rules and random rewiring of a given number of wires. Sets of states can be specified and highlighted in the attractor basin to see how mutations affect their distribution (chapter 34).

2.17 Quantitative, statistical and analytical measures

Some of the measures and data on network dynamics available in DDLab are listed below. In most cases this information can be displayed graphicaly.

2.17.1 Behaviour parameters

The following static parameters measured on rule look-up tables are available (section 16.19.1).

- The λ parameter (sections 14.1.1 and 16.3.1) and equivalent P parameter.
- The Z-parameter[19, 27].

- The (weighted) average λ and Z for a mixed rule network (section 17.8.2).
- The frequency of canalizing "genes" and inputs (chapter 15 for a rule-mix network, section 18.5 for single rules).
- Post-function data (section 14.12).

λ, Z, and canalization can be set to any arbitrary level,

2.17.2 Network connectivity

The following measures on network connectivity, i.e. the wiring, are available,

- Average k (inputs), number of reciprocal links, and self links (section 17).
- Histograms of the frequency distribution of inputs (i.e. k), outputs, or both (i.e all connections) in the network (section 17.8.13).
- The recursive inputs/outputs to/from a network element, whether direct or indirect, showing the "degrees of separation" between elements (sections 17.5.6 and 17.5.7).
- The network-graph (section 20), where the wiring is analyzed in two ways, as an adjacency matrix or table, and derived from this, as a a graph that can be analyzed in various ways, and rearranged and unraveled, including dragging vertices and defined components to new positions with "elastic band" edges.

2.17.3 Measures on local dynamics

The following measures on local dynamics, i.e. running the system forward from some initial state, its space-time patterns or trajectories, are available,

- A rule-table lookup frequency histogram, which can be toggled between 2d and 3d to include a time dimension (section 31.10).
- The entropy of the lookup frequency over time (section 32.11.2).
- The variance of the entropy, and an entropy/density graph, where complex rules have their own distinctive signatures (section 32.11.4).
- A plot of mean entropy against the variance of the entropy for an arbitrarily large sample of CA rules, which allows ordered, complex and chaotic rules to be classified automatically (chapter 33), also shown as a 2d frequency histogram (section 33.6.2). Ordered, complex and chaotic dynamics are located in different regions allowing a statistical measure of their frequency. In addition the rules can be sorted by entropy variance allowing complex rules to be found automatically.
- The pattern density in a moving window of time-steps (section 31.10).
- The activity/stability of network elements. Frozen islands, the fraction of "genes" that have not changed over the last x generations. For binary networks v=*2* the fraction of frozen 0s and 1s, and "genes" colored according the fraction of time they have been "on" (i.e. 1) in the same window of x time-steps, falling into preset "frequency bins". This is also generalized for multi-value.

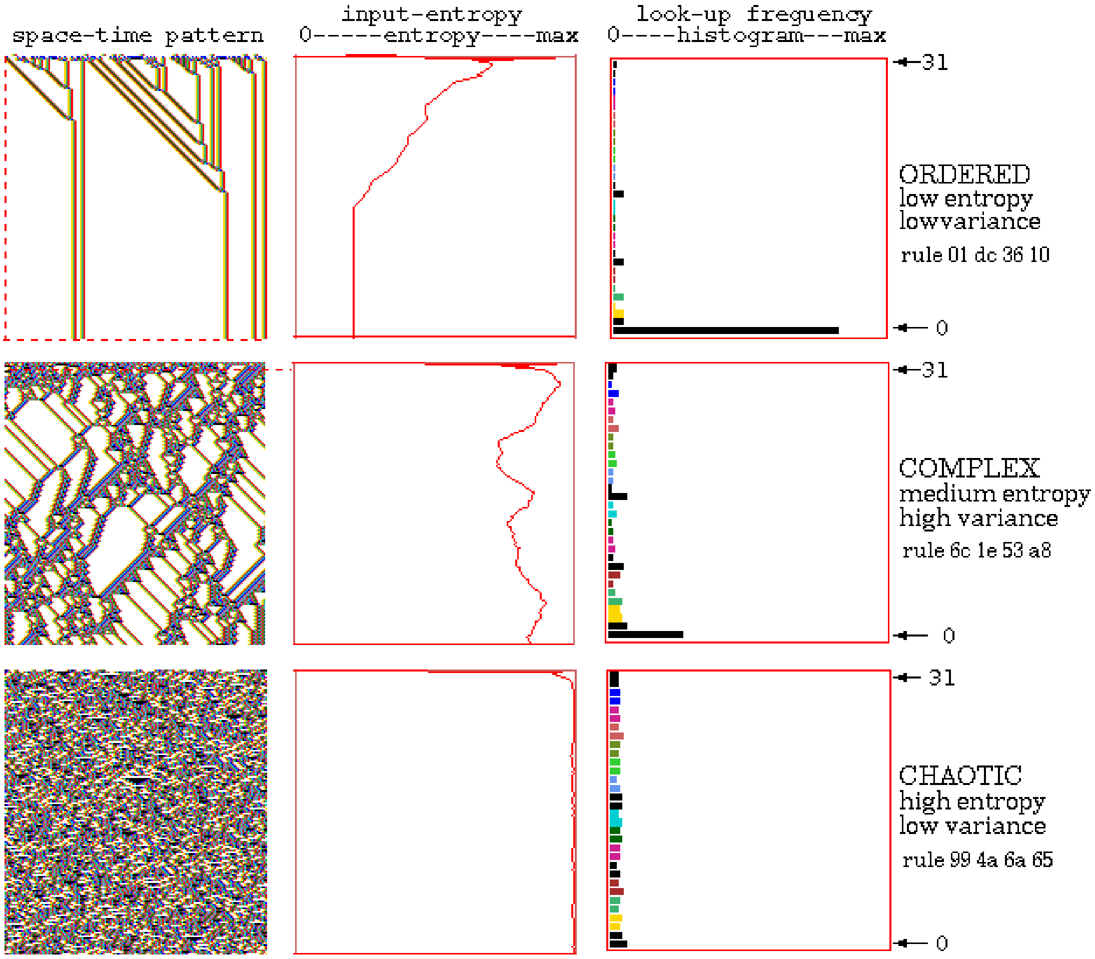
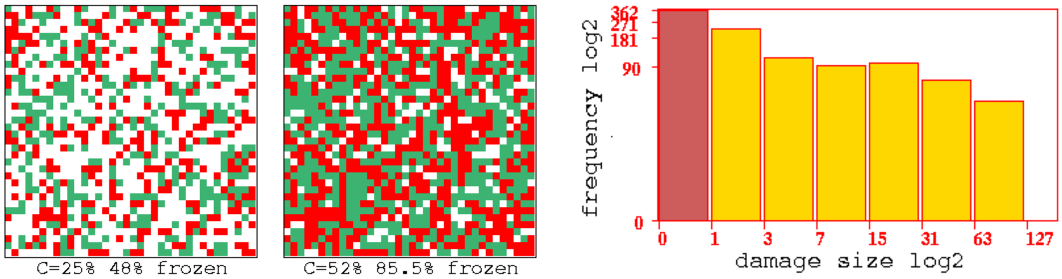


Figure 2.6: Typical 1d CA space-time patterns showing ordered, complex and chaotic dynamics (n=150, k=5,rule numbers shown in hex). Alongside each space-time pattern is a plot of the input-entropy, where only complex dynamics (*centre*) exhibits high variance caused by glider collisions.

- The damage spread, or pattern difference, between two networks in 1d or 2d. A histogram of damage spread can be automaticaly generated for identical networks with initial states differing by 1 bit (section 31.16).
- The "Derrida plot", and Derrida coefficient, analogous to the Liapunov exponent in continuous dynamical systems, measures how pairs of network trajectories diverge/converge in terms of their Hamming distance. This indicates if a random Boolean network is in the ordered or chaotic regime (chapter 22).
- A scatter plot of successive iterations in a 2d phase plane, the "return map", showing a fractal structure, especially for chaotic rules (section 32.11.6).

2.17.4 Measures on global dynamics

The following measures on global dynamics, i.e. attractor basins, are available,



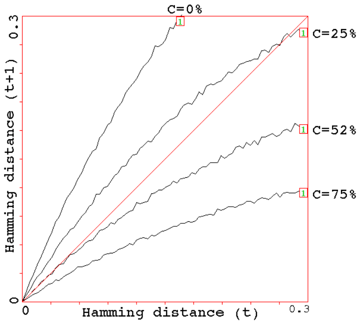


Figure 2.7: Order-chaos measures for a RBN 36×36, $k = 5$. $C =$ the percentage of canalizing inputs in the randomly biased network. *top left*: frozen elements that have stabilized for 20 time-steps are shown, 0s-green, 1s red, otherwise white, for C=25% and 52%. *top right*: the log-log "damage spread" histogram for C=52%, sample size about 1000. *left*: the Derrida plot for C=0%, 25%, 52%, and 75%, for 1 time-step, H_t=0-0.3, interval = 5, sample for each H_t = 25.

- Data on attractor basins. The number of basins in the basin of attraction field, their size, attractor period and branching structure of transient trees. Details of states belonging to different basins, subtrees, their distance from attractors or the subtree root, and their in-degree (chapter 27).

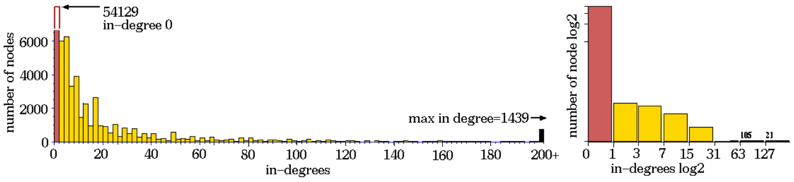
- A histogram of attractors (or skeletons) showing the frequency of arriving at different attractors from random initial states. This provides statistical data on the basin of attraction field for large networks. The number of basins, their relative size, period, and the average run-in length can be measures statistically (section 31.18). An analogous method shows the frequency of arriving at different "skeletons", partly frozen patterns.

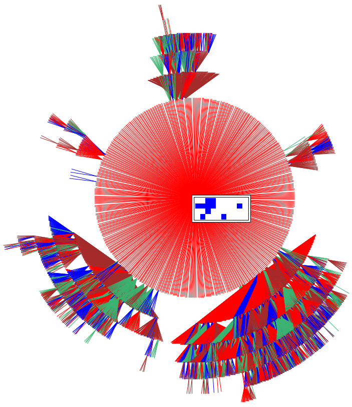
- Garden-of-Eden density plotted against the λ and Z parameters (section 24.9), and aginst network size (section 24.11).

- A histogram of the in-degree frequency of attractor basins or subtrees (section 24.6).

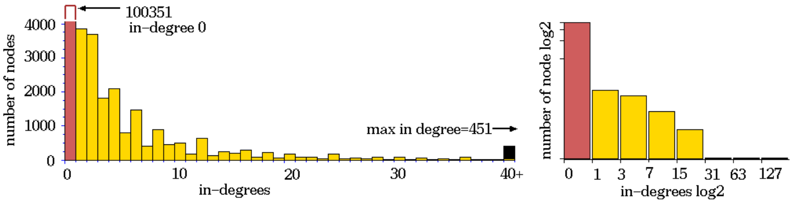
- The state-space matrix, a scatter-plot of the left half against the right half of each state bit string, using colour to identify different basins, or attractor cycle states (section 24.5).

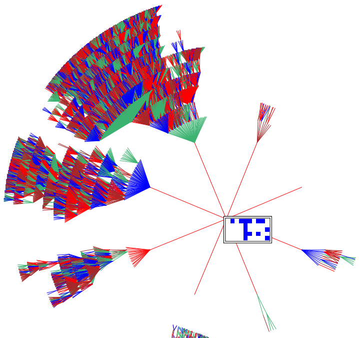
- The attractor jump-graph, an analysis of the basin of attraction field tracking where all possible 1-bit (or 1-value) flips to attractor states end up, whether to the same, or to which other, basin (section 20). The information is presented in two ways, as a jump-table: a matrix showing the jump probabilities between basins, and as a jump-graph: a graph with weighed vertices and edges giving a graphic representation of the jump-table. The jump-graph itself can be analyzed and manipulated in various ways, and rearranged and unraveled, including dragging vertices and defined components to new positions with "elastic band" edges.



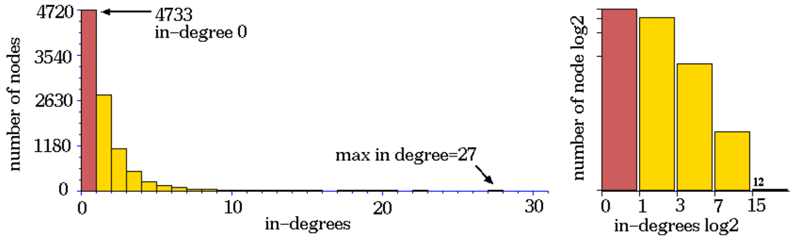


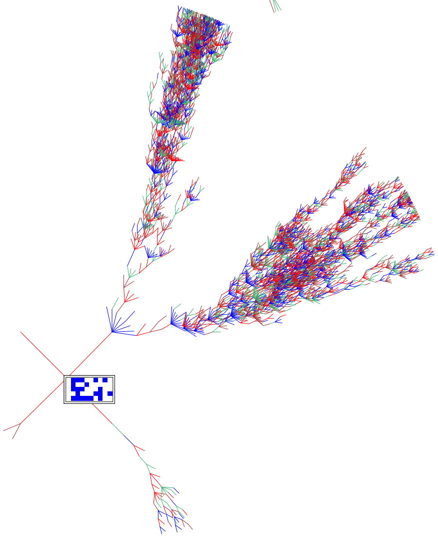
Ordered dynamics. Rule 01dc3610, n=40, Z=0.5625, λ_{ratio}=0.668. *right:* The complete sub-tree 7 levels deep, with 58153 nodes, G-density=0.931.





Complex dynamics. Rule 6c1e53a8, n=50, Z=0.727, λ_{ratio}=0.938. *right:* The sub-tree, stopped after 12 levels, with 144876 nodes, G-density=0.692.





Chaotic dynamics. Rule 994a6a65, n=50, Z=0.938, λ_{ratio}=0.938. *right:* The sub-tree, stopped after about 75 levels, with 9446 nodes, G-density=0.487.

Figure 2.8: Subtrees of ordered-complex-chaotic CA. The space-time patterns of the rules are shown in figure 2.6. The in-degree histogram of a typical sub-tree showing that convergence (bushiness of subtrees) is: ordered-high, complex-medium, chaotic-low.

2.18 Reverse algorithms

There are three different reverse algorithms for generating the pre-images of a network state.

- An algorithm for local 1d wiring[19] – 1d CA but rules can be heterogenious.
- A general algorithm[20] for RBN, DDN, 2d or 3d CA, which also works for the above.
- An exaustive algorithm that works for any "random mapping" inluding the two cases above.

The first two reverse algorithms (section 29.6) generate the pre-images of a state directly; the speed of computation decreases with both neighborhood size k, and network size. The speed of the third exhaustive algorithm (section 29.7) is largely independent of k, but is especially sensitive to network size.

The method used to generate pre-images will be chosen automatically, but can be overridden. For example, a regular 1d CA can be made to use either of the two other algorithms for benchmark purposes and for a reality check that all methods agree. The time taken to generate attractor basins is displayed in DDLab. For the basin of attraction field a progress bar indicates the proportion of states in state-space used up so far.

2.18.1 1d CA wiring

The CA reverse algorithm applies specifically for networks with 1d CA wiring (local wiring) and homogeneous k, such as 1d CA, though the rules may be heterogeneous. This is the most efficient thus fastest algorithm, described in [19, 27]. Furthermore, compression of 1d CA attractor basins by rotation symmetry (section 26.1) speeds up the process.

2.18.2 Non-local wiring

Any other network architecture, with non-local wiring, will be handled by a slower *general* reverse algorithm described in [20, 27]. A histogram revealing the inner workings of this algorithm can be displayed. Regular 2d or 3d CA will also use this general reverse algorithm though in principle more efficient algorithms that take advantage of 2d or 3d local wiring could be devised. However, compression algorithms will come into play in 2d to take advantage of the many rotation symmetries on the torus[1].

2.18.3 Exhaustive

A third, brute force, reverse algorithm first sets up a mapping, a list of "exhaustive pairs" of each state in state-space and its successor (this can be saved). The pre-images of states are generated by reference to this list. The exhaustive testing method is restricted to small systems because the size of the mapping increases exponentially as 2^n, and scanning the list for pre-images is slow compared to the direct reverse algorithms for CA and RBN. However, the method is not sensitive to increasing neighborhood size k, and is useful for small networks with large k. Exhaustive testing is also used for sequential updating ((section 29.9). The basin of attraction field of the 4x4 "game-of-Life", where k=9, generated by this method took about 30 minutes on my 66MHz 486).

2.19 Random map

The random mapping routine (section 29.8) creates a list of "exhaustive pairs", assigning a successor state at random to each state in state space, possibly with some bias (rules and wiring previously set are ignored). The attractor basins are reconstructed by reference to this random map with the exhaustive testing algorithm. The space of random maps for a given system size corresponds to the

[1] Compression does not apply for a regular 2d CA where k=6 or k=7, which has a hex grid, or for 3d CA, as the algorithms to take account of these symmetries have not been resolved.

space of all possible basin of attraction fields and is the super-set of all other deterministic discrete dynamical systems.

2.20 Sequential updating

(section 29.9)

By default, network updating is synchronous, in parallel. DDLab also allows sequential updating, both for space-time patterns (sections 32.3) and attractor basins (section 29.9.5). Default orders are forwards, backwards or a ramdom order, but any specific order can be set out of the $n!$ possible orders for a network of size n. The order can be saved/loaded from a file.

2.20.1 Neutral order components

(section 29.10)

An algorithm in DDLab computes the neutral order components (limited to network size $n \leq 12$). These are sets of sequential orders with identical dynamics. DDlab treats these components as subtrees generated from a root order, and can generate a single component subtree, or the entire set of components subtrees making up sequence space (the neutral field) which are drawn in an analogous way to attractor basins.

2.21 Sculpting attractor basins

(section 34)

Learning and forgetting algorithms allow attaching and detaching sets of states as predecessors of a given state by automatically mutating rules or wiring couplings. This allows "sculpting" the attractor basin to approach a desired scheme of hierarchical categorization. Because any such change, especially in a small network, usually has significant side effects, the methods are not good at designing categories from scratch, but might be useful for fine tuning a network which is already close to where its supposed to be.

When an attractor basin is complete, within the learning routine, a "target" state, together with a number of "aspiring pre-images" (predecessors) can, be selected. These states may be just highlighted in successive mutant attractor basins, or the learning/forgetting algorithms will attempt to attach/detach the aspiring pre-images to/from the target state, and can be set for either rule-table bit-flips or wire moves. In fact the bit-flip method cannot fail. New attractors can be created and sub-trees transplanted. The result of learning/forgetting, including side effects, will be apparent in the new attractor basins. The algorithms, and their implications are described in[20].

More generally, a very preliminary method for reverse engineering a network, also known as the inverse problem, is included in DDLab, by reducing the connections in a fully connected network to satisfy an exhaustive map (for network sizes $n \leq 13$, section 18.10). The inverse problem is finding a minimal network that will satisfy a full or partial mapping (i.e. fragments of attractor basins such as trajectories).

Chapter 3

Accessing DDLab

This chapter gives detailed instructions for downloading, unzipping, unpacking, and running DDLab. Information is also given about the DDLab web site, manual, license, copyright and disclaimer. The Unix, Linux, Irix, Mac and Cygwin versions will be refered to as "Linux-like", as opposed to the DOS version.

3.1 The DDLab web site

For the latest version check the DDLab web site located at one of the following,

www.ddlab.org
www.cogs.susx.ac.uk/users/andywu/ddlab.html

3.2 Latest version of DDLab

To download the latest precompiled versions of DDLab, click on the ftp link for the latest version, then choose the required platform from the list below (at time of writing).

Jan 2008 DDLab version m06, sub-directory ddm_2008_jan.

ddlab_linux_m06.tar.gz:	...	Linux/PC: compiled in Ubuntu 6.06.
ddlab_cygwin_m06.tar.gz:	...	Cygwin/Windows/PC: compiled with Cywin/X, a Linux envoronment within Windows.
ddlab_unix_m06.tar.gz:	...	Unix/XWindows/Sun: compiled in SunOS 5.8.
ddlab_irix_m06.tar.gz:	...	Irix/SGI: compiled for 6.5.27 with MIPSprp C 7.4.3 and -n32.
ddlab_mac_m06.tar.gz:	...	MAC OSX with X11: compiled in Mac OSX 10.4.10.
ddlab_dos_m06.tar.gz:	...	DOS/PC.compiled with Watcom C version 11.

Also download dd_extra.tar.gz for full functionality.

3.3 Older versions of DDLab

Older versions of DDLab (still available) are listed below. See also links to “versions” and “new features” relating to each version.

Nov 2005 DDLab version m05, sub-directory ddm_2005_nov.

ddlab_linux_m05.tar.gz:	...	Linux/PC: compiled with SuSE 9.0.
ddlab_cygwin_m05.tar.gz:	...	Cygwin/Windows/PC: compiled with Cywin/X, a Linux envoronment within Windows.
ddlab_unix_m05.tar.gz:	...	Unix/XWindows/Sun: compiled in SunOS 5.8.
ddlab_irix_m05.tar.gz:	...	Irix/SGI: compiled for 6.5.27 with MIPSprp C 7.4.3 and -n32.
ddlab_mac_m05.tar.gz:	...	Mac: compiled in Mac OSX 10.3. with X11.
ddlab_dos_m05.tar.gz:	...	DOS/PC.compiled with Watcom C version 11.

May 2002 DDLab version 24, sub-directory dd_2001_july.

ddlab_dos_24.tar.gz:	...	DOS/PC.
ddlab_unix_24.tar.gz:	...	Unix/XWindows/Sun, compiled for Solaris 2.5.
ddlab_linux_24.tar.gz:	...	Linux/PC, compiled for Red Hat 7.1.
ddlab_irix_24.tar.gz:	...	Irix 6.5 for SGI.

Feb 1999 DDLab version 22, sub-directory dd_1999_feb.

ddlab_22.zip:	...	DOS/PC.
ddlab_unx_22.tar.gz:	...	Unix/XWindows/Sun, compiled for Solaris 2.5.
ddlab_lnx_22.tar.gz:	...	Linux/PC, compiled for Red Hat 5.1.

Sept 1997 DDLab version 21, sub-directory dd_1997_sept.

ddlab_21.zip:	...	DOS/PC.
ddlabx_21.tar.gz:	...	Unix/XWindows/Sun, compiled for Solaris 2.5.
ddlabx_linux_21.tar.gz:	...	Linux/PC, compiled for Linux 1.99 ELF.

March 1996 version version of DDLab, sub-directory dd_1996_march.

ddlab_20.zip:	...	DOS/PC.
ddlabx_20.tar.gz:	...	Unix/XWindows/Sun, compiled for OS 4.1.

July 1995 version of DDLab, sub-directory dd_1995_jul.

ddlab_10.zip:	...	DOS/PC.

3.4 The DDLab manual

The manual you are reading relates to the latest version of DDLab, released in April 2009. The software and manual will be updated from time to time. Check the DDlab web site for update news. To download the DDLab manual click the ftp link. The manual is available in the sub-directory dd_manual, in the following formats,

relating to DDLab version m06, Jan 2008

m_ddman07.ps.gz	...	gzip'ed PostScript file, read with GhostView.
m_ddman07.pdf	...	a pdf file, read with Acroread.

3.4.1 Previous DDLab manuals

Previous editions of the DDLab manual relating to earlier releases of DDLab are as follows:

relating to DDLab version 24, July 2001

ddop2001.ps.gz	...	gzip'ed PostScript file, read with GhostView.
ddop2001.pdf	...	a pdf file, read with Acroread.

relating to DDLab version 20, March 1996

ddman_20.zip	...	will unzip (with PKUNZIP) to ddman.doc, a Word for Windows file. This early manual is not illustrated.

3.5 Unzipping and running - DOS

The latest DOS wersion of DDLab, ddlab_dos_m06.tar.gz, can be unzipped with Winzip in earlier versions of Windows. For Windows Vista, open source "7zip" works to uncompress.

The DOS version may have some drawbacks; for better results install Cygwin (a Linux environment inside Windows) and use the Cygwin wersion of DDLab.

The DOS version will unzip to give a number of files, including,

ddlabm06.exe ... the program *(for example)*.
dos4gw.exe ... the DOS extender, access to extended memory.

Keep all the files together in their own directory.

For best results DDLab should be run in pure DOS (available in Windows98 and before), otherwise, prior to Vista, DDLab can be run from a DOS or "command line" window (some precautions will apply – see sections 5.2.1 - 5.2.2). In Vista the DOS version does not work from the command line, but it works nicely (but slowly) in "DOSBox", an open source MS-DOS emulator, intended for old PC games.

In pure DOS you can also add the following program parameters for a different graphics setup (this can also be changed later).

-w ... for a white background.
-m ... for 800x600 resolution.
-h ... for 1024x768 resolution.

For example, for a white background and 1024x768 enter `ddlabm06 -w -h`.

3.6 Unzipping and running - Linux-like versions

The Linux-like versions refer to Unix, Linux, Irix, Mac, and Cygwin, as opposed to the DOS version. Place the `.tar.gz` file in its own directory, called say, `ddlab`. To unzip the `.gz` file, then unpack the `.tar` file, follow the example below,

```
gunzip ddlab_linux_m06.tar.gz     ...  to unzip the .gz file
tar -xvf ddlab_linux_m06.tar      ...  to unpack the .tar file
```

This will give one or more files, including the executable, `ddlabm06`. Keep these files in the same `ddlab` directory. To run the program enter `ddlabm06 &` (or `./ddlabm06 &` if there is no dot in your path). The `&` allows you to retain control of the xterm window, where messages and data might be shown.

You can also place and unpack the the `ddextra.tar.gz` file in the `ddlab` directory. The way I prefer to run DDLab is to create a sub-directory to `ddlab`, called say `ddfiles`, and place the files in `ddextra.tar.gz` in this. From `ddfiles` enter `../ddlabm06 &` to run. This ensures that files created within DDLab stay within `ddfiles`, and do not clutter up the in `ddlab` directory, were I also have all my source files.

The default background is black. This can be changed with a program parameter, `-w` for a white background.

3.6.1 Unix library files

DDLab for Unix is compiled with “static” set, so that missing library problems should not occur. However, the libraries `libx11` and `libsunmath` do need to be in your system, they usually are. If they are missing, they can be downloaded from the file,

```
unix_libs.tar.gz
```

which will unzip and unpack as described above to give the following files,

```
libX11.so.6.1
libsunmath.so.1
```

These files should be installed in the same directory as DDLab.

3.7 Extra data files

The files in `dd_extra.tar.gz`, common to all platforms, contain data used by DDLab. The files should be in the same directory as the DDLab executable (though DDLab will run without the files), or as related in section 3.6, the files can be in a sub-directory and DDLab run from that sub-directory by prefixing “../” before the executable.

The lists below show some, but possibly not all, the files.

3.7.1 Complex rule collections

These are collections of complex rules for various combinations of v, k and lattice dimensions which can be loaded on-the-fly when running space-time patterns. Enter 'g' for a random rule, or the next rule in the sequence (depending on the setup), see section ??.

1d complex rule collections, based on the full look-up table, rcode
g_v2k5.r_s, g_v2k6.r_s, g_v2k7.r_s … $v = 2$ complex rules, as in binary DDLab.
g_v3k3.r_s, g_v4k2.r_s, g_v4k3.r_s, g_v5k2.r_s … $v > 2$ complex rules.

2d complex rule collections, hexahonal lattice, based on the k-totalistic rules, kcode
g_v3k6.r_v
, g_v3k7.r_v

3.7.2 Selected 2d complex rules, $v = 3$

Selected $v = 3$ complex 2d rules based on the k-totalistic lookup table. These rules can be loaded individually while running space-time patterns. For $k = 6$ and $k = 7$ the lattice is hexagonal, otherwize the lattice is square. Some $k6$ and $k7$ rules also give interesting dynamics in 3d.

v3k4x1.vco … see 4-way glider gun seed below.
v3k5x1.vco … gliders bounce off static structures.
v3k6x1.vco … the Beehive rule[33], hexagonal lattice.
v3k6x2.vco … spirals overcome gliders, hexagonal lattice.
v3k6B1.vco … burning paper or preditor-prey..
v3k6n6.vco … slow moving gliders, see 2-way glider-gun seed below.
v3k7w1.vco … the spiral rule[34], hexagonal lattice
v3k7x1.vco … gliders collisions make static structures.
v3k8x1.vco … gliders move orthogonally and diagonally.
v3k9x1.vco … gliders move orthogonally and diagonally.

3.7.3 Selected seeds

Selected 1d, 2d and 3d seeds (initial states) which can be loaded while runing space-time patterns for interesting results for various rules.

pento.eed … r-pentomino seed for the game-of-Life
v3k4gun.eed … seed for 4-way glider-gun in v3kn6.vco
v3k6n64.eed … initial state for 2-way glider-gun in v3kn6.vco
seed for the 2d and 3d beehive-rule
Bcgun.eed … seed for 6-way glider-gun.
Bpuff.eed … seed for the amazing puffer-train.
B3d_gg.eed … seed for a 3d 4-way glider-gun.

3.7.4 Sorted rule samples

Random rule samples sorted by input-entropy standard deviation, and max-minmax-entropy (see section ??), showing how rule-space is distributed between order, complexity and chaos[27]. The samples can be loaded and viewed in various ways while runing space-time patterns.

1d complex rule samples, based on the full look-up table

`v2k5ss.sta v2k6ss.sta v2k7ss.sta` ... $v = 2$, standard deviation

`v3k3ss.sta v4k2ss.sta v4k3ss.sta v5k2ss.sta`... $v > 2$, standard deviation

2d complex rule samples, based on k-totalistic rules

`v3k4bs.sta v3k5bs.sta v3k6bs.sta v4k4bs.sta v4k6bs.sta` ... standard deviation

`v3k7bs.sta` ... max-minmax-entropy

3.7.5 Byl's self reproducing loop

J.Byl's self reproducing loop[1], is a $v6k5$ 2d CA, a simplification of Langton's loop.

`v6k5_byl.rul` ... Byl's self reproducing loop rule (full look-up table).

`v6k5_byl.eed` ... as seed to initiate Byl's loop.

3.8 The Quick Start Examples

Chapter 4 gives brief "quick start" examples for a number of typical routines. Its a good idea to try these first get the flavour of DDLab before reading the detailed manual.

3.9 License and Copyright

3.9.1 License

DDLab is free shareware for personal users. Other users, including commercial users, companies, government agencies, research or educational institutions are required to register.
See `www.ddlab.com/ddinc.html` for further details.

The DDlab code is open source under the GNU General Public License (GPL) as published by the Free Software Foundation. Compiled versions of DDLab, and the source code, is available from the DDLab home page `www.ddlab.com`.

3.9.2 Copyright

DDLab is copyright (c) 1993-2007, Andrew Wuensche (`andy@ddlab.com`).

3.9.3 Disclaimer

No warranty is made for DDLab or its performance, and no responsibility or liability whatsoever is accepted to anyone for the consequences of using it.

[1] J. Byl. "Self-Reproduction in small cellular automata." Physica D, Vol. 34, pages 295-299, 1989.

Chapter 4

Quick Start Examples

This chapter briefly describes the DDLab graphical user interface, and gives a number of examples of DDLab functions and routines. Try these examples first, to get the flavor of DDLab, before reading the detailed program reference – chapter e5 onwards.

4.1 The DDLab screen

In Linux-like platforms, or Windows, DDLab occupies a window within the monitor screen, (in pure DOS the whole screen is occupied). For simplicity we will refer to the DDLab "screen", and various panels which appear from within the screen as "windows" or "prompts". Their location within the screen is usually indicated, for example "top-right", "top-left", "bottom-left", etc.

To run DDLab, for DOS enter `ddlabm06` at the DOS or command prompt. For Linux-like systems enter `ddlabm06 &` (or `./ddlabm06 &`) – always add a final & to retain control of the teminal window, where various data may be displayed.

When DDLab is run with no program parameters, the screen appears with a black background. The program parameter `-w` gives a white background, i.e. `./ddlabm06 -w &` for Linux, `ddlabm06 -w` for DOS. Descriptions of colors in this manual assume a white background.

If an **UNLICENSED** banner is displayed, enter **return** to continue[1]. A title bar is displayed across the bottom of the screen. A series of prompts are presented to set up the network, functions to be performed, and presentation. These prompts appear either in a main sequence for the most common settings, or in various windows that automatically open up.

The mouse cursor (if detected) is used to set bits or values in rule-tables and network states, for "drawing" patterns, especially in 2d networks, for dragging nodes in the network-graph and jump-graph, and for some other functions, but most user inputs are from the keyboard.

4.1.1 User Input

The flashing cursor (usually green) prompts for input. Enter appropriate input from the keyboard. To revise the input, press **q**, **backspace**, or the right mouse button. To accept the input, and move on to the next prompt or routine, press **return** or the left mouse button. If no input was entered, or if the input was inappropriate, a default input is automatically selected.

[1] Another program parameter turns off the **UNLICENSED** banner for licensed users.

4.1.2 Backtrack

To backtrack to the preceding prompt, to revise, or interrupt a running routine such as space-time patterns or attractor basins, press **q**, or right mouse button.

You can backtrack to any stage in the prompt sequence with **q** (or right mouse button), eventually to exit the program.

4.1.3 Quitting DDLab

To quit DDLab immediately (except in DOS) enter **Ctlr-q** at any prompt, followed by **q**. Otherwise backtrack with **q** to the start of the program, then enter **q** to exit.

4.1.4 Skipping Forward

At some points in the prompt sequence, its possible to skip forward, to avoid a succession of prompts for special settings. When the following top center banner is visible,

accept defaults-d ... enter **d** to skip forward.

4.1.5 The graphics setup - Linux-like systems

Linux-like systems include Unix, Linux, Irix, Mac-X11 and Cygwin. The screen will start with a black background if no program parameters were set, and with a resolution automatically set to comfortably fit the monitor, but with an initial maximum of 925×694 pixels.

To change the graphics setup after DDLab has started, at the first prompt select **g** for graphics. A graphics setup screen will appear. Enter **b** to toggle the background between black and white. Other options allow changing the resolution, font size, text line spacing and cursor flash speed. The screen can be resized by dragging the corners (or edges) with the mouse in the usual way.

4.1.6 The graphics setup - DOS, or Windows command line

Although the DOS version of DDLab is still available, we recomend that you install Cygwin (a Linux environment inside Windows) and run the Cygwin version of DDLab.

If you are still running DDLab in DOS, or in a Windows command line, read on ...
Its recomended that you run DDLab in pure DOS mode, which is available in Windows98 and prior versions, where there is an option “Restart the computer in MS-DOS mode”. otherwise run DDLab in a DOS or command-line window where it should be run initially in low resolution only (the default). If you expand to full screen (toggle **Alt-Enter**) you may use higher resolutions, but the mouse cursor will probably not be visible. Be sure to revert to low resolution before changing from full screen back to the DOS/command-line window to avoid unforseen consequences.

In DOS, the background can be changed between black and white, and the resolution between 640×480, 800×600, and 1024×768 pixels, given the necessary monitor, graphics card and driver.

The DDLab screen will start up at a resolution of 640×480 (VGA) with a black background, if no program parameters are set. Program parameters may be added for SVGA as follows: `-h` (for high, 1024×768) and `-m` (for medium, 800×600), i.e. for high resolution start DDLab by entering `ddlabm05 -h` at the DOS prompt. For a white background enter `ddlabm05 -w`, or `ddlabm05 -h -w` for both high resolution and a white background.

To change the resolution or background after DDLab has started, at the first prompt select **g** for graphics. A graphics setup screen will appear. Enter **b** to toggle the background. Other options allow changing the resolution, font size and text line spacing and cursor flash speed.

4.2 Basin of attraction fields

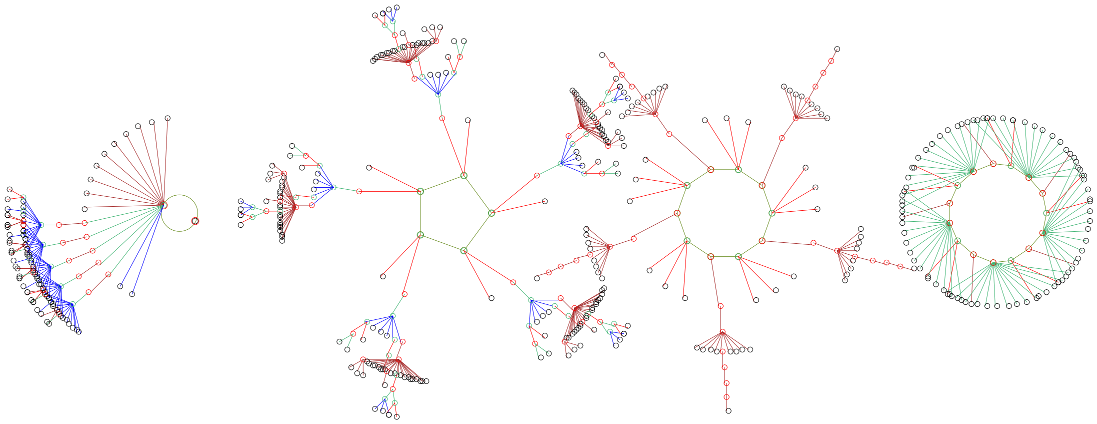


Figure 4.1: A basin of attraction field of a binary 1d Cellular Automaton, $v2k3$, n=10, decimal rule 9, with copies of equivalent basins suppressed.

To generate a basin of attraction field similar to figure 4.1 above, do the following:

1. From the first prompt keep accepting defaults with **return** or left mouse button (about 14 presses), until the top center **output parameters** banner appears, and a top-right window with a list of options starting with **accept all basin defaults -d**. Enter **d** to skip these special options.
2. A final top-right prompt window appears, just before drawing basins. Enter **return**.
3. The basin of attraction field will be generated. Copies of equivalent basins are suppressed. The initial default setup is for a 1d CA, network size n=10, value-range v=2, and neighborhood size k=3. If these parameters were changed they become the new defaults. The rule (chosen at random by default) appears in a window at bottom of the screen. A top-right window shows brief data on the field once it is generated. A progress bar below this window shows the proportion of state-space as it is used up. Vertical lines on this bar indicate the states used to seed the basins.
4. A prompt window appears top-left. Enter **return**.
5. Another basin of attraction field is generated, a one-bit "mutant" of the previous rule, with corresponding data. This process can continue indefinitely (it can also be set on automatic).
6. Enter **q** to interrupt and backtrack up the prompt sequence.

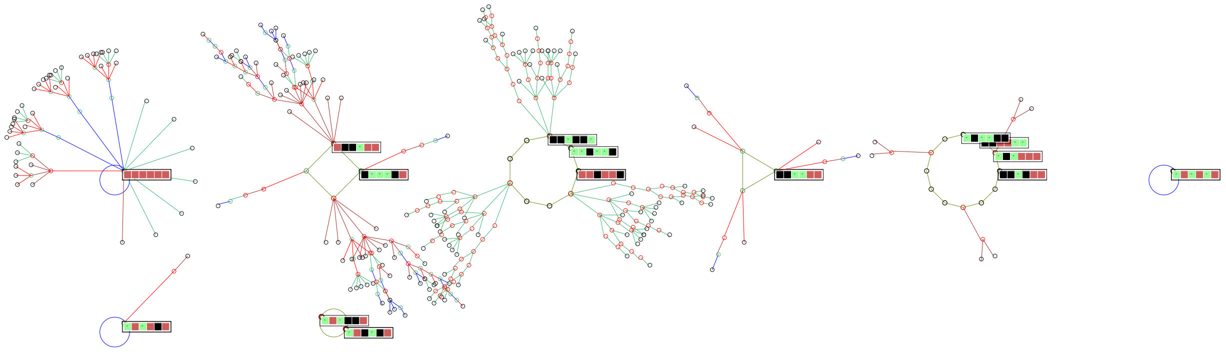


Figure 4.2: The basin of attraction field of multi-value $v3k3$ n=6 1d CA. The lookup table is 120201201020211201022121111 (1886122584a655 in hex). Just the 8 nonequivalent basins are shown from a total of 23, and attractor non-equivalent states are shown as 2d patterns. State-space= $v^n = 3^6 = 729$. Note that the overlap can be fixed with the layout options.

4.2.1 Changing basin parameters

Try the previous routine again, changing v, n or k. The scale, position, node display etc. can be fine-tuned with the special "**output parameters**" options that were skipped before. For example, to create figure 4.2 below, enter the following: (otherwise enter **d** to accept defaults).

1. To revise v backtrack to the the prompt **Value range** ... **:** enter 3.
2. To revise n, at the prompt **Network size** ... **:** enter 6.
3. To revise k, at the prompt **Neighborhood size** ...**:** enter 3.
4. At the prompt **Select v3k3 rule** ... **:** enter **h** for the rule in hex, then enter 1886122584a655, then **return** to accept (for a random rule just enter **return** or **r**).
5. At the **output parameters** prompt, enter **p** to jump directly to the "display" options.
6. Enter **return** until the prompt **highlight attractor** ... **:**, then enter **a** for "all", then **d** twice to accept further defaults and start drawing basins.
7. Enter **return** for a mutant basin, or **q** to backtrack.

Note that increasing v will reduce the maximum allowable n and k (see section ??), and as these values increase basins will take longer to generate.

4.3 "Backwards" space-time patterns, and state-space matrix

While the attractor basins are generating, various display settings can be changed on-the-fly. These are indicated on the right of the bottom title bar. If the basins generate too fast to intervene on-the-fly, enter **s** for **speed** in the top-left window, and follow self-explanatory prompts to slow down. Alternnatively, backtrack to slightly increase n, v or k, as in the previous example.

1. Enter **s** to toggle the "backwards" space-time pattern on-off, and see predecessors (pre-images) being generated on the left of the screen (figure 4.3). Initially the attractor states will be displayed, then each state and its set of pre-images. Expand or contract the scale of the backwards space-time pattern with **e** and **c**. Toggle scrolling on/off with #.

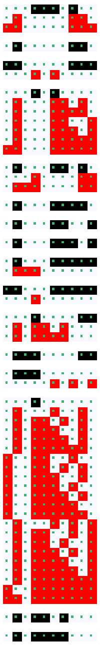


Figure 4.3: Backwards space-time patterns relating to the basin of attraction field of the Cellular Automaton in figure 4.1. Space across, time top down. Foe $v = 2$, The red and white bit patterns are the predecessors of black and white bit patterns.

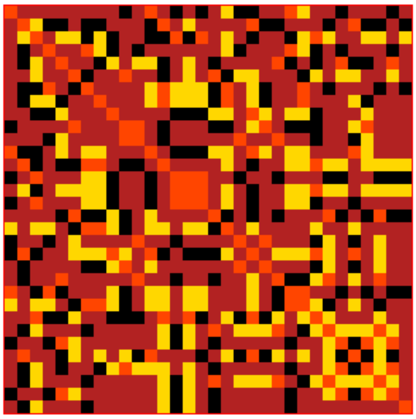


Figure 4.4: The state-apace matrix represents state-space, plotting the left half of each state bitstring against the right half. Colors represent different basins of attraction in figure 4.1.

2. Enter **m** to toggle the display of the state-space matrix in the lower right corner (figure 4.4. This reveals interesting symmetries. Different colors represent states in different basins.
3. Enter < to incrementally slow down, or > for restore maximun speed.
4. Enter **q** to interrupt and backtrack up the prompt sequence.

4.4 Basin of attraction fields for a range of network sizes

To produce output similar to pages in the "Atlas of Basin of Attraction Fields", Appendix 2 of the book "The global ynamics of Cellular Automata"[19], proceed as follows:

1. Backtrack with **q** (or right mouse button) to the start of the program.
2. At the third prompt, **range of network size-r:** enter **r**.
3. Enter **return** until the top center **output parameters** banner appears, then **a** to restore all defaults, then **d** to skip further special options.
4. Enter **return** to start a *range* of CA basin of attraction fields (with the same rule), for increasing sizes from 5 to 12.
5. Enter **return** for the next "mutant" CA rule.
6. Toggle the display of the "backwards" space-time patterns with **s** and the state-space matrix with **m** as described in 4.3.
7. To slow down the the generation of basins (probably required to intervene on-the-fly) enter **s** for **speed** in the top-left window, and follow self-explanatory prompppts.
8. Enter **q** to interrupt and backtrack up the prompt sequence.

You may need to readjust the size of basins for everything to fit. To do this, backtrack to **output parameters** , enter **l** for layout, then adust the size and spacing of basins with various self-explanatory prompts (described fully in chapter 25).

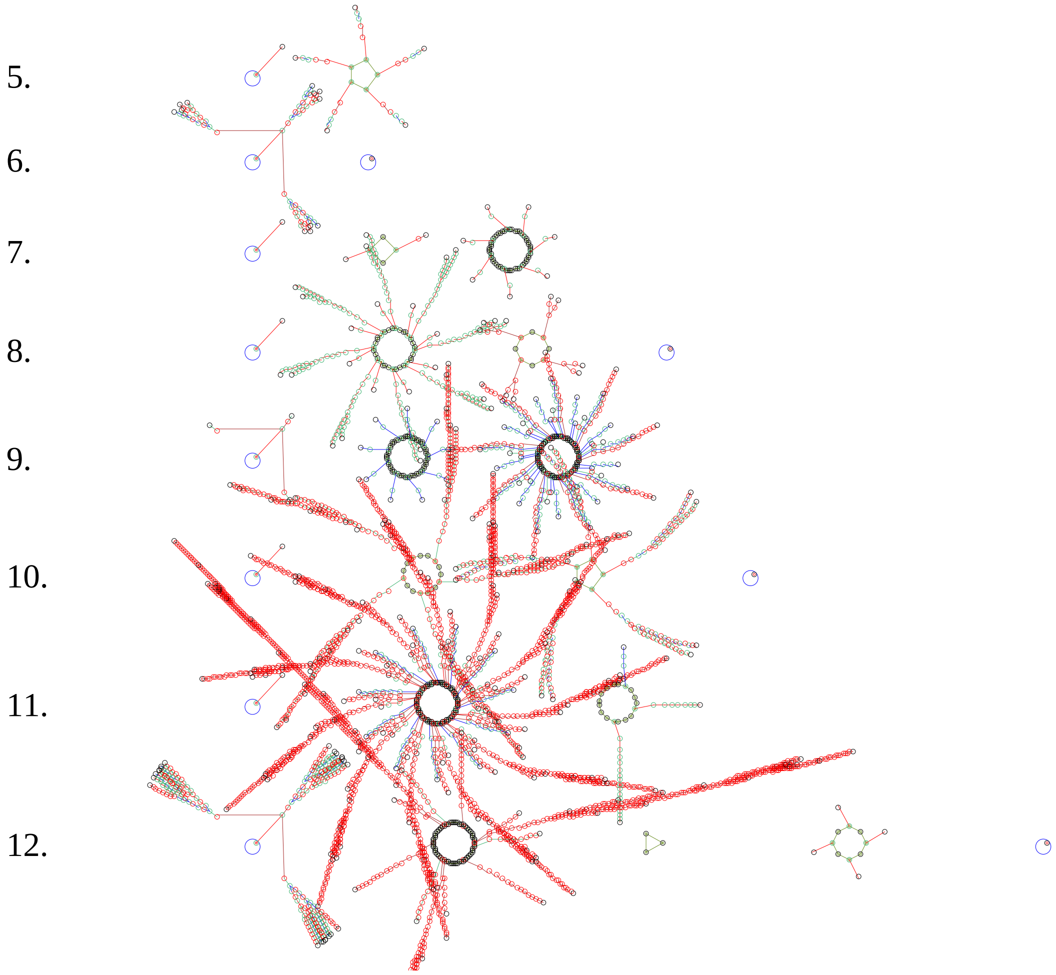


Figure 4.5: Basin of attraction fields for a range of network size n=5-12. $v2k3$, rule(dec)=30

4.5 A single basin of attraction

Backtrack with **q** (or right mouse button) to the start of the program.

1. At the very first prompt enter **s**.
2. At the **Neighborhood size** . . . prompt, enter 4.
3. Enter **return** until the top center **output parameters** , then **a** to restore all defaults, then **d** to skip further special options.
4. Enter **return** in response to further prompts in top-right windows.
5. A singe basin for a CA, size 14, is generated.
6. Enter **return** for the next mutant.
7. Toggle the display of the “backwards” space-time patterns with **s** and the state-space matrix with **m** as described in 4.3.
8. Enter **q** to interrupt and backtrack up the prompt sequence.

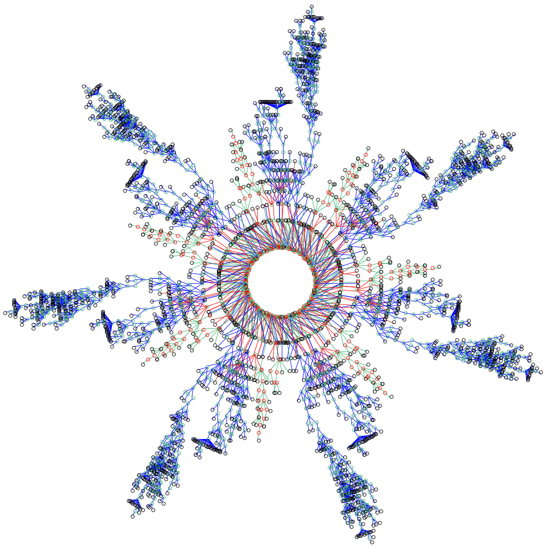


Figure 4.6: A single basin of attraction with 4333 states, and an attractor period of 140, $v2k4$, n=14, hex rcode=76b5, decimal seed=3187.

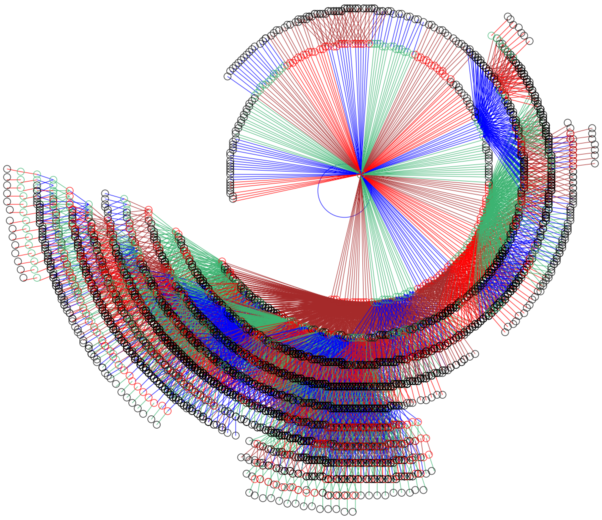


Figure 4.7: A single basin of attraction with 15541 states and an attractor period of one, a point attractor, $v2k4$, n=14, hex rcode=ac88, decimal seed=3187.

4.6 A subtree

Backtrack with **q** (or right mouse button) to the start of the program.

1. At the very first prompt enter **s**.
2. Enter **return** in response to further prompts until the prompt **Network size** . . . **:** select 27.
3. At the prompt **Neighborhood size** . . . **:** select 5.
4. Enter **return** in response to further prompts until the **Select SEED** . . . **:** window appears. Enter **r** for a "random seed" then **a** to set *all* cells at random.
5. Enter **return** until the top center **output parameters** , then **a** to restore all defaults, then **d** to skip further special options.
6. At the prompt **backward for subtree-b, forward for basin-(def):** select **b**.
7. At the next prompt, **forwards before backwards?**
   how many steps (default 0): select 3.

   This runs the CA forward by 3 time-steps (from the "seed"), before running backward from the state reached. The original randomly selected seed is likely to be a "garden-of Eden" state with no predecessors, so not much use as the root of a sub-tree.
8. Enter **return** in response to further prompts in top-right windows. The subtree will be generated. To generate a bigger sub-tree, enter a greater number of forward time-steps at the previous prompt. However, this might reach an attractor state. In this case the whole basin will be generated with the message **subtree=basin** in the top-right data window. If this is taking too long, enter **q** to interrupt and backtrack to reduce the number of forward time-steps.

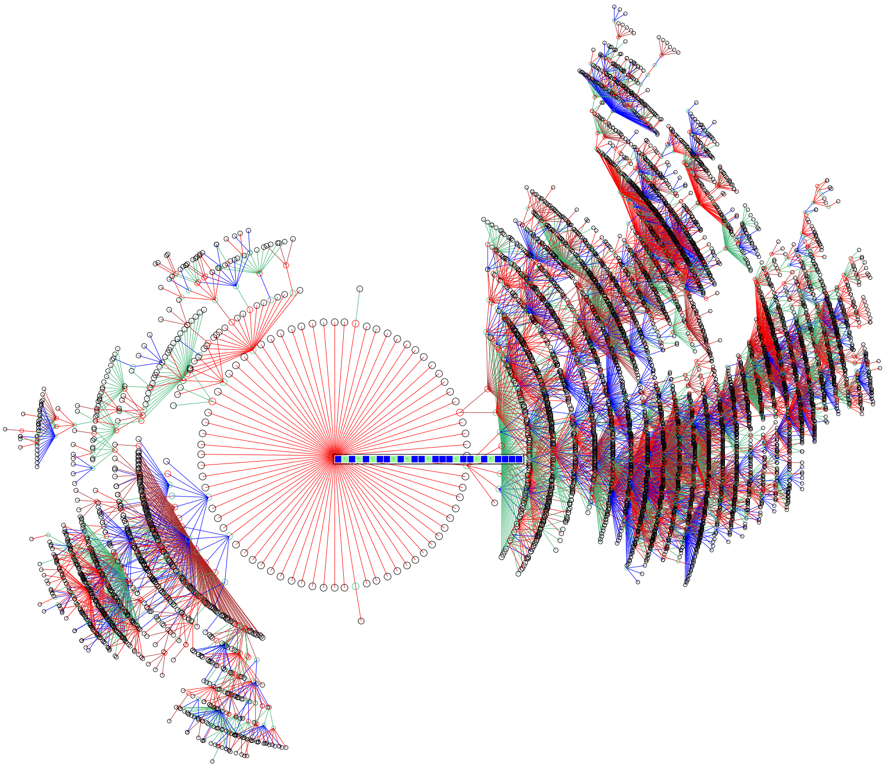


Figure 4.8: A subtree with 11324 states generated from the bit pattern at the center. 1d CA, $v2k4$, $n27$, hex rcode 5afbb1ae.

9. Enter **return** for the next mutant.
10. While the subtree is being generated, toggle the display of “backwards” space-time patterns with **s** and the state-space matrix with **m** as described in section 4.3.
11. Enter **q** to interrupt and backtrack up the prompt sequence.

To highlight the “root” state as a bit pattern, backtrack to **output parameters** , enter **p** for display, then then **return** until the prompt **highlight attractor (or subtree root . . . :** enter **1**. There are various ways of displaying states or nodes in basins, described in section 26.2.

4.7 1d Space-time patterns

Backtrack with **q** (or right mouse button) to the start of the program.

1. At the very first prompt **SEED:forward/single basin/subtree-s (FIELD-def):** enter **s**.
2. At the prompt **Network size . . . :** select 150.
3. At the prompt **Neighborhood size . . . :** select 5.
4. Enter **return** until the top-center **output parameters** banner appears, then enter **s** for **space-time pattern only**.
5. At the next prompt, **FORWARDS ONLY options:**. . . the top-center **accept defaults-d** banner appears, enter **d** to skip yet more special options.

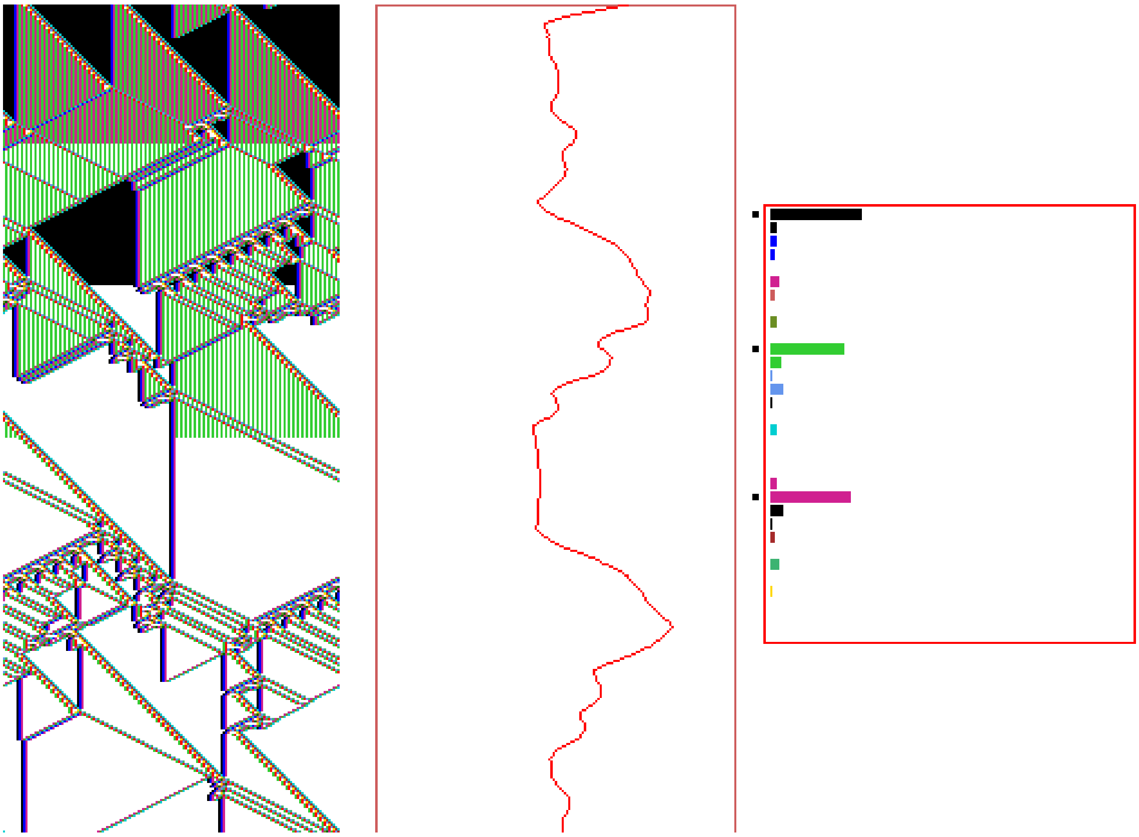


Figure 4.9: A space-time pattern of a “complex” 1d CA, $k = 5$, hex rcode e9 f6 a8 15, $n = 150$. About 360 time-steps, including some analysis shown by default. *left*: The space-time pattern colored according to neighborhood, and progressively “filtered” at three times with key **f**, suppressing the background domain to show up “gliders” more clearly. *center*: The input-entropy/time plot. *right*: The lookup frequency histogram for the last time step shown.

A space-time pattern is generated, scrolling upwards, on the left of the screen. To the right is a histogram of the lookup frequency for each neighborhood relating to a window of 10 time-steps, and a plot of the entropy of this histogram, the “input-entropy”.
The **on-the-fly key index** on the right of the screen gives a complete list of immediate changes that can be made with a key press. Some of these options are repeated on the right of the bottom title bar. Try the following to see what happens:

- **g** to change the rule to a “complex” rule, chosen at random from a database of complex rules in the file g_v2k5.r_s, available in the “ddextra” download.
- **3** to toggle cell color; according to the lookup neighborhood or the value.
- **u** to toggle the input-entropy - density plot.
- **4** for a new random initial state.
- **f** to progressively “filter” the space-time pattern, and **a** to restore the unfiltered pattern.
- **1** for a random “bit-flip” (mutate one output in the rule-table), and **2** to flip back. If “bits” are flipped successively with **1**, **2** will flip them back in reverse order.

- **e** and **c** to expand and contract the scale of the space-time pattern.

6. Try the many other on-the-fly key presses to change the rule, seed, color, analysis, size, updating, frozen cells, dimension, etc.
7. Enter **q** to pause, a top-right interrupt prompt appears with many options. To select a particular rule, such as the rule in figure 5, enter **v** to revise the rule. A bottom-right prompt appears, enter **return**, then **h** for the new rule in hex, then enter the hex characters e9 f6 a8 15. Then enter **return** twice to continue the space-time pattern from where it left off, but with the new rule.
8. Enter **q** to backtrack further.

4.7.1 1d ring of cells, and scrolling the ring

A 1d CA has periodic boundaries, effectively a ring or circle of cells. Space-time patterns can be displayed as a movie of the changing patterns on this ring (section 32.15 and figure 32.17), and the ring itself can be scrolled, making a scrolling tube (figure 4.10).

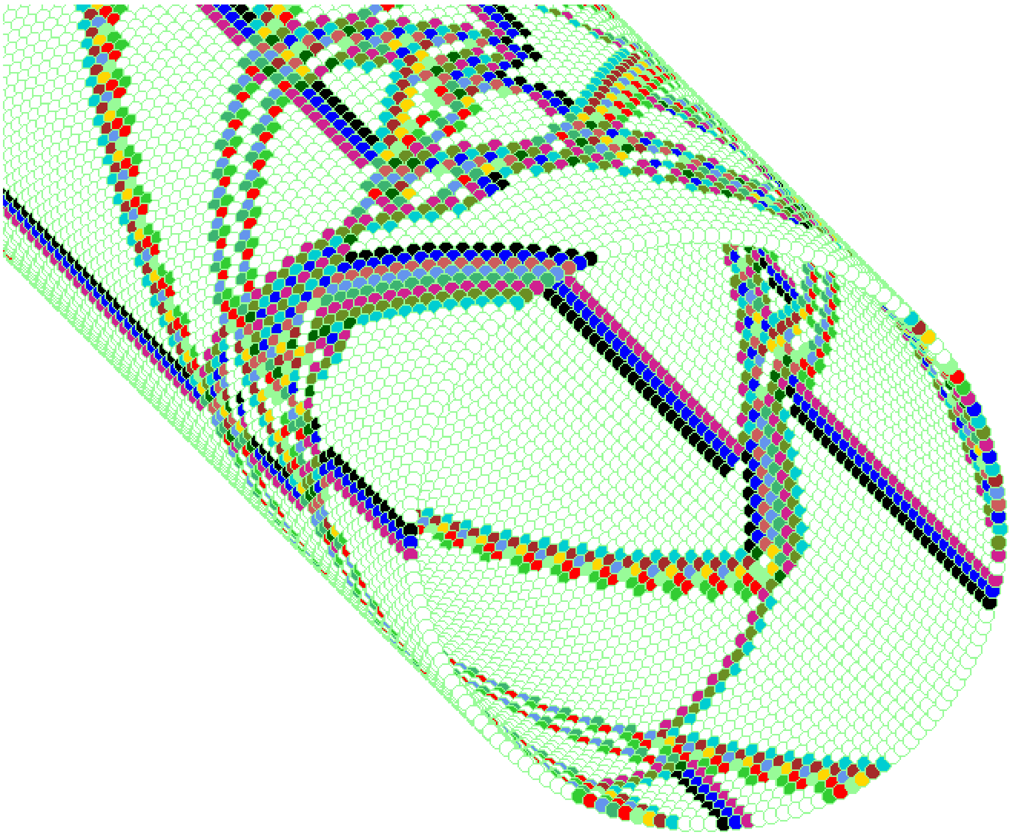


Figure 4.10: A 1d space-time pattern shown as a ring of cells scrolling diagonaly upward – a scrolling tube. The present moment is the ring at the bottom right – the closest ring. The space-time pattern is colored according to neighborhood, and has been filtered. 1d CA, $k = 5$, hex rcode e9 f6 a8 15, $n = 150$.

To display the ring alongside the normal space-time pattern (figure 5) above:

1. Set up the normal space-time pattern as in section 4.7.
2. Enter **q** for the interrupt prompts.
3. Enter **g** to show the network as a graph, then **return**. A circle of cells will appear.
4. Enter **q** to exit the graph and continue. On-the-fly changes work on the circle as well as the normal space-time pattern.

4.7.2 Scrolling the ring

To display the ring and make it scroll:

1. Set up the mormal space-time pattern as above in section 4.7.
2. Enter **T** on-the-fly to display the 1d space-time pattern in 2d.
3. Enter **q** for the interrupt prompts.
4. Enter **g** to show the network as a graph, then **return**. A circle of cells will appear. The appearence/position of the ring can be altered (chapter ??) but the default setting will be fine.
5. Enter **q** to exit the graph and continue. Space-time patterns will play out as a movie on the ring.
6. Enter # to toggle scrolling the ring.
7. Enter **J** to toggle hiding the 2d space-time pattern, which may also scroll superimposed on the ring.

Initially the ring will move diagonaly towards the bottom right, then scroll diagonaly upward, so that the present moment is the ring at the bottom right.

Most on-the-fly options work as usual with the ring. Try:

- **3** to toggle to cell color; according to the lookup neighborhood or the value.
- **@** to toggle the cell outline.
- **c** to contract (or **e** expand) the spacing between successive rings,
- **4** for a new random initial state.
- **f** to progressively filter, **a** to totally unfilter,
- **4** for a new random initial state.

The size and position of the ring (or any arbitrary layout of cells) can be manipulated as described in chapter 20.

4.7.3 Multi-value 1d space-time patterns

Backtrack with **q** (or right mouse button) to the start of the program.

1. At the very first prompt **... TFO:totalistic/forward only-t**... **:** enter **t**.
   The first prompt changes to:
   EXIT-q, graphics setup-g, randseed-r, disable TFO allow basins-b
   TFO:totalistic rules and forwards only-(def): (note **b** restores the original first prompt).

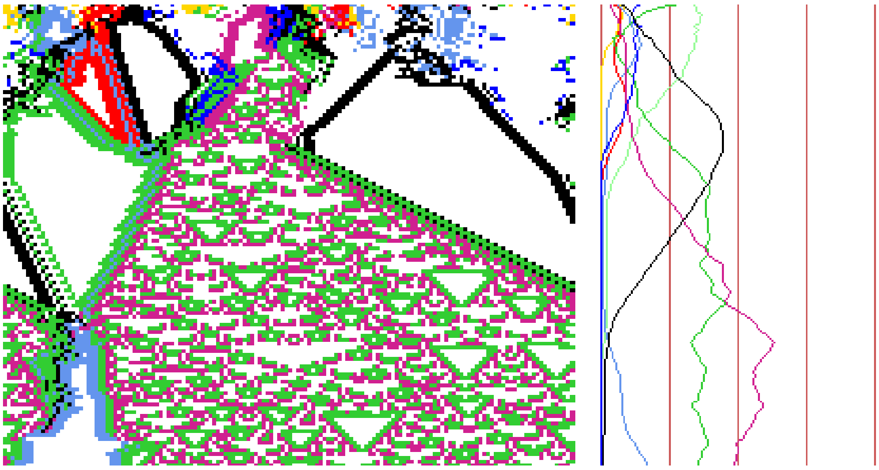


Figure 4.11: A 1d CA with a filtered Altenberg rule, kcode $v8k7$, n=150, where the probability of a rule-table output depends on the fraction of colors in its neighborhood. On the right the color density is plotted for each of the 8 colors, relative to a moving window of 10 time-steps.

TFO-mode allows combinations of greater $[v, k]$ is required than SEED-mode with option **s** in the previous space-time pattern examples (section 4.7). As well as constraining DDLab to various types of totalistic rules (with shorter rule-tables), TFO-mode disables attractor functions.

2. At the second prompt **value-range** . . . **:** select 8, the current maximum in DDLab.
3. At the prompt **Network size** . . . **:** select 150.
4. At the prompt **Neighborhood size** . . . , select 7.
5. Enter **return** until the promp **FORWARDS ONLY options:**. . . (and the top-center **accept defaults-d** banner) appears, enter **d** to skip special options.

   A space-time pattern is generated, scrolling upwards, on the left of the screen, as in section 4.7. Because of high $[v, k]$, the randomly selected rule will appear extremely disordered, with high entropy. To set a rule with a much more ordered pattern, as in figure 4.11, or a rule with symmetry, try the following on-the-fly:

   - Enter **A** for an "Altenburg" totalistic rule, where the output of each neighborhood relates to the fraction of colors in the neighborhood (section 16.9).
   - Enter **s** to toggle the density plot, the fraction of colors relating to a moving window of 10 time-steps.
   - Enter **r** for a random rule followed by **A** for another Altenburg rule, and **4** for a random initial state.
   - To filter the space-time pattern, first toggle with **s** to show the input-entropy plot instead of the density plot, then enter **f** repeatedly as required to filter domains, highlighting the domain boundaries.
   - Enter **k** for a k-totalistic rule (or **K** for a t-totalistic rule, then **5** for a singleton seed, one cell against a uniform background of a different color. Note the symmetry of the pattern, which must be conserved however many times the totalistic rule is changed with **k**, **K** or **A**.

4.7.4 Noisy space-time patterns

Its possible to introduce noise for any space-time pattern updating, in the 1d examples above, and the 2d and 3d examples to follow. To do this:

1. At the output parameters prompts (backtrack if necessary), enter **s**.
2. At the prompt **FORWARDS ONLY...:** enter **return** (about 6 times), until the prompt **set probability (def 100%) 0-100, update:** enter **return**.
3. There will be a continuation of the prompt ... **output:** enter 99. This sets the updating of cells to be accurate 99% of the time, so 1% at random (the alternative "update" would mean that 1% stay the same).
4. enter **d** to skip more options and start space-time patterns.

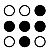
Try this with "complex" rules (on-the-fly **g**, section 4.7), and with the "Altenburg" rules (on-the-fly **A**, section 4.7.3).

4.8 2d space-time patterns ("game-of-Life")

Backtrack with **q** (or right mouse button) to the start of the program.

1. At the very first prompt **SEED:forward/single basin/subtree-s (FIELD-def):** enter **s**.
2. At the second prompt **Value range ...:** enter 2.
3. Enter **return** until a top-right **WIRING** prompt window appears,

   WIRING: special-s load-l random-r
   regular: 3d-3, 2d-2, (hex+x), 1d-def: select 2 for regular 2d wiring.

   Any previous network and neighborhood size settings will be superseded.
4. Enter **return** until the top-right prompt, **2d, enter width (def-40):** enter 66, then a continuation prompt **depth (def 66):** enter **return**, for a 66 × 66 square lattice.
5. At the next top-right prompt,
   Neighborhood size k: mixed-m, or enter 1-13 (def 3): enter 9.
6. Enter **return** until the main sequence prompt for rule selection appears,
   Select v2k9 rule ...: select **L** for the "game-of-Life". The lookup-table of the rule will be displayed as a bit pattern.
7. Enter **return** until the prompt **Select SEED (v2 2d ij=66,66)...:** enter **return** for a random block.
   Alternatively select **e** to "empty" all cells to zero, then **l** to load a seed, then enter the file name "pento" at the **LOAD SEED...** prompt. This is the "r-pentomino" pattern  that guarantees gliders.
8. Enter **return** until the top center output parameters banner appears, then enter **s** for **space-time pattern only**.
9. At the next prompt, **FORWARDS ONLY options:...**, a top center accept defaults-d banner appears, enter **d** to skip more special options.
   The 2d space-time pattern is generated in the top-left corner of the screen. The **on-the-fly key index** appears on the right of the screen. Try the following (among others) to see what happens.

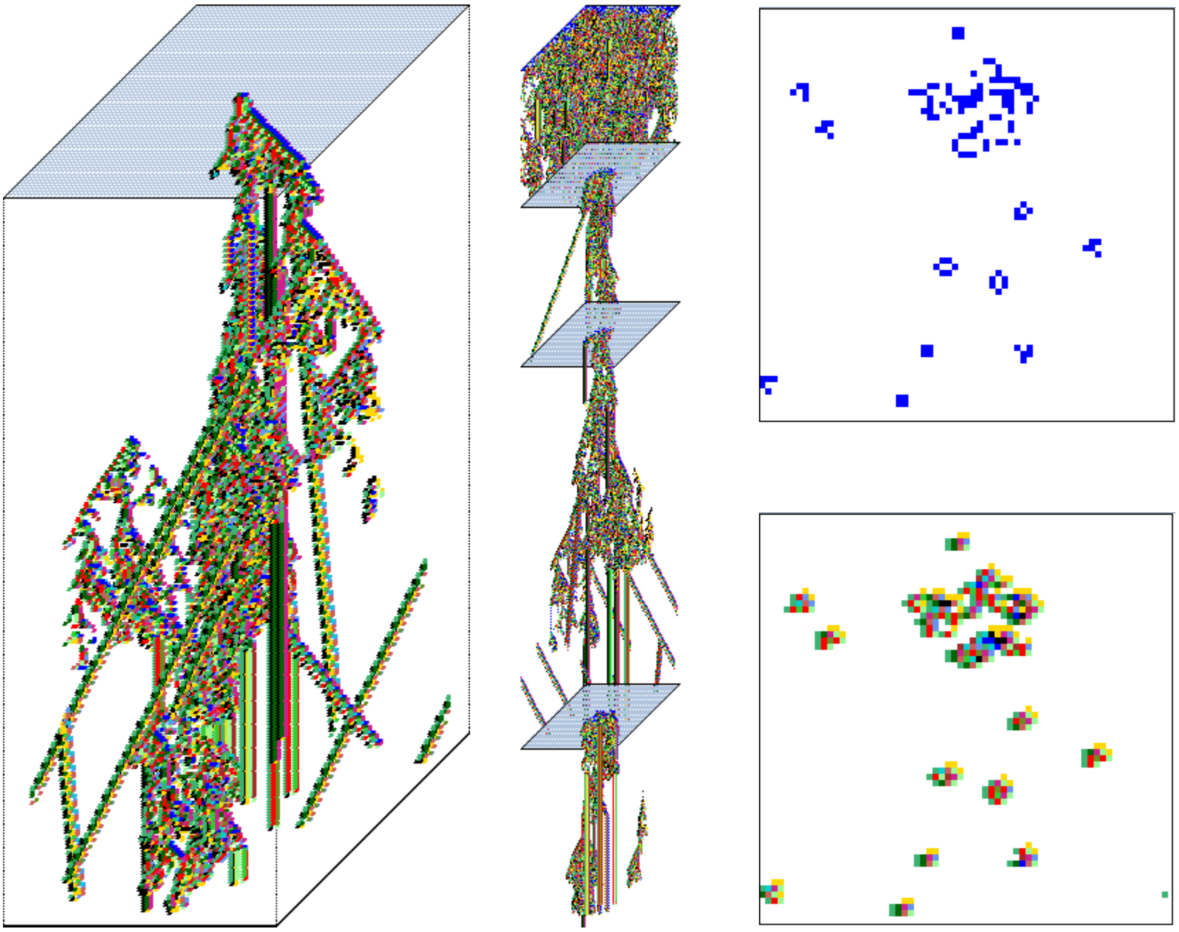


Figure 4.12: Space-time pattern of the 2d game-of-Life, (k=9, $n = 66 \times 66$) stacked below each other in a isometric projection scrollin upwards. *left*: Starting from the "r-pentomino" seed. *center*: Re-scaled to the smallest scale, new seeds set at intervals. *upper right*: The state at time-step 230. *lower right*: The same state colored according to the neighborhood look-up instead of the value.

- **3** to toggle cell color; according to the lookup neighborhood or the value.
- **h** to toggle three ways of displaying the stability of the pattern.
- **o** to restore the original seed.
- **t** to toggle between the 2d display and an isometric projection scrolling vertically upwards.
- **#** to toggle between the 2d display and an isometric projection scrolling diagonally upwards.
- **P** to toggle skipping aleternate time-steps.
- **k** for a new random central block. **4** for a fully random seed.
- **e** and **c** to expand and contract the scale of the space-time pattern.

10. Enter **q** to interrupt and backtrack up the prompt sequence.

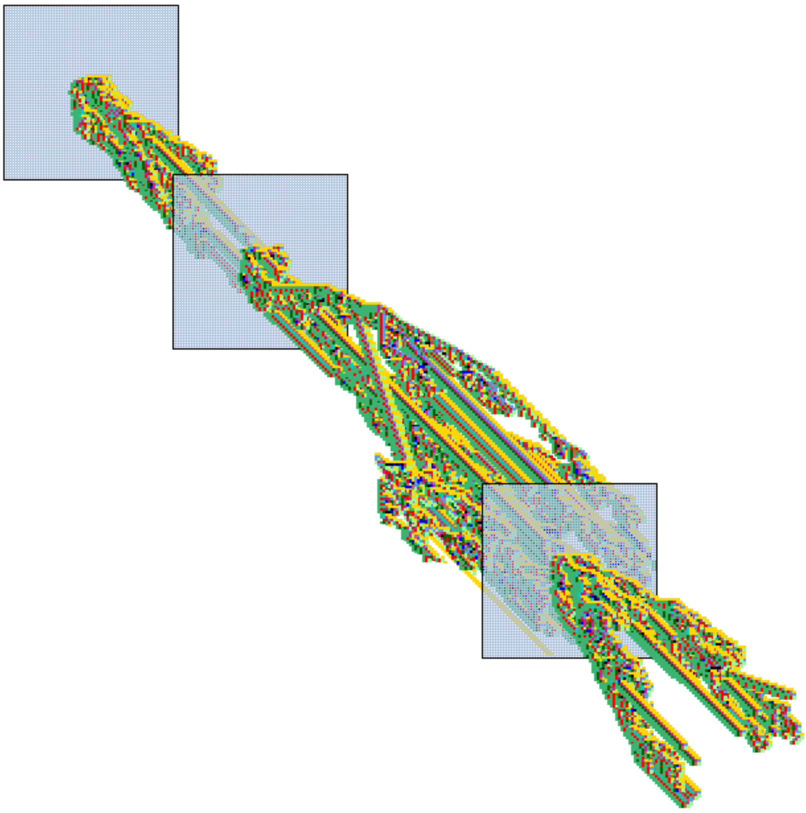


Figure 4.13: Space-time pattern of the game-of-Life (k=9, $n = 66 \times 66$) stacked in front of each other in a isometric projection scrolling upwards diagonally, with new seeds set at intervals, and alternate time-steps skipped.

4.9 2d Space-time patterns (binary totalistic rules)

Backtrack with **q** (or right mouse button) to the start of the program.

1. At the very first prompt **SEED:forward/single basin/subtree-s (FIELD-def):** enter **s**.

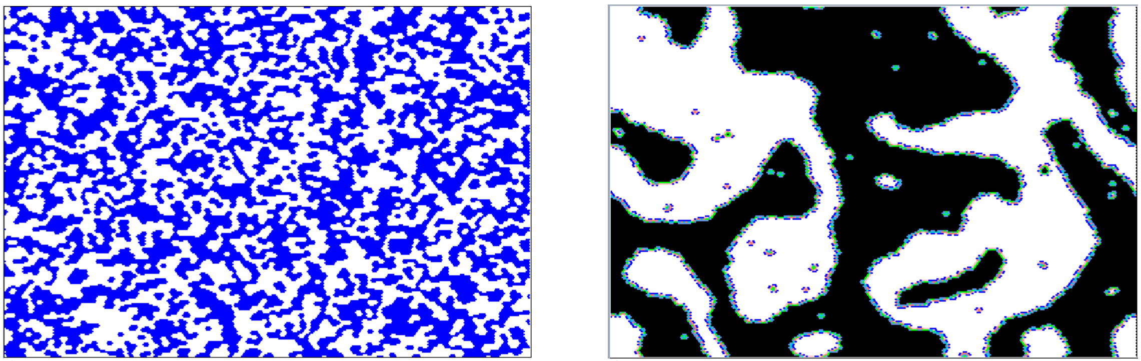
2. Enter **return** until a top-right **WIRING** prompt window appears,

   WIRING: special-s load-l random-r
   regular: 3d-3 2d-2 1d-def: select 2 for regular 2-d wiring.

   Any previous network and neighborhood size settings will be superseded.

3. At the next top-right prompt, **2d, enter width (def-40):** enter 240, then a continuation prompt **depth (def 240):** enter **return**, for a 240 × 240 square lattice.

4. At the next top-right prompt,
   Neighborhood size k: mixed-m, or enter 1-13 (def 3): enter 7.



(a) k7 majority rule
totalistic tcode=11110000.

(b) k7 modified majority rule
totalistic tcode=11101000.

Figure 4.14: 2d space-time patterns for a k7 CA on a hexagonal grid (240x240), from random initial states showing aggregating behavior.

This results in a hexagonal 2d array, where each cell has 6 nearest neighbors.

5. Enter **return** until the main sequence prompt **totalistic: tcode-t...:** enter **t**.

6. At the prompt **Select v2k7 rule ...:** enter **m**. The lookup-table of the majority rule will be displayed as a bit pattern with a flashing cursor on the left bit. Enter **return** to accept the majority rule 11110000 as in figure 4.14(a), or first modify the bit pattern to 11101000 (enter 1 or 0, arrow keys to move) as in figure 4.14(b).

7. Enter **return** until the prompt **Select SEED (v2 2d ij=66,66)...:** enter **return** for a random, then **a** for "all" of the lattice, as opposed to a central block.

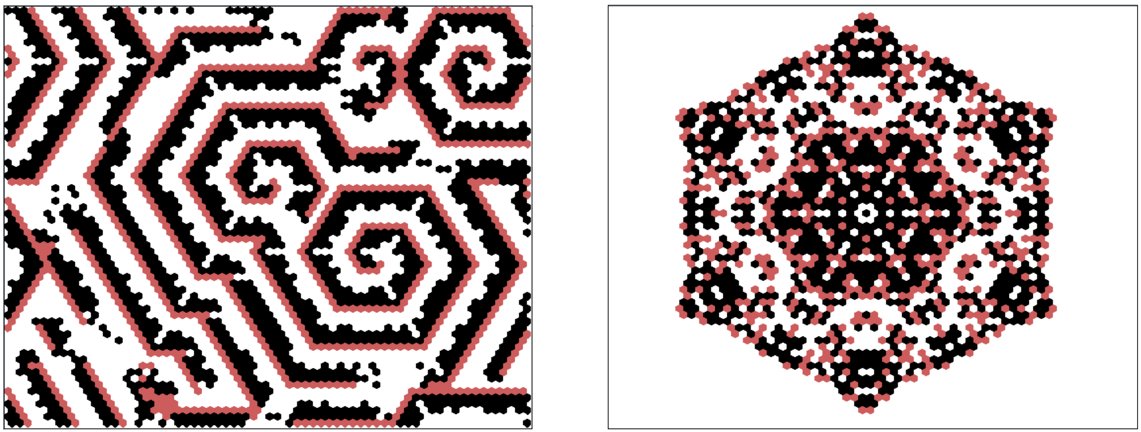
8. Enter **return** until the top center **output parameters** banner appears, then enter **s** for **space-time pattern only**.

9. At the next prompt, **FORWARDS ONLY options:...**, a top center **accept defaults-d** banner appears, enter **d** to skip more special options.

   The 2d space-time pattern is generated in the top-left of the screen. An **on-the-fly key index** on the right of the screen. Try the following (among others) to see what happens.

   - **4** for a new random seed.
   - **e** and **c** to expand and contract the scale of the space-time pattern.
   - **3** to toggle between cells colored according to rule-table lookup or by value.
   - **h** to toggle three ways of displaying the stability of the pattern.
   - **U** to toggle sequential updating.
   - **Y** to toggle partial order updating.

10. Enter **q** to interrupt and backtrack up the prompt sequence.



(a) v3k6 spiral-rule totalistic code(hex)=0a0282815a0154, random initial state.

(b) v3k6 rule, picked at random totalistic code(hex)=a0468298295220, singleton seed.

Figure 4.15: 2d space-time patterns, v3k6 k-totalistic 2d CA on a hexagonal grid (240x240).

4.10 Totalistic rules - forwards only - TFO mode

Backtrack with **q** (or right mouse button) to the start of the program.

1. At the very first prompt
   Exit-q, graphics setup-g, randseed-r, TFO:totalistic/forward only-t
   SEED:forward only/single basin/subtree-s (FIELD-def): enter **t**.

   This restricts DDLab to totalistic rules only expressed as tcode or kcode, which is much shorter than a full lookup-table, and therefore allows larger neighborhood. Functions are restricted to run forward only, so attractor basins are excluded.

2. The prompt changes to
   Exit-q, graphics-g, randseed-r, disable TFO allow basins-b
   TFO:totalistic rules and forwards only-(def):, enter **return**.

   (To reset DDLab back to full lookup-tables and allow basins you would enter **b**).

3. At the next prompt **Value range** . . . **:** enter 3.

4. Enter **return** until a top-right **WIRING** prompt window appears,

   WIRING: special-s load-l random-r
   regular: 3d-3, 2d-2(hex+x), 1d-def: enter 2x for hexagonal 2d wiring.

   Any previous network and neighborhood size settings will be superseded.

5. Enter **return** until the top-right prompt, **2d, enter width (def-40):** enter 240, then at the continuation prompt **depth (def 240):** enter **return**, for a 240 × 240 lattice.

6. At the next top-right prompt,
   Neighborhood size k: kmix-m, or enter 1-25 (def 3): enter 6.
   This results in a hexagonal 2d array, where each cell has 6 nearest neighbors.

7. Enter **return** until the main sequence prompt
   totalistic only: outertot-o/+o, tcode-t, (def-kcode): enter **return** for kcode.

8. At the prompt **Select v3k6 rule ... :** enter **h** to specify the kcode in hex. Then enter 0a0282815a0154 for the kcode in figure 4.15(a). To correct an entry use **backspace** or the arrow keys.

9. Enter **return** until the prompt **Select SEED (v3 2d ij=240,240), ... :** enter **return** for a random central block.

10. Enter **return** until **FORWARDS ONLY options:...**, appears, and a top center **accept defaults-d** banner appears, then enter **d** to skip more special options.

    The 2d space-time pattern is generated in the top-left of the screen. An **on-the-fly key index** also appears on the right of the screen. Try the following (among others) to see what happens.

    - **4** for a new random seed.
    - **e** and **c** to expand and contract the scale of the space-time pattern.
    - **3** to toggle between cells colored according to rule-table lookup or by value.
    - **#** to toggle diagonal scrolling of the 2d space-time.
    - **h** to toggle three ways of displaying the stability of the pattern.
    - **U** to toggle sequential updating.
    - **Y** to toggle partial order updating.

11. Enter **q** to interrupt and backtrack up the prompt sequence.

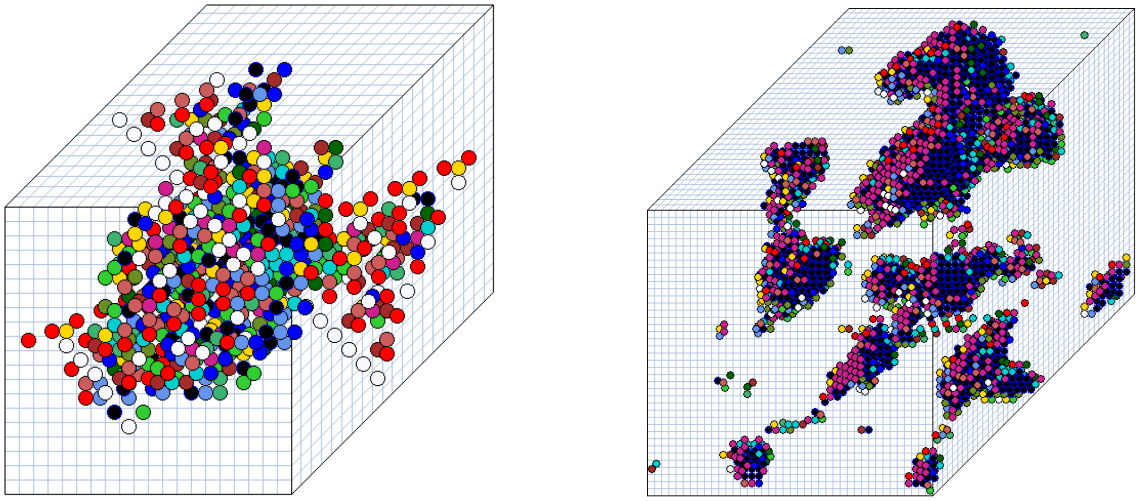
4.11 3d space-time patterns

Backtrack with **q** (or right mouse button) to the start of the program.

1. At the very first prompt
   SEED:forward/single basin/subtree-s (FIELD-def): enter s.

2. Enter **return** until a top-right **WIRING** prompt appears,
   WIRING: special-s load-l random-r
   regular: 3d-3 2d-2 1d-def: select 3 for regular 3d wiring.

   Any previous network and neighborhood size settings will be superseded.



(a) 20 × 20 × 20,
random $k7$ rule from a singleton seed

(b) 40 × 40 × 40,
k7 totalistic code=11101000

Figure 4.16: Examples of 3d CA with a k=7 neighborhood arranged as a 3d cross. The projection is isometric seen from below, as if looking up at the inside of a cage. Cells are shown colored according to neighborhood lookup by default for a clearer picture (instead of by value: 0,1).
(a) shows the evolution of a 3d CA, $k = 7$, $n = 20 \times 20 \times 20$, with a randomly selected rule. The initial state is a "singleton seed", a single on cell in an otherwise empty array.
(b) shows a 3d CA, $k = 7$, $n = 40 \times 40 \times 40$ (the maximum size DDLab supports), The initial state was set at random, but with a bias of 45% of on cells. The rule is the same as in section 4.9, showing reaction-diffusion behavior.

3. Enter **return** until the top-right prompt, to set the size of the 3d lattice,
   3d, enter width (def-9): depth (def 9): height (def 9): enter the width depth and height. The maximum cube would be 40 × 40 × 40. The array has 3-torus boundary conditions.

4. At the next top-right prompt,
   Neighborhood size k: mixed-m, or enter 1-13 (def 3): enter 7. The neighborhood is arranged as a 3d cross.

5. Enter **return** until the main sequence prompt for rule selection appears,
   Select v2k7 rule (S=128): empty-e fill-f maj-m prtx-x Alt-A life-L chain-c rand-r bits-b ReDiff-R hex-h rep-p load-l table-t (def-r):
   enter **return** for a random rule.

   Alternatively follow the steps in section 4.9 to set the "totalistic code". The lookup-table of the rule will be displayed as a bit pattern.

6. Enter **return** (to accept defaults including the seed) until the top center **output parameters** banner appears, then enter **s** for **space-time pattern only**.

7. At the next prompt, **FORWARDS ONLY options:**. . . , a top center **accept defaults-d**

banner appears, enter **d** to skip yet more special options.

The 3d space-time pattern is generated in the top-left of the screen. An **on-the-fly key index** on the right of the screen gives a list of key presses to change settings on-the-fly. Try the following to see what happens.

- **5** for a new "singleton" seed, a single central 1, or **k** for a new random central 3d block.
- **r** for a new random rule.
- **e** and **c** to expand and contract the scale of the space-time pattern.
- **3** to toggle between cells colored according to rule-table lookup (the default) or by value.

8. Try other key presses to change the updating, frozen cells, dimension, etc.

9. Enter **q** to interrupt and backtrack up the prompt sequence.

4.12 "Screen-saver" demo

The "screen-saver" demo[2] provides a continuous show of single basins of attraction or subtrees "boiling" on the screen, generated at random positions, from random seeds, and according to the network architecture selected. For each successive basin (or subtree) the rules or wiring are mutated as specified in chapter 28 – the default is a random rule. For larger systems its unlikely that the same basin would ever repeat.

For a screen-saver demo of a CA similar to figure 4.17, backtrack with **q** (or right mouse button) to the start of the program.

1. At the very first prompt **SEED:forward/single basin/subtree-s (FIELD-def):** enter **s**.

2. At the second prompt **Value-range v (def 2, max 8):** select 3.

3. At the **Network size** ... prompt, select 10.

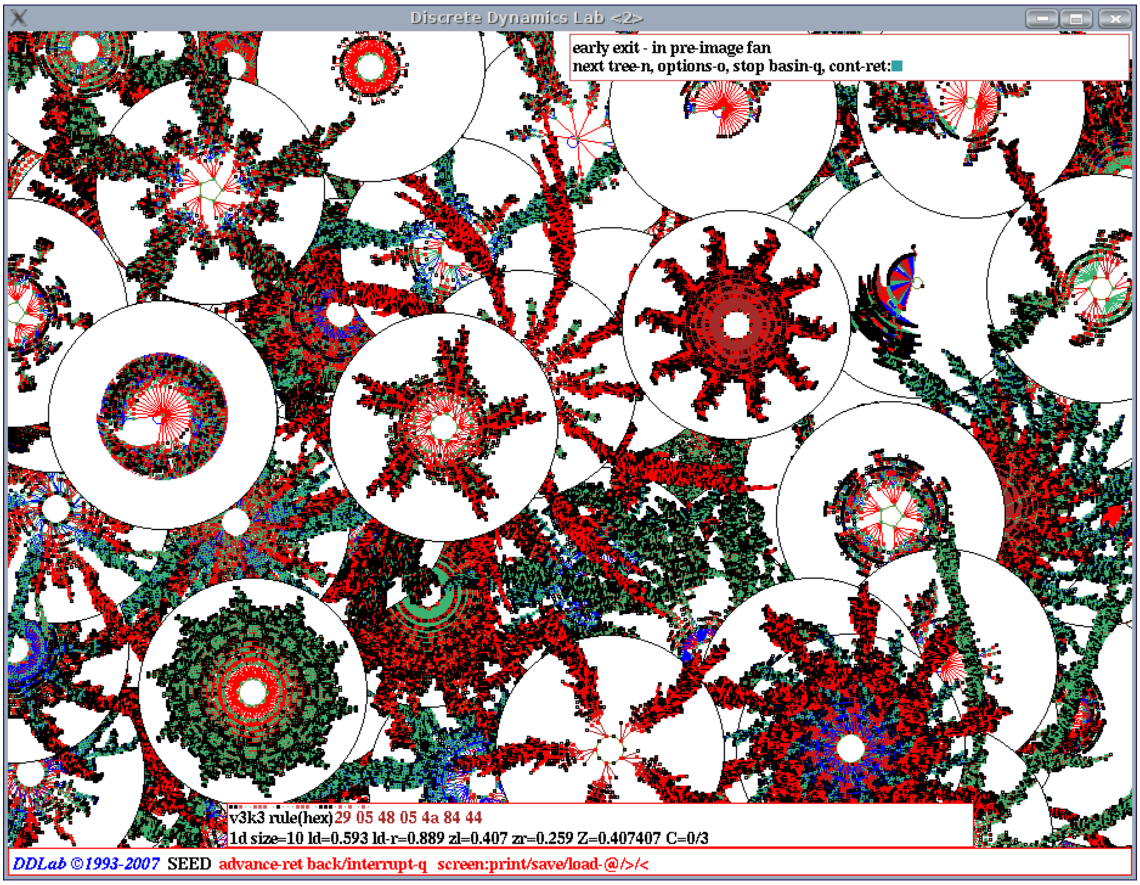
4. At the **Neighborhood size** ... prompt, select 3. Enter **return** until the top center **output parameters** banner appears, then **a** to restore all defaults, then **return** until the **savescreen demo -s** prompt; enter **s** to accept the option.

5. Then enter **d** twice (or **return**) to accept all further defaults. The demo will start. The rule picked at random will be shown in the bottom rule-window.

6. While the demo is in progress, toggle the display of the "backwards" space-time pattern with key **s**, and the state-space matrix with key **m** as described in section 4.3.

7. Enter **q** to interrupt and backtrack up the prompt sequence.

[2] See also section 24.8



early exit - in pre-image fan
next tree-n, options-o, stop basin-q, cont-ret:

v3k3 rule(hex)29 05 48 05 4a 84 44
1d size=10 ld=0.593 ld-r=0.889 zl=0.407 zr=0.259 Z=0.407407 C=0/3

DDLab ©1993-2007 SEED advance-ret back/interrupt-q screen:print/save/load-@/>/<

Figure 4.17: Screen-saver demo for Cellular Automata, $v = 3$, $k = 3$, $n = 10$. Single basins with rules and initial states chosen at random are generated at random positions in the DDLab window.

4.13 Random Boolean Networks and Discrete Dynamical Networks

Until now, the Quick Start Examples have dealt with cellular automata, which have a homogeneous *local wiring template* and a single *rule* throughout the network. A random Boolean network (RBN), or the more general Discrete Dynamical Network (DDN, where $v > 2$), departs from CA network architecture by allowing each cell in the network to have different nonlocal *wiring*, a different *rule*, and different k (the number of inputs), or any combination of the above. However, the value-range v must be homogeneous.

The network can be assigned a *wiring-scheme*, *rule-scheme* and *k-mix* in a variety of ways, including randomly (with or without biases).

- just a **wiring-scheme** may be set, the *rule* and k remain homogeneous, as in CA, by default.

- just a **rule-scheme** may be set, the *local wiring template* and k remain homogeneous, as in CA, by default.

- both a **wiring-scheme** and **rule-scheme** may be set, k remains homogeneous by default. If $v = 2$, this is what is usually understood as a random Boolean network (RBN).
- a k**-mix** may be set, which implies both a *wiring-scheme* and a *rule-scheme*, but the *wiring-scheme* remains *local* by default.
- a network with a k**-mix**, a non-local **wiring-scheme** and a **rule-scheme** may be set.

To generate attractor basins and space-time patterns for DDN, for any combination of parameters listed above, the preceding examples in sections 4.2-4.12 can be adapted as described below.

Note that running backwards for networks with non-1d-CA wiring employs a different algorithm, with a greater computational load, than for 1d CA wiring (see section 1.6), so when generating attractor basins for non-1d-CA wiring v, k, and especially n, should be kept small, and 2d or 3d networks should be avoided.

1. For a random wiring in 1d: At the first top-right prompt,
   WIRING: special-s load-l random-r
   regular: 3d-3, 2d-2 (hex+x), 1d-def:
2. select **r** for 1d random wiring.
3. select **s** for special wiring, which includes 2d or 3d random wiring, then follow further top-right prompts as follows:
   (a) **3d-3, 2d-2 (hex-x), 1d-def:** select the dimension, i.e. **2**.
   (b) **hand wire-h,**
   regular 2d-2 (hex+x), 1d-1, random-def: enter **return**.
   (c) **2d, enter width** … **depth** … set the size.
   (d) **Neighborhood size k: kmix-m** … select k, or enter **m** for mixed k, in which case there will be a series of further prompts to set the kmix (see section ??), but just enter **return** for a random kmix.
   (e) **bias random wiring:**… these further set of prompts can be skipped with **d**.
   (f) Note: any new entries here for n and k will supersede earlier entries.
4. The next top-right prompt (as below or similar) allows the wiring to be examined in detail, and amended, (section ??):
5. Whatever wiring is defined, local or random, dimension, the next top-right prompt (similar to below) allows the wiring to examined in detail, and amended, (section ??):

   2d network (122x122),review/revise, wiring only - rule not set
   graph-g, matrix: revise-m, view-M printxterm+p
   graphic:1d+timestep-1 circle-c, 2d-2:
6. For a random **rule-scheme**, at the top-right prompt,
   RULES: single rule (def), load rule mix-l, …
   mix: no limit-n, or set limit up to 200: enter **n**.

7. For a random **k-mix** for a 1d network, at the main sequence prompt, **Neighborhood size k: kmix-m, or enter 1-13 (def 3):** enter **m**. Various further options to bias the k-mix are presented. If in doubt enter **return**.

8. For a random **k-mix** for a 2d or 3d network, first select the random wiring in 2d or 3d as above.

   At the top-right prompt, **Neighborhood size k: kmix-m, or enter 1-13 (def 3):** enter **m**. Various further options to bias the **k-mix** are presented. If in doubt enter **return**.

9. Once rules have been set, to examine and amend the resulting network in detail, (wiring and rules) a top-right prompt, similar to the one below, is presented,

   1d network (n=150),review/revise/learn, wiring and rules
   graph-g, matrix: revise-m view-M printxterm+p
   graphic: 1d+timestep-1 circle-c 2d-2:

   - Enter **return** to skip and continue, **q** to backtrack.
   - Enter **1**, **c**, **2** or **3** to review the network as a wiring graphic, using the arrow keys to examine the wiring and rules for chosen elements. (see chapter 17).
   - Enter **m** or **M** to review the network as a wiring matrix (see section 17.2)
   - Enter **g** to review the network as a graph, which can be rearanged by dragging nodes with the mouse (see chapter 20).

   Enter **return** to skip. These options, which have there own follow-up options, are also applicable to CA, of course.

Intentionally Blank

Chapter 5

Starting DDLab

This and the following chapters provide the detailed program reference, listing all the various options and prompts with explanatory notes. Before tackling this part of the manual, its a good idea to try the Quick Start Examples in chapter 4. This chapter includes starting DDLab, the initial DDLab "screen", and some permanently available options to save, load and print the DDLab screen image.

In Linux-like systems run DDLab from a terminal (xterm) window rather than directly from an icon on the desktop, because messages and data are often shown in the terminal.

Before starting DDLab, make sure you are in the DDLab directory containing the DDLab executable file(s), or in the appropriate sub-directory (see section 5.1). The executable file(s) downloaded from the DDLab website, are:

`ddlabm06`	...	for Linux-like operating systems
`ddlabm06.exe` and `dos4gw.exe`	...	for DOS.

The files in `dd_extra.tar.gz` are not essential for DDLab to run, but are required for some functions. They can be in the DDLab directory, or in a subdirectory which can be changed into within DDLab, or DDLab can be started from this subdirectory as shown in section 5.1 below. In either case the subdirectory becomes the default directory for filing (see chapter 35).

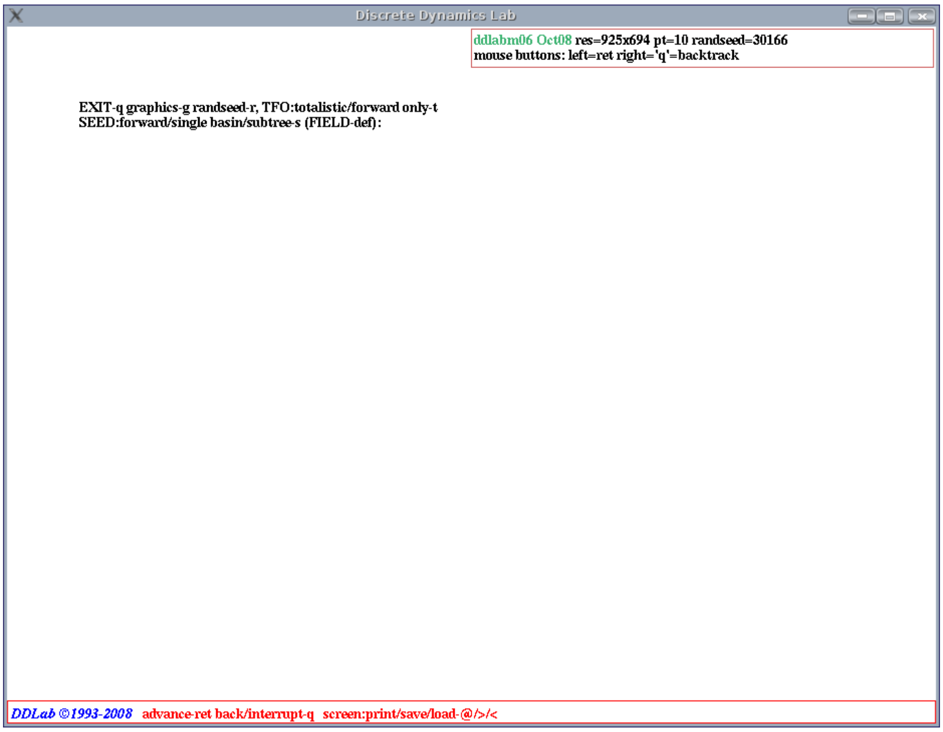
5.1 Running in Linux-like operating systems

(Refer also to section 4.1.5)

For Linux-like systems (Unix, Linux, MacOSX, Irix, and Cygwin) enter:

`ddlabm06 &`	...	if "dot" is in your path
`./ddlabm06 &`	...	if "dot" is not in your path
`../ddlabm06 &`	...	from a subdirectory (containing the files in `dd_extra.tar.gz`) of the DDLab directory.

i.e. the name of the executable followed by `&`, to retain control of the terminal (or xterm) window, where information is often printed.



```
ddlabm06 Oct08 res=925x694 pt=10 randseed=30166
mouse buttons: left=ret right='q'=backtrack

EXIT-q graphics-g randseed-r, TFO:totalistic/forward only-t
SEED:forward/single basin/subtree-s (FIELD-def):

DDLab ©1993-2008  advance-ret back/interrupt-q  screen:print/save/load-@/>/<
```

Figure 5.1: Typical graphical user interface when starting DDLab

The DDLab window will appear, similar to figure 5.1. The message "`Using backing store`" will appear in the terminal window[1], indicating that the DDLab window will be restored in the usual way when covered by other windows or moved off the screen. However, if DDLab is iconised, resized, or if the "desktop" (or "workspace") is changed, the immediate current contents in the screen will blank out[2], but the contents will reapear as DDLab continues.

5.2 Running in DOS or Windows

(Refer also to section 4.1.6)

DDLab is still compiled for old fashioned DOS in the Watcom compiler, now open source and freely available. The DOS version of DDLab can be run in pure DOS (Window 98 or prior), or in a DOS or "command line" window in later editions of Windows prior to Vista. In Vista the DOS version does not work from the command line, but it works nicely (but slowly) in "DOSBox", an open source MS-DOS emulator, intended for old PC games.

Enter `ddlabm06` or the name of the executable without the `.exe` extension. The `DOS extender` banner will briefly appear[3].

Note that the Linux-like versions perform better, and are better supported, than the DOS version, so in a Windows environment its reommended you install Cygwin, which is a freely avalable Linux-like environment for Windows, and run the Cygwin version of DDLab within that.

[1] Refer to the file `dd_linux_readme.txt` for details about "backing store", and also possible missing fonts.

[2] In FVWM the contents remain when resizing or changing desktops, they only blank out when iconising.

[3] To suppress this banner enter `set dos4g=quiet` at the DOS prompt before running DDLab.

5.2.1 Running in a pure DOS environment

In a pure DOS environment, available in Windows 98 and prior, DDLab will start, occupying the whole screen. The resolution might need to be changed to correspond to the monitor and graphics driver, otherwise fonts might appear distorted (see sections 5.7 and 6.3.2).

To get into pure DOS from Windows98, from **Start** select **Shut Down...**
then **Restart the computer** in **MS-DOS mode**.

5.2.2 Running in Windows95, 98, and later editions

In early versions of Windows, open a DOS window: in Windows95 or 98: **Start** - **Programs** - **MS-DOS Prompt**; in WindowsME: **Start** - **Run**, and type **command**. In later versions of Windows prior to Vista[4], open a "command line" or "command prompt" window, for example in WindowsXP: **Start** - **All Programs** - **Accessories** -**Command Prompt**.

From the DOS (or command line) window, DDLab will start either in the window, or in full screen (which is unpredictable) but whichever it is **Alt** + **Enter** toggles between the two. The initial resolution will be 640 × 480. Do not use the program parameter `-m` or `-h` unless in full screen.

In full screen the resolution should be changed to correspond to the monitor and graphics driver (otherwise fonts may appear distorted). However, before toggling back from a full screen to a DOS (or command line) window, the resolution must be changed back to "low" to avoid Windows warnings and unforseen consequences. In higher resolutions than 640 × 480 the mouse cursor will probably not be visible.

DDLab is more stable and faster in pure DOS (section 5.2.1 above) or in full screen as above, than in a DOS (or command line) window.

5.3 Command line arguments

Command line arguments are entered after the executable. Start with a dash `-` then the parameter(s) without spaces in any order, for example `ddlabm06 -wt &`.

The following are available,

`w`	...	white screen
`b`	...	black screen (the default)
`t`	...	TFO-mode, totalistic rules and space-time patterns only - basins disabled

DOS only

`m`	...	medium resolution 800x600
`h`	...	high resolution (VGA) 1024x768

[4] In Vista the DOS version of DDLab will run in DOSBox, see sections 3.5 and 5.2.

5.4 The UNLICENSED banner

On start-up (and also before exiting) an unlicensed version of DDLab will first display the following "UNLICENSED" banner in the center of the screen. Press **return** to continue. With a licensed this banner can be suppressed.

UNLICENSED, see www.ddlab.org, continue-ret:

5.5 Title bar

On start-up of a licensed version (or if **return** is entered at the "UNLICENSED" banner) DDLab's graphical user interface appears as described in figure 5.1

DDLab ©1993-2009 **advance-ret back/interrupt-q screen:print/save/load-@/->/<**

A title bar is displayed across the foot of the screen, including a copyright reminder, the current mode if set (TF, SEED, or FIELD), and some reminders of key presses which can be activated whenever the prompt cursor is flashing.

key press reminder	*what it means* - permanently displayed
forward-ret ...	enter **return** (or click the left mouse button) to register entries to prompts (or accept defaults) and advance through the prompt sequence.
back/interrupt-q ...	enter **q** (or click the right mouse button) to backtrack through prompt sequence or to interrupt a run.

key press reminder	*what it means* - displayed while the prompt cursor is flashing
screen:print-@ ...	enter **@** to print the DDLab screen image, described in section 5.6.2.
screen:save/load->/< ...	enter **>** to save, or **<** to load the screen image as a `.nat` file, in DDLab's own format, described in section 5.6.1.

There are other context dependent options that can be activated on-the-fly while basins of attraction or space-time patterns are being generated, and some reminders for the relevant key presses are displayed on the right within the title bar, displacing the **screen** options (see sections ?? and ??).

5.6 Saving, Loading and Printing the DDLab screen

The screen image can be saved and loaded within DDLab itself using its own file format. Alternatively the screen image can be saved and printed as a PostScript file for Linux-like systems. These options are permanently available while the prompt cursor is flashing.

5.6.1 Saving and Loading the screen in DDLab's own format

DDLab has its own file format (.nat) for saving and loading the DDLab screen image, which works for both Linux-like and DOS versions. If > or < is entered at any time when the prompt cursor is flashing, the following top-right prompt is presented,

> **SAVE SCREEN-filename(no ext) .nat will be added** *(if >)*
> or
> **LOAD SCREEN-filename(no ext) .nat will be added** *(if <)*
> then
> **list-?, quit-q, default myimage:**

Filenames within DDLab (for all systems) follow the old DOS conventions: eight characters plus a three character extension. Enter the filename without the extention (**.nat**) which is added automatically, or accept the default filename **myimage**. There are futher details on these filing options in chapter 35. Note that a DDLab screen image loaded as above can subsequently be saved and printed as described in section 5.6.2 below for Linux-like systems.

5.6.2 Saving and Printing the screen in Linux-like systems

If **@** is entered at any time when the prompt cursor is flashing, the DDLab screen can be printed to the default printer, or saved as a PostScript (.ps) file. In the DOS version, although this option to print (but not to save) still exists, it only works for some ancient printers, so is not recommended.

In Linux-like versions of DDLab, the following prompt is presented,

> **save/print: without frame -s/p, with frame -S/P, no greyscale +n**
> **then click mouse button in window, quit-q:**

Enter **s** or **S** to just save, or **p** or **P** to save and print, were lowercase selects the DDLab screen without its frame.

Add **n** (i.e. **sn**) to save or print in black only, i.e. suppress greyscale or colors - some lighter colors will not be printed.

On pressing **return** the cursor becomes a diagonal cross-hair ×. Make sure this cursor is within DDLab and click the left mouse button. The DDLab screen image will be saved as a PostScript file called `temp_window_file.ps` (size about 285k) in the DDLab directory.

If printing was selected, the file will be sent to the default printer. While this is happening the message **wait** will appear in a top right window, followed by,

> **done - press any key to continue:**

This reverts the program to the place before printing was selected. Note that the PostScript file `temp_window_file.ps` remains current until overwritten with another print instruction[5].

The following messages might show up in the terminal window,

[5] To convert the PostScript to a PDF file, enter `ps2pdf filename.ps &` in the terminal. The command to print a postscript file from the terminal is (for example) `lpr -Pprintername filename.ps`.

```
pnmtops:  writing color PostScript...
xwdtopnm:  writing PPM file
pnmtops:  warning, image too large for page, rescaling to 0.7 (for example)
```

The last indicates that the size of the DDLab window was too large for the printer paper size, so was automatically rescaled to fit. If the following error message is shown in the top-right window,

```
could not print -- xwdtopnm not found
```

or the following appears in the terminal window,

```
You need to set the environment variable ""XWDTOPNM_PATH""
to the full path of your xwdtopnm to print in greyscale
```

...consult your systems manager.

A more versatile way to print the DDLab window, or a selected part of it, is to run XV (enter `xv &` in the terminal), or another installed screen grabber . XV can grab, save, scale and print the image in one of the many file formats available.

5.6.3 Printing the screen in DOS

In DOS, the option to enter @ to print the DDLab screen is retained, but is largely redundant because it only works for some ancient printers (Epson MX-82 dot matrix, Cannon BJC 4000 bubble jet). The alternative is to use a DOS or Windows compatible screen grabber.

5.7 DDLab version and graphics info

On startup, a top-right window appears similar to this:

Unix/Linux
ddlabm06 Feb09 res=925x694 pt=10 randseed=51717
mouse buttons: left=ret right='q'=backtrack

DOS *(screen resolution is 640×480, 800×600 or 1024×768)*
ddlabm06 Feb09 res=640x480, randseed=24081
mouse found: buttons: left=ret right='q'=backtrack *(if a mouse is detected)*
or
mouse not found: ... *(probably because of a missing DOS mouse driver)*

This shows the following:

ddlabm06 ... the version of DDLab.

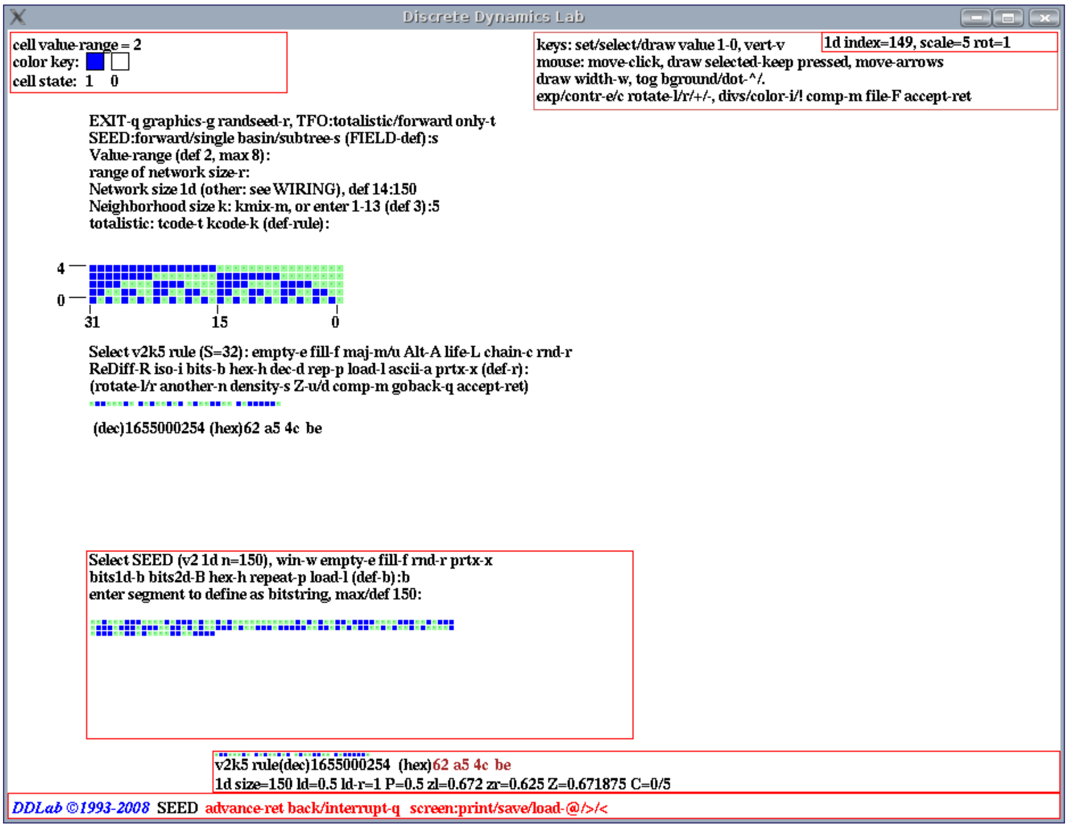
Sept07 ... the release date.

res=925x694 ... DDLab screen resolution, which can be reset.

pt=10 ... the current text point size, which can be reset (also it weight).

randseed=51717 ... the random number "seed" which changes each time DDLab is run, and which can be reset, to exactly repeat default parameters.

mouse buttons ... a reminder that the left mouse button is equivalent to **return**, and the right mouse button is equivalent to **q** for backtracking through prompts.



```
cell value-range = 2
color key: 🟦 □
cell state:  1    0

keys: set/select/draw value 1-0, vert-v
mouse: move-click, draw selected-keep pressed, move-arrows
draw width-w, tog bground/dot-^/.
exp/contr-e/c rotate-l/r/+/-, divs/color-i/! comp-m file-F accept-ret

1d index=149, scale=5 rot=1

EXIT-q graphics-g randseed-r, TFO:totalistic/forward only-t
SEED:forward/single basin/subtree-s (FIELD-def):s
Value-range (def 2, max 8):
range of network size-r:
Network size 1d (other: see WIRING), def 14:150
Neighborhood size k: kmix-m, or enter 1-13 (def 3):5
totalistic: tcode-t kcode-k (def-rule):

Select v2k5 rule (S=32): empty-e fill-f maj-m/u Alt-A life-L chain-c rnd-r
ReDiff-R iso-i bits-b hex-h dec-d rep-p load-l ascii-a prtx-x (def-r):
(rotate-l/r another-n density-s Z-u/d comp-m goback-q accept-ret)

 (dec)1655000254 (hex)62 a5 4c  be

Select SEED (v2 1d n=150), win-w empty-e fill-f rnd-r prtx-x
bits1d-b bits2d-B hex-h repeat-p load-l (def-b):b
enter segment to define as bitstring, max/def 150:

v2k5 rule(dec)1655000254  (hex)62 a5 4c  be
1d size=150 ld=0.5 ld-r=1 P=0.5 zl=0.672 zr=0.625 Z=0.671875 C=0/5

DDLab ©1993-2008  SEED  advance-ret back/interrupt-q  screen:print/save/load-@/>/<
```

Figure 5.2: The main sequence of prompts, for the most commonly applicable 1d CA parameters, are presented down the left side of the screen, other prompts occur in various pop-up windows, for example in the top right hand corner.

In Linux-like systems the default DDLab screen resolution is automatically set to fit comfortably within the monitor screen. The DDLab screen can be resized by dragging with the mouse in the usual way, or set to an exact size (see section 6.3.1). In either case the text size will attempt to resize accordingly.

In DOS the default screen resolution is 640×480 pixels (VGA), but can be set higher within DDLab (section 6.3, or with a command line argument (section 5.3). In Windows, the resolution should be reset to higher (or lower) only within DDLab in full screen mode.

5.8 The mouse cursor in DOS and Windows

In DOS and Windows the mouse cursor behaves as it should in VGA resolution (640×480), and also in higher resolutions (SVGA) in Windows95 and prior. However, in Windows98 and above, at higher resolution only, the mouse is recognised and present (its position can be tracked), but it is not visible. This remains an unresolved problem.

5.9 Memory (DOS only)

DDLab allocates and frees memory as required. In DOS the RAM available is monitored, mainly for diagnostic purposes. A top center "memory window" shows the amount of RAM available in bytes (including extended memory), for example **mem=349049312** which is frequently updated. Other information is also displayed in this window from time to time.

5.9.1 Virtual Memory

skip this section unless your computer is antique

You may create a swap file on disk to augment RAM, using Watcom's "Virtual Memory Manager" (VMM), and use more memory than your computer actually has. When RAM is not sufficient, part of the program is swapped to the disc file until needed. The combination of the swap file and available RAM is the virtual memory.

To specify virtual memory, set the DOS environment variable, "dos4gvm" by entering the following at the DOS prompt:

```
set dos4gvm=[option[#value]] [option[#value]]...
```

or

`set dos4gvm=1` *to accept all defaults*

the VMM options and default values are listed below,

`minmem` ... the minimum amount of RAM managed by VMM, default 512KB.
`maxmem` ... the maximum amount of RAM managed by VMM, default 4MB.
`swapname` ... the swap file name, default `dos4gvm.swp` in the root directory.
`virtualsize` ... the size of the virtual memory space, default 16MB.

For example, typing

```
set dos4gvm=maxmem#8192 virtualsize#32768 swapname#c:\ddlab\ddlab.swp
```

at the DOS prompt before running DDLab gives 32MB of virtual memory in a swap file called `ddlab.swp` in the `ddlab` directory. The virtual memory available will be displayed in the memory window (section 5.9).

5.10 DDLab prompts, and backtrack

On startup, the first of a series of prompts is displayed, described in chapter 6.2. A flashing cursor (usually green) prompts for input. Just enter **return** if in doubt, or the appropriate input from the keyboard. Press **q**, **backspace** (or the right mouse button) to revise, **return** (or the left mouse button) to accept and move on to the next prompt or routine. Just **return** (or left mouse button) automatically selects a default.

To backtrack to the preceding prompt, or up the prompt sequence, or to interrupt a running process such as space-time patterns or attractor basins, enter **q** (or right mouse button).

5.10.1 Prompts: main sequence and pop-up windows

Prompts occur in a main sequence for the most common 1d CA parameters, and also in various pop-up windows for RBN, 2d and 3d networks, analytical measures, data collection, presentation settings, and other special settings,. Note that for 2d and 3d CA, and RBN, the main sequence prompts for the network "wiring" and neighborhood size are superseeded in the first pop-up window, labeled "WIRING".

5.11 Exit DDLab

To exit DDLab immediately (for Linux-like systems) enter **Ctlr-q** at any prompt,. For DOS or Windows, backtrack with **q** to the start of the program. The following central prompt will appear:

EXIT DDLab-q, Restart-ret:

Enter **q** to exit (or **return** to restart). On exiting, the following message should appear in the terminal, or at the DOS or command line prompt: `DDLab clean exit`. If not, a bug report would be appreciated.

Intentionally Blank

Chapter 6

The first prompt in DDLab

Prompts in DDLab are presented in a main sequence down the left side of the screen for the most commonly applicable parameters (including 1d CA); other prompts are presented in various pop-up windows (figure 5.2).

This chapter describes the very first main sequence prompt (figure 5.1) which makes basic choices that set the stage for all subsequent DDLab operations: TFO-mode, SEED-mode, and FIELD-mode (the default). The first prompt also provides various options for graphics, filing, printing and the random number seed from pop-up windows.

6.1 TFO-mode, SEED-mode, FIELD-mode

- TFO-mode: constrains DDLab to run "forwards only" for space-time patterns, at the same time limiting the rules to various types of totalistic, outer-totalistic and reaction diffusion rules. Totalistic lookup tables are much smaller than full look-up tables. All attractor functions, which depend on full lookup tables (and their prompts) are disabled in TFO mode. These constraints reduce memory load, allowing larger neighborhoods (section 7.2). For binary v=2, k_{Lim}=25 instead of 13.. Selecting and deselecting TFO-mode can only be made at this first prompt. TFO-mode requires an initial state or seed, in the same way as SEED-mode (below).

- If DDLab is not constrained in TFO-mode (above), there is a further choice,

  - FIELD-mode: to show the basin of attraction field, which does not require an initial state (the seed).
  - SEED-mode: show anything else which *does* require an initial state or seed: running *forward* for space-time patterns, or generating a *single basin* of attraction, or a *subtree.* The TFO-mode also requires a seed.

  The FIELD/SEED-mode setting can also be swapped when attractor basins are complete (see section 30.4). Note that totalistic rules can also be selected in FIELD and SEED-modes, but in this case their full lookup tables will be applied.

6.2 The first prompt

The first (main sequence) prompt in DDLab is as follows,

> **EXIT-q graphics setup-g randseed-r, TFO:totalistic/forwards onty-t**
> **SEED:forward/single basin/subtree-s (FIELD-def):**

6.2.1 TFO-mode: totalistic forwards only

Enter **t** to select TFO-mode. The first prompt changes:

> **EXIT-q graphics setup-g randseed-r, disable TFO allow basins-b**
> **TFO:totalistic rules and forward only (def:)** *(this line in red)*

Enter **return** to continue in TFO-mode. Enter **b** to deselect TFO-mode, and restore the original prompt in section 6.2.

6.2.2 FIELD, or a run requiring a SEED

If TFO-mode is not selected in section 6.2.1 above, a choice needs to be made between SEED and FIELD-modes.

Enter **return** at the first prompt in section 6.2 for a basin of attraction field. Generating a "basin of attraction field" (the default), does not require a seed because successive basins are seeded automatically, starting with the null state (all 0's).

Enter **s** at the first prompt in section 6.2 if you intend to generate any of the following, which do require a seed,

- *forward* space-time patterns, starting from a seed.
- *single basin* of attraction, containing a seed.
- *subtree* running backwards from a seed.

The seed will will be set at a later stages in the prompt sequence (see chapter 21). Its easy to change between *forward*, *single basin* and *subtree* at later stages.

6.2.3 Notice of the current mode

A reminder of the current mode (TFO, SEED, or FIELD), once set will appear in the bottom title bar, for example,

DDLab ©1993-2009 **FIELD advance-ret back/interrupt-q screen:print/save/load–@/>/<**

6.3 Graphics setup

Enter **g** in section 6.2 or 6.2.1 to change the window size, background color, font size, line spacing or flashing cursor speed. A new graphics screen shows the current font and color palette. A top-right window gives prompts as below. When the graphics changes are complete the program will revert to the first prompt in section 6.2.

Unix/Linux
window size=925x694 resize-S
character width=9 height=15 pointsize=10 resize/weight-s *(for example)*
colors: toggle background-b showold-o showall-a
flashing cursor speed-f quit-q cont-ret:

DOS
screen=640x480 change:1024x768-h 800x600-m *(where h=high m=medium l=low)*
colors: tog background-b
fontsize-s linespace-v
flashing cursor speed-f, quit-q cont-ret:

In Linux-like systems, the initial resolution of the DDLab window is set at 925×694, but can be resized with the mouse in the usual way, or a particular size can be set from these prompts.

In DOS there are three alternative resolutions, given an appropriate monitor and graphics driver.

6.3.1 The DDLab window resolution, Linux-like systems

Enter **S** in section 6.3 to resize the DDLab window to a particular size in pixels.

The following top-right prompt is presented,

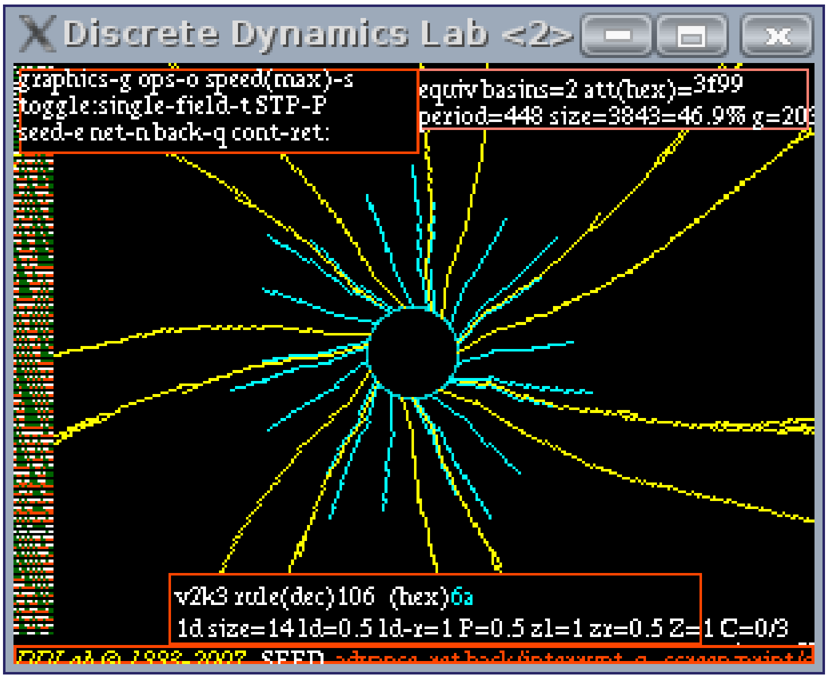
new window size (min 300x250)
width (default 925): *(for example)*
height (default 694):

The smallest size allowed by this method is 300×250 shown in figure 6.1. The window can also be resized to any size with the mouse by dragging a corner in the usual way. Note that when the window is resized the font size is automatically changed to an appropriate size.

6.3.2 The DDLab window resolution in DOS

Sections 5.2.1 and 5.2.2 describe the DDLab window in pure DOS, and in a DOS or command line window in various versions of Windows, which can be toggled to full screen with **Alt** + **Enter**. Once in full screen enter **l**, **m** or **h** in section 6.3 above to change the resolution, where **l**=low:640x480, **m**=medium:800x600 and **h**=high:1024x768. Take care to reset **l** low resolution before toggling back to the DOS or command line window, to avoid bad things!

In pure DOS, the program parameters -m or -h can be entered, i.e. ./ddlabm06 -h for immediate higher resolution.



```
graphics-g ops-o speed(max)-s
toggle:single-field-t STP-P
seed-e net-n back-q cont-ret:
```

```
equiv basins=2 att(hex)=3f99
period=448 size=3843=46.9% g=20
```

```
v2k3 rule(dec)106 (hex)6a
1d size=14 1d=0.5 1d-r=1 P=0.5 zl=1 zr=0.5 Z=1 C=0/3
```

```
DDLab © 1993-2007 SEED
```

Figure 6.1: The smallest DDLab window in Linux, 300×250 pixels, that can be set from the graphics setup prompts. It is really too small to contain prompts and data. This example has the default black background.

6.3.3 White or black background

Select **b** in section 6.3 to toggle between a black and white background. In both cases the new color palette will be shown. A black background may look more elegant but is expensive in ink when printing. References to colors in this manual are based on a white background. Colors on a black background may be different. In DOS, toggling the background works in full screen, not in a DOS window.

6.3.4 The color palette

The graphics screen shows a color palette of the 16 colors (including black and white). In DOS just these colors are used. In UNIX these are the main colors, but more colors are also used, and can be seen by entering **a** (**showall-a**) in section 6.3, reverting to the 16 main colors with **o** (**showold-o**). This feature is intended mainly for program development.

6.4 Changing the font size and line spacing

The default font size and spacing between lines of text can both be changed. The methods differ between Linux-like systems and DOS.

6.4.1 Font size and line spacing, Linux-like systems

If **s** is entered in section 6.3, the following top-right prompts are presented in sequence,

> **set new font size (min 7 max 35, now 10): weight, bold-2 medium-1**
> **set new linespacing (now 15):** *(values shown are examples)*

Enter the new *font size*, followed by *weight*, followed by *new linespacing*, or enter **return** to accept defaults. The new settings will take effect immediately and appear in the Graphics setup prompts (section 6.3 where they can be readjusted.

6.4.2 Font size, DOS

If **s** is entered in section 6.3, a new screen appears showing some text in numbered fonts of various sizes, including a line segment font, and the following prompt,

font 5, to change enter font no: *(values shown are examples)*

Enter the font number corresponding to the new font.

6.4.3 Line spacing, DOS

If **v** is entered in section 6.3, the following top-right prompt is presented,

set new linespace (now 20): *(values shown are examples)*

Enter the new line spacing.

6.5 Changing the flashing cursor speed

The speed at which the cursor flashes depends on the speed of your CPU and other factors. However the flash rate can be altered, enter **f** in section 6.3. The following top-right prompt is presented,

change flashing cursor: restore-r or enter factor:

Enter a factor to slow down or speed up the flash rate. For example to make it half as fast enter .5, twice as fast enter 2. To restore the default setting enter **r**.

6.6 Random number seed

A new random number seed is generated by the computer's "timer" whenever DDLab is started. To change the random number seed, enter **r** at the first prompt in section 6.2. A specific random number seed can be set, or a *random* random number seed. The following prompt appears in a top-right window,

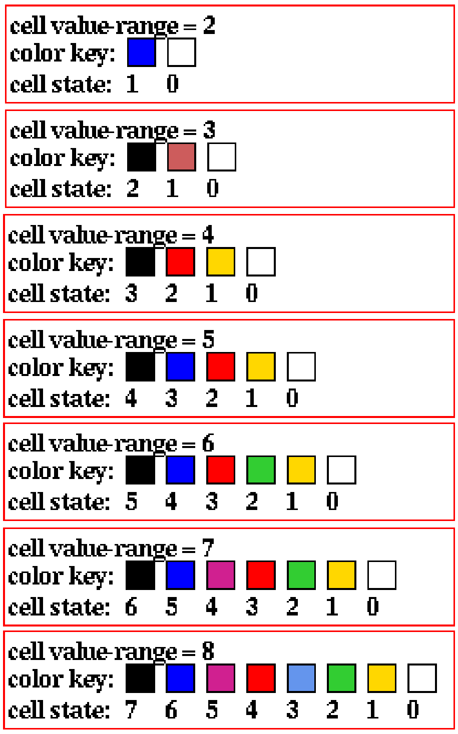
enter random number seed 0-32767 (rnd-def):

Using the same random number seed results in the same sequence of pseudo random numbers being produced by the random number generator. This allows identical multiple runs using default random settings of network parameters such as wiring, neighborhood mix, rules and initial state. Alternatively these setting can be loaded from previously saved files.

Intentionally Blank

Chapter 7

Value-range, v



cell value-range = 2
color key: 🟦 ⬜
cell state: 1 0

cell value-range = 3
color key: ⬛ 🟫 ⬜
cell state: 2 1 0

cell value-range = 4
color key: ⬛ 🟥 🟨 ⬜
cell state: 3 2 1 0

cell value-range = 5
color key: ⬛ 🟦 🟥 🟨 ⬜
cell state: 4 3 2 1 0

cell value-range = 6
color key: ⬛ 🟦 🟥 🟩 🟨 ⬜
cell state: 5 4 3 2 1 0

cell value-range = 7
color key: ⬛ 🟦 🟪 🟥 🟩 🟨 ⬜
cell state: 6 5 4 3 2 1 0

cell value-range = 8
color key: ⬛ 🟦 🟪 🟥 🟦 🟩 🟨 ⬜
cell state: 7 6 5 4 3 2 1 0

Figure 7.1: The cell value color key window for value-range v = 2 to 8, for a white background. The window appears when the value-range is selected, and also at other times. Colors will be different on a black background (figure 2.1). The values themselves are indexed from v-1 to 0.

This chapter describes how to set the "value-range", v, available to a cell or network element, which can also be thought of as the number of its internal states, colours, or the size of its "alphabet". Older versions of DDLab were limited to binary (or Boolean) networks, where v=2.

Note that setting a high value-range, v, limits the maximum neighbourhood size, max-k, (section 7.2) and also, for basin of attraction fields, the maximum network size, n (section 7.3).

7.1 The value-range prompt

The value-range prompt, the second prompt in the main sequence, is as follows,

Value-range v (def 2, max 8): *(on startup, or the default becomes the last v set)*

Enter a new value-range (which becomes the new default) or enter **return** to accept the default. The selection of the value-range can only be made at this early stage in the program.

7.2 Value-range, limitations on neighborhood size, k_{Lim}

The upper limit of neighborhood size k_{Lim} decreases with increasing value-range v because look-up tables may become too large. A full look-up table v^k is larger than a k-totalistic look-up table $(v+k-1)!/(k!(v-1)!)$, so this also depends on whether DDLab was constrained to run "forwards-only" for totalistic rules (TFO-mode). k_{Lim} for increasing value-range v, for both cases, are given in the tables below, and these limits are included in the prompts. The tables also show the size of the corresponding rule-tables S.

Note that k_{Lim} is different from k_{max} which is a deliberate selection of the maximun k in a k-mix, a network with mixed neighborhood sizes. $k_{max} \leq k_{Lim}$.

Unconstrained

v	k_{Lim}	table size S
2	13	8162
3	9	19683
4	7	16484
5	6	15629
6	5	16807
7	5	16807
8	4	4096

TFO-mode

v	k_{Lim}	table size S
2	25	26
3	25	351
4	25	3276
5	25	23551
6	17	26334
7	13	27132
8	11	31824

Table 7.1: The upper limit of k, k_{Lim}, for different value-ranges v. *Left:* FIELD-mode or SEED-mode with a full, unconstrained, rule-tables, capable of computing basins of attraction. *Right:* TFO-mode, for running totalistic rules, which have smaller rule-tables, forwards only, so the k_{Lim} can be larger. In both cases the sizes S of the rule-tables for v and k_{Lim} are listed.

7.3 Value-range, limitations on network size, n_{Lim}

For basin of attraction fields (FIELD-mode) the upper limits[1] of network size n_{Lim} decrease with the increasing value-range v as set out in table 7.2, and these limits are included in the prompts. In practice much smaller sizes are advisable because of probable time/memory constraints.

v	n_{Lim}	state-space v^n ≈ billions
2	31	2147483647 ≈ 2.15
3	20	3486784400 ≈ 3.49
4	15	1073741824 ≈ 1.07
5	13	1220703124 ≈ 1.22
6	12	2176782335 ≈ 2.18
7	11	1977326742 ≈ 1.98
8	10	1073741823 ≈ 1.07

Table 7.2: For a basin of attraction field (FIELD-mode) the maximum network size, n_{Lim}, for different value-ranges v, and the corresponding maximum state-space v^n.

[1] These limits are required so that the size of state-space v^n is smaller than an unsigned long interger ($2^{32} = 4294967296$), a number used to index bits in a char array that identifies "used" states.

Chapter 8

1d network size n, or range-n

This chapter describes setting the 1d network size n, or a range of sizes. Setting the size of 2d or 3d networks, as well as other special options, is done at a later stage (section 11.6) in which case any 1d size will be superceeded and can be ignored by entering **return**.

1d n is set here in the main sequence of prompts (as is k in section 9) to simplify setting up CA attractor basins, which are usually 1d, and to run 1d CA forward for space-time pattrens.

The chapter also looks at the upper limits of n possible in DDLab, and the practical computational and speed limitations of combinations of v, k and n for attractor basins, with examples.

8.1 Setting range of sizes, 1d

It is possible to generate a series of basin of attraction fields for a range of 1d network sizes (figure 8.4), with the network size printed on the left of the screen. The presentation is similar to the "Atlas" in "The Global Dynamics of Cellular Automata" [19]. Single basins or subtrees for a range of network sizes may also be generated.

If not in TFO mode, the following main sequence prompt is presented,

range of network size-r, else-ret:

If **r** is selected the next two prompts are presented in sequence as follows, where the defaults for start and end depend on the value-range v,

range size: start (def 5): end (def 12): *(for FIELD-mode, v=2)*

Enter the start and end sizes, or enter **return** to accept the defaults, taking care to avoid excessively large sizes (see section 8.3 below).

8.2 Setting the size of one network, 1d

If **r** was not selected in section 8.1 above, the 1d network size is set with one of the following main sequence prompts, depending on the mode selected at the first prompt (section 6.2 FIELD, SEED, or TBO, and where default n depends on the value-range v,

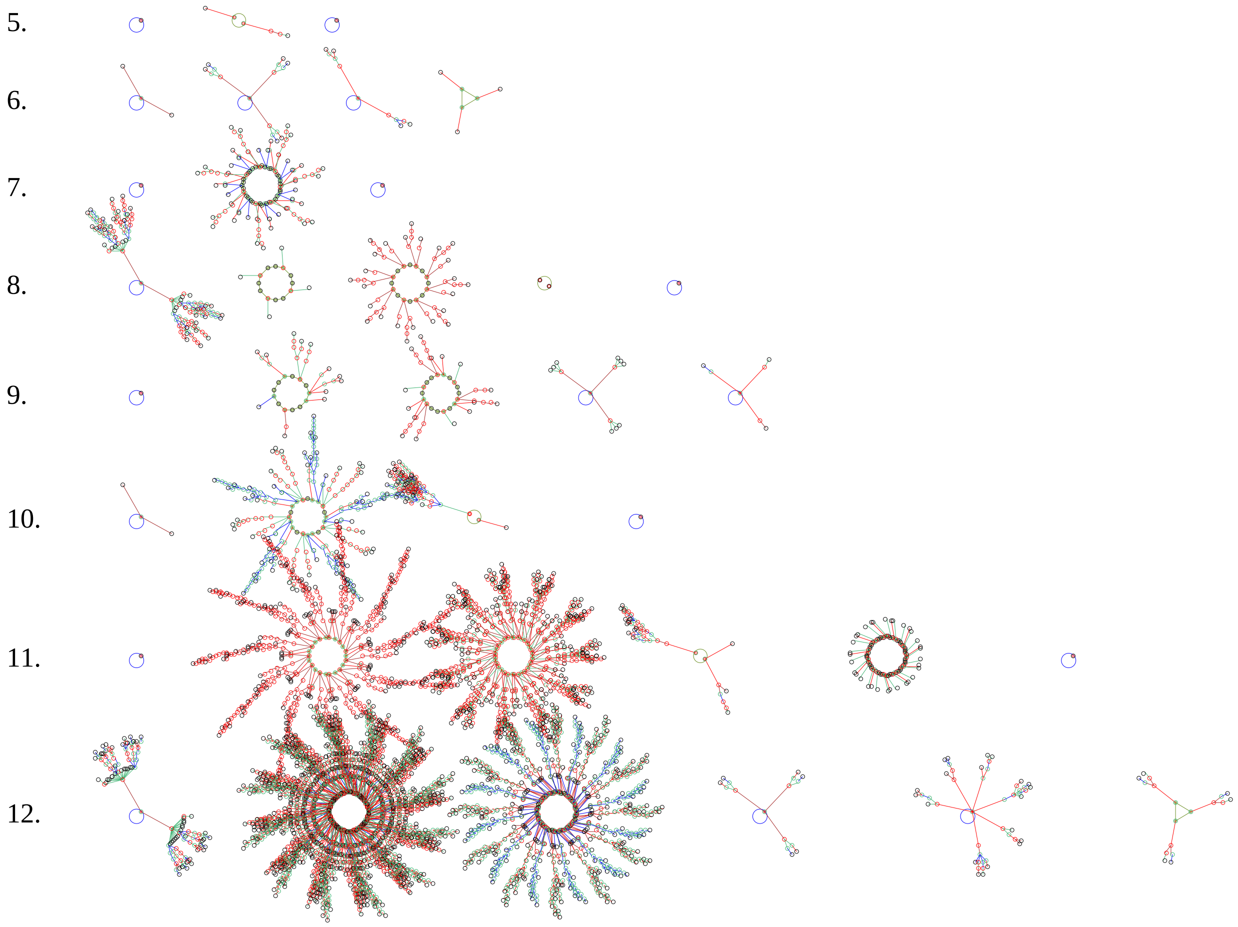


Figure 8.1: Basin of attraction fields for a range of network size 5-12.v=2, $k = 4$, rule(hex)=61a4.

for FIELD mode
Network size n, max 31, default 10: *(for v=2, max n depends on v, section 7.3)*
for SEED mode
Network size n, 1d (other:see WIRING), def 14:
for TFO mode)
Network size n, 1d (other:see WIRING), def 150: *(for TFO mode)*

Enter the required 1d network size, n, For 2d or 3d networks, the network size will be set as part of the wiring options (chapters 11 - 12), so these prompts in sections 8.1- 8.2 may be ignored by pressing return.

The network size n cannot be greater than n_{Lim} (section 8.3). If a larger n is selected, the default n is appled with the following message,

... **:33 too big! n=10** *(in FIELD-mode, for example.* n_{Lim} *is listed in table 7.2)*
... **:66000 too big! limit=65025! n=14** *(in SEED-mode or TFO-mode, for example)*

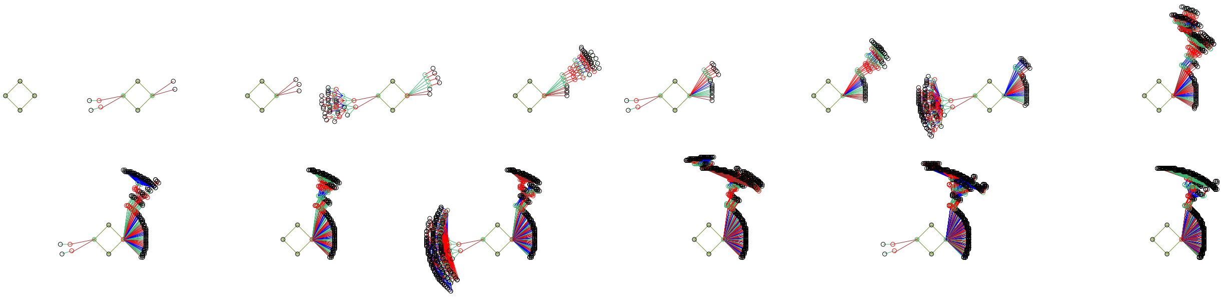


Figure 8.2: Single basins for a range of network sizes containing the state all 0s, n=1 to 15. v=4, k=2, rule(hex)=a7857c0d.

8.3 Network size limitations

Network size limitation are also discussed in sections 1.6 and 7.3. To summarize, the upper limit of network size, n_{Lim}, supported by DDLab is as follows,

FIELD-mode

Basin of attraction fields: n_{Lim} varies according to value-range v (table 7.2) between $v = 2$: n_{Lim}=31 (figure 8.6) and $v = 8$: n_{Lim}=10. In practice n should be well below these maximum limits.

SEED-mode and TFO-mode

Space-time patterns (1d, 2d or 3d): n_{Lim}=65025, which corresponds to the maximum size of a square 2d network, 255×255, and can be represented by an unsigned short (16 bits). The maximum size of a 3d cube is 40×40×40. The networks need not be square or cubic as long as $i \times j \leq n_{Lim}$, or $i \times j \times h \leq n_{max}$,
For 1d networks shown in 1d, a much smaller size is preferable, say n=150 to 200, to fit the screen and for a clear view of analytical graphic windows. However, 1d networks can be shown in 2d or 3d, allowing a better view of analytical graphic windows.

Sub-trees: n_{Lim}=65025. In practice the size should be much smaller, say $n \leq 50$. For a practical size try n=25.

Single basins: n_{Lim}=65025. In practice the size should be smaller than for sub-trees, say $n \leq 25$. For a practical size, try n=18.

8.3.1 Computational and speed limitations for attractor basins

In general, networks with large network size n may be run "forwards" to generate space-time patterns, and also find attractors for rules with low Z-parameter[27], but only networks with modest n can be run "backwards" to generate attractor basins. Of these, a subtree allows the largest networks, and basin of attraction fields the smallest. If in doubt, small sizes should be tried first. Larger sizes might exhaust computer memory or take an excessive length of time to compute, especially for networks other than regular 1d CA.

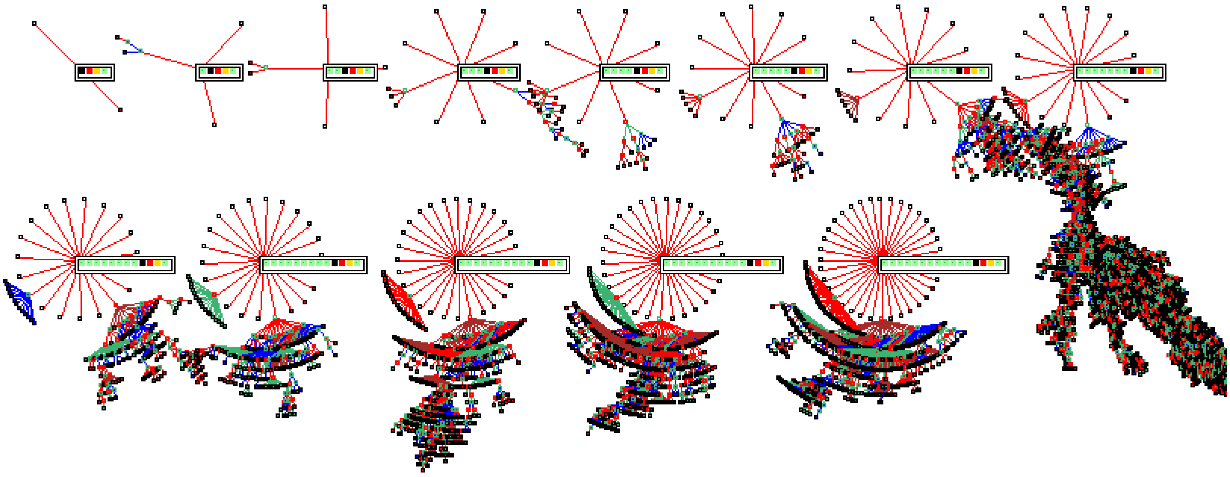


Figure 8.3: Subtrees for a range of network sizes from a root state containing from the string 3210, which is highlighted, n=4 to 15, v=4, k=3, rule(hex)=1c49a5b05dbcdd148377635fb0b60d84.

n is not the only criterion effecting computational and speed limitations for attractor basins. The complexity, thus speed, of computation for running backwards to find predecessors, and generate basin of attraction fields, single basins or subtrees, depends on a combination of v, k and n. It also depends on the reverse algorithm, which is different and slower for random (non-local) wiring; if wiring is local 1d, the faster algorithm is applied, even if rules are mixed.

The speed also depends on the quality of the rule itself - how "branchy" are the resulting subtrees, the "in-degree". Typical in-degrees (number of predecessors of states) are predicted by a rule's Z parameter[27] which varies between 0 and 1, or the average Z across the network for RBN and DDN. A rule with low Z will have high "in-degree" (characteristic of "order"), thus short transients; excessive in-degree can overwhelm computer resources. On the other hand, a rule with high Z will have low in-degree (characteristic of disorder or "chaos"), allowing subtrees for large n to be computed, but transients and attractor periods tend to be extremely long, making basins of attraction difficult to reconstruct. The subset of chain-rules (section 16.11) have the lowest in-degree, usually just one, which does not increase with n; 1d CA chain rules can therefore be run backwards for extremely large n, as in figure 8.4.

Finally, the speed will of course depend on the computer itself and what other programs are running.

8.4 Times for basin of attraction fields

The elapsed times to generate basin of attraction fields for a range of v, k and n, for CA, RBN and DDN, on a 1.66GHz Laptop running Linux/Ubuntu 6.06[1] are shown in tables 8.1- 8.4. below. These times are displayed as part of the data in a top-right window when the field is complete (section 27.5).

The system size n that is appropriate for generating attractor basins in a reasonable time for combinations of v and k can be inferred from these examples. Large sizes may be tried, but

[1] Lenovo 3000 N100 Laptop, Duo processor T2300e 1.66GHz, 1024 Mb RAM, running Linux Ubuntu 6.06.

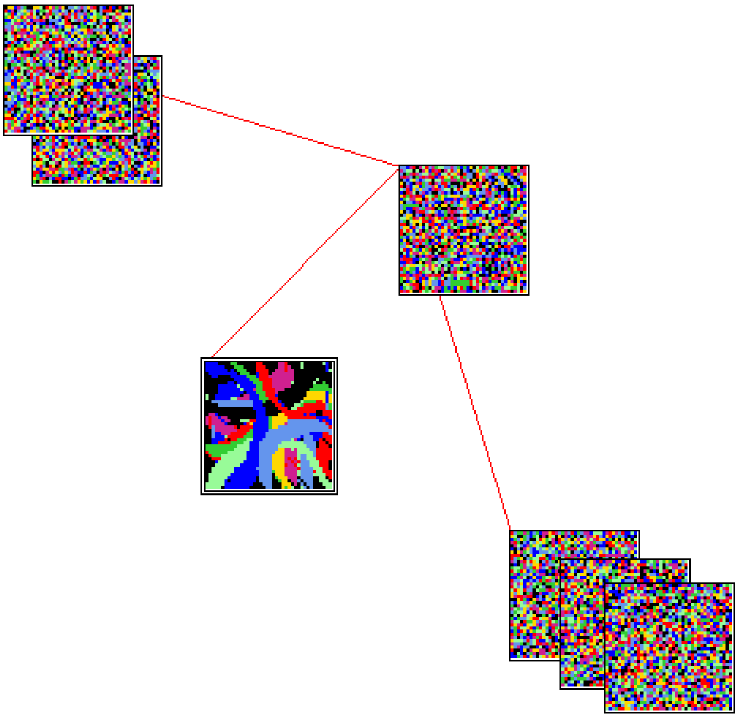


Figure 8.4: A subtree for a 1d CA chain-rule for a large network. n=1600 shown as 40×40. v=8, k=4. The root state of the subtree is located left of centre.

may impose unacceptable time, memory and display constraints. Single basins, and especially sub-trees, may be generated for relatively much larger systems, especially for chain-rules (section 16.11, or close mutants of chain-rules, which have low in-degree.

8.4.1 Examples for 1d CA

Examples of the time needed to generate the basin of attraction field of some 1d CA rules for a range of v, k and n, are shown in tables 8.1 and 8.2. The rules are specified in hex for v=2, and as the rule file names for $v \geq 3$.

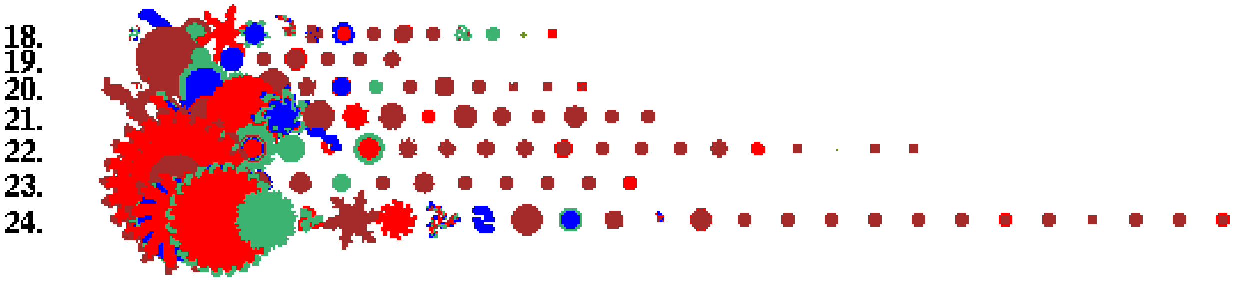


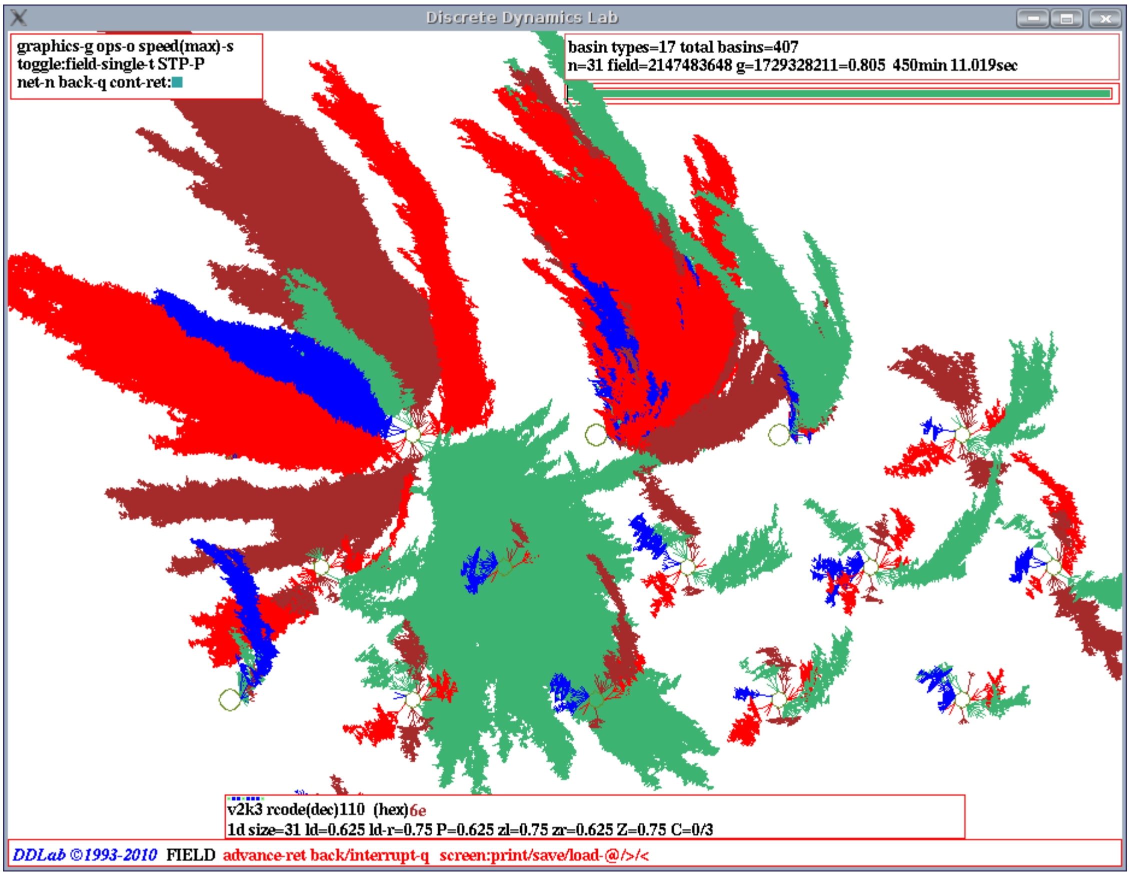
Figure 8.5: Basin of attraction fields for a range of network sizes, v=2, k=5, n=18 to 24, to assemble part of the data for table 8.1 below.

n	18	19	20	21	22	23	24	k and rule (hex)
time	2.2s	4.6s	9.4s	19.4s	40.2s	1m 23s	2m 52s	k=3 rule c1 (dec) 193
	3.8s	6.8s	13.9s	28.8s	58.0s	1m 59s	4m 10s	k=4 rule e924
	9s	10s	20.9s	41.8s	1m 26s	2m 55s	6m 0s	k=5 rule b755d3d9

Table 8.1: Times for increasing k and n for v=2, basin of attraction fields, 1d CA.

n	12	13	14	15	16	17	18	v and rule file
time	7.2s	18.5s	59s	3m 0s	84m 54s	27m 55s	100m	v=3 `s_v3k3.rul`
	6m 56s	20m 37s	109m					v=4 `s_v4k3.rul`
	132m 34ss							v=5 `s_v5k3.rul`

Table 8.2: Times for increasing v and n for k=3, basin of attraction fields, 1d CA.


Discrete Dynamics Lab
graphics-g ops-o speed(max)-s
toggle:field-single-t STP-P
net-n back-q cont-ret:

basin types=17 total basins=407
n=31 field=2147483648 g=1729328211=0.805 450min 11.019sec

v2k3 rcode(dec)110 (hex)6e
1d size=31 ld=0.625 ld-r=0.75 P=0.625 zl=0.75 zr=0.625 Z=0.75 C=0/3

DDLab ©1993-2010 FIELD advance-ret back/interrupt-q screen:print/save/load-@/>/<

Figure 8.6: The basin of attraction field of $v2k3$ rcode (dec)110, for n=31, which is the n_{Lim} for v=2, elapsed times to generate 7.5 hours. The field is shown on the DDLab screen, with compression on (section 26.1), equivalent trees and subtrees suppressed (section 26.1.3), and nodes suppressed. (section 26.2).

8.4.2 Examples for RBN and DDN

A similar exercise for RBN and DDN gave the timings shown in tables 8.3 8.4.

The same rules were used as for the CA in section 8.4.1, shown in hex for v=2, and as rule files for $v \geq 3$. However, the random wiring was not recorded. Note that it is the random wiring that requires a different and slower reverse algorithm, not the inhomogeneity of rules, so having one rule still provides a good indication of the speed, and allows a meaningful comparison between the various examples.

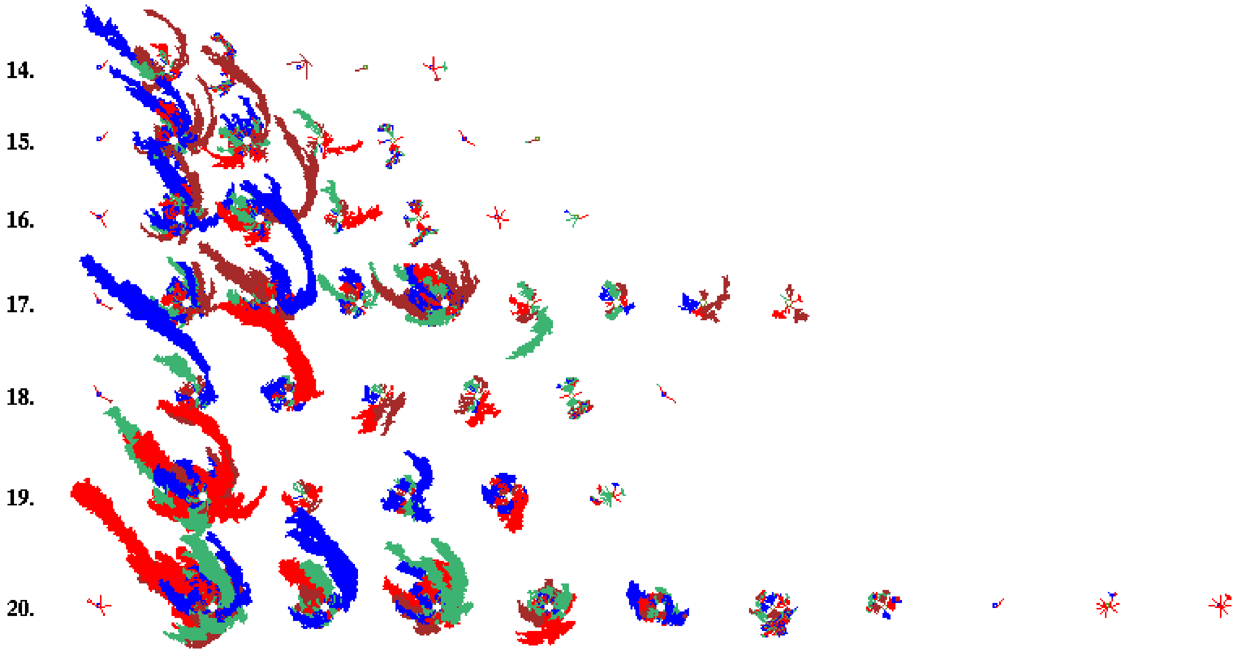


Figure 8.7: Basin of attraction fields for a range of network sizes, RBN, $v2k3$=2, n=14 to 20, to assemble part of the data for table 8.3 below.

n	14	15	16	17	18	19	20	k and rule (hex)
time	0.81s	4.4s	8.4s	24.6s	39.2s	1m50s	4m54ss	k=3 rule c1 (dec: 193)
	10.7s	25.7s	1m53s	4m19s	12m31s	26m46s	64m40s	k=4 rule e924
	32.3s	2m50s	9m15s	15m24s	75m28s	143m	195m	k=5 rule b755d3d9

Table 8.3: Times for increasing k and n for v=2, basin of attraction fields, RBN.

n	10	11	12	13	14	v and rule file
time	8.0s	2m5s	9m20s	28m7s	152m	v=3 `s_v3k3.rul`
	9m2s	59m43s	268m			v=4 `s_v4k3.rul`
	223m					v=5 `s_v5k3.rul`

Table 8.4: Times for increasing v and n for k=3, basin of attraction fields, DDN.

Intentionally Blank

Chapter 9

Neighborhood, k, or mixed k

This chapter describes setting a homogenious 1d CA neighborhood k with local wiring (nearest neighbour, next nearest etc.), or a k-mix, a mix of k sizes, with the exact k-mix to be specified in a top-right window. Randomizing this wiring (making it non-local), 2d or 3d, and other special wiring options come at a later stage (described in chapters 11 - 12) in which case any 1d k entries at this stage will be superceeded and can be ignored by entering **return**.

1d k is set here in the main sequence of prompts (as is n in section 8) to simplify setting up CA attractor basins, which are usually 1d, and to run 1d CA forward for space-time pattrens.

9.1 Selecting k, or a k-mix, for 1d networks

To set the neighborhood size k, or a k-mix for 1d networks, the following main line prompt is presented,

> **Neighborhood size k: kmix-m, or enter 1-13 (def 3):** *(k_{Lim}=13 depends on TFO-mode and v as listed in section 7.2. Default k=3 changes according to a previous selection)*

Enter the required value of k, or **m** for a k-mix - The actual mix is set in section 9.4 below. For 2d and 3d networks, both the mix and size will be reset in the wiring options (chapters 11 - 12), so the main sequence prompt can be ignored by entering **return**. The k set becomes the new default in this option.

Once the network's rule or rules have been set (chapters 13-16), the network's k or k-mix may be “neutrally” modified, both by increasing k or reducing to effective k (sections 18.7 - 18.9), for the network as a whole or for a particular cell.

9.2 k_{Lim} and minimum k

The maximum number of input wires to a cell, k_{Lim}, currently supported in DDLab depends on TFO-mode and value-range v as listed in section 7.2. For v=2, in TFO-mode k_{Lim}=25, and 13 otherwise. Minimum k=1. k=0 is illegal as it would require an array size of zero, but a cell in a k-mix network can be set to effective k=0 as described in section 9.3 below.

Note that k_{Lim} is different from k_{max} which is a deliberate selection of the maximun k in a k-mix. $k_{max} \leq k_{Lim}$.

9.3 Effective k=0

Setting k=0 directly is illegal in DDLab. For effective k=0 you need single self-wiring and a neutral rule that conserves a cell's value. Set k=1 with the cell wired to itself (local wiring in 1d, 2d or 3d), and set its rule-table as follows,

v	rule-table
2	10
3	210
4	3210
5	43210
6	543210
7	6543210
8	76543210

Table 9.1: Effective k=0 neutral rules – the same for all rule types. v=2 to 8, for single self-wiring, where k=1 is wired to itself.

Because k=1 these rule-tables are exactly the same for all rule types, rcode, kcode and tcode. Setting a neutral rule to a self-wired cell ensures that the cell is not recieving any inputs from other cells, and that it will conserve its current value. However, other cells may recieve inputs from the effective k=0 cell. DDLab can identify any self-wired k=1 cells, and gives an option to set the rule at these cells to make them effectivly k=0 automatically (see section 14.3), or this can be done by hand - the easiest way is to "kill a cell" from the 1d wiring graphic (section 17.8.9).

Setting effective k=0 is usefull for providing a fixed input signal to a network, or potentially for a arbitrary external input.

9.4 Specifying the k-mix

If **m** is selected in section 9.1 above, the following prompt appears in a top-right window,

> **set k-mix: load-l hand-h norm-n power/specify-s rnd-(def):**
> *(**norm-n** only if $n \geq k_{Lim}$, i.e. not for very small networks)*

The k-mix can be set in a variety of ways described below. Note that settings for particular cells in a k-mix can be changed later in the "wiring graphic" options (see chapter 17).

9.5 Loading a k-mix file

Enter **l** in section 9.4 above to load the k-mix as a `.mix` file (see Filing, chapter 35). The file would have been saved in sections 19.5 or 9.12.3. If the file is for a bigger network, as much k-mix data as will fit into the current network will be loaded, starting from cell index 0. If on the other hand the file is for a smaller network, all the data will be loaded from cell index 0, the excess cell indexes will be allocated a uniform background k-mix with the following prompt,

> **k-mix loaded (10) less than net size (150)** *(values shown are examples)*
> **enter background k-mix 1-13, (default 5):**

The k-mix encoding is described in section 19.3.1. The k-mix can be also be save/loaded in section 19.1 for network architecture, (see also section 17.8.12).

9.6 Setting the k-mix by hand

If **h** is selected in section 9.4, a series of prompts allow the k at each cell index (starting from 0) to be set.

> **set neighborhood size (1-13) at cell 0 (back-b restrnd-r):**

Enter the required k, or **return** for a random k value between 1 and k_{Lim}. **b** allows backtracking to the previous cell index. Enter **b** to assign k at random for all remaining cells.

9.7 Setting a normal distribution

Enter **n** to set a normal (Poisson) distribution of the frequencies of different k. This option does not apply for very small networks, when $n < k_{Lim}$. The distribution is achieved by assigning potential links at random to the network, up to a preset limit. However, every cell will be assigned at least one link. The following top-right prompts are presented in sequence,

> **set k upper bounds (def 13, limit 13):**
> **set average k (def 2.60, max 8.67):** *(values depend on k upper bounds)*

Enter the upper bounds of k for the distribution, followed by the target average k in the network. An example of a normal distribution is shown in figure 9.1 generated in section 9.12.

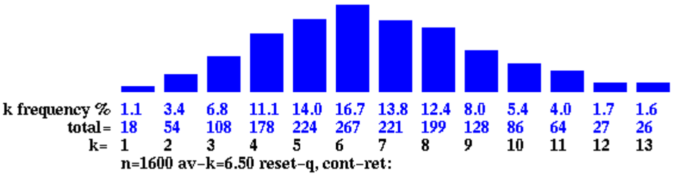


Figure 9.1: An example of a normal (Poisson) distribution of the frequencies of different k. n=40×40, the range of k=13 and the average k=6.5

9.8 Specify each k, or power-law distribution

Enter **s** in section 9.4, to specify the distribution of different k, or to select a power distribution. The following top-right prompt is presented,

> **power-law-p, specify-(def):**

These options are explained in sections 9.8.1 to 9.8.3 below.

9.8.1 Specify the percentage of different k

If **return** is entered in section 9.8 above, the percentage of cells with different k can be specified, and within those constraints the k-mix is assigned at random (i.e. shuffled). Alternativly, the k-mix need not be shuffled in which case it is assigned to the network in continuous blocks.

The following series of prompts are presented for each k starting with k=1 (shown with example settings),

enter % k=1 (100.0% left):22 *(for 22% k=1)*
enter % k=2 (78.0% left) back-b:25 *(78.0% of the network remains)*
enter % k=3 (53.0% left) back-b:33 *(53.0% of the network remains)*
...etc.

The prompts continue until none of the network remains, or until $k = k_{Lim}$. **Return** assigns zero of that particular k. An assignment that is too large for the network still remaining will be truncated. If on completion some of the network remains, this will be set with the last k that was assigned. If none of the k's are assigned, the prompt reverts to section 9.4. At any stage from $k = 2$ onwards it is possible to backtrack and revise with **b**.

9.8.2 Setting a power-law distribution of k

Enter **p** at the prompt in section 9.8 to set a power-law distribution of the frequencies of different k. However, every cell will be assigned at least one input. The following top-right prompts are presented in sequence,

set k upper bounds (def 13, limit 13): *(if k_{limi}=13)*
enter power-law exponent 1 to 3 (def 2.0):

Enter the upper bounds $k_{max} \leq k_{Lim}$ for the distribution, followed by the taget power-law exponent. An example of a power-law distribution is shown in figure 9.2. The output distribution can also be set to approximate a power-law (see section 17.8.5 and figure 17.18).

If both the inputs, k, and the outputs, follow a power-law, the network is said to be "scale-free", and characteristic of many natural and artificial networks, from metabolic networks [2] to the word-wide-web. The graph of a "scale-free" network is shown in figure 20.2.

9.8.3 Showing the distribution

When the k-distribution has been assigned, either by specifying the percentage of different k in section 9.8.1, or by setting a power-law in section 9.8.2, data on the distribution, and the following prompt, are presented in a top-right window (for example, for a 2d 40×40 network, and a power-law),

k1=63.2%=1011 k2=15.6%=245 k3=7.15%=114 k4=4.0%=64 k5=2.6%=41
k6=1.8%=29 k7=1.4%=22 k8=1.1%=17 k9=0.8%=13 k10=0.7%=11
k11=0.6%=9 k12=0.5%=8 k13=0.4%=7
shuffle: no-n yes-(def):

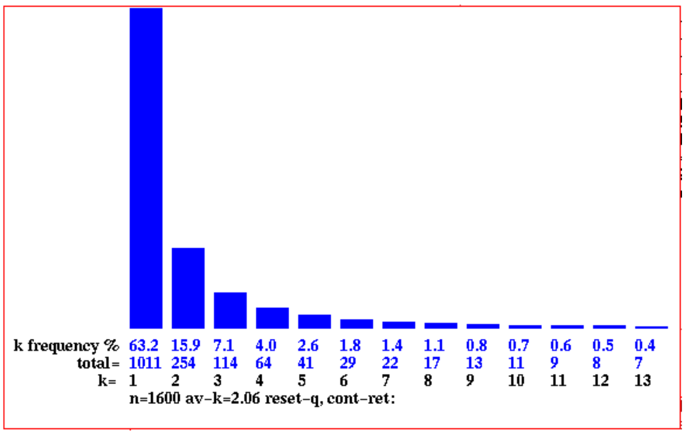


Figure 9.2: An example of a power-law distribution of the frequencies of different k. In this case n=40×40, the range of k=13, the power-law exponent is 2, and the average k=2.05.

This shows the percentage and number of different k's in the network. The k's are initially assigned in continuous blocks starting from the network cell 0. Enter **return** to randomly shuffle the assignment, or **n** *not* to shuffle and retain the blocks.

The distribution can be displayed as a histogram as in figures 9.1 and 9.2 generated in section 9.12 (and later in section 17.8.13).

9.9 Setting the k-mix at random

If **return** was entered in section 9.4, the default, a random k-mix is selected. A further prompt allows the k-mix be confined within upper and lower bounds before the random mix is assigned. In this example, k_{Lim}=13 (section 7.2),

set k bounds 1-13: lower (def 1): upper (def 9, limit 13): *(for example)*

The upper bound is refered to as k_{max}. The different k's will be assigned with equal probability.

9.10 Setting a k-mix with uniform k

Its sometimes useful to set up a k-mix network with homogeneous k, for example for a regular 1d, 2d or 3d CA into which a DDN can be inserted (see section 19.4), for example to provide a source of noise in the regular network.

A k-mix with homogeneous k is easily created in section 9.8.1, by setting the percentage of the required k at 100%, or in section 9.9 by setting equal values for the lower and upper bounds in a "random" the k-mix. The homogeneous k network is treated as a k-mix network, and as such is also treated as a having non-local wiring and mixed rules even if the wiring is set as local 1d (see section 12.4.1), and there is just one rule (see section 14.4.3). These issues relate to how memory is allocated to different types of network.

9.11 Increasing k_{max}

Options for changing k for individual cells (or blocks of cells) occur at later stages in DDLab (see sections 17.8.8 and 18.7).

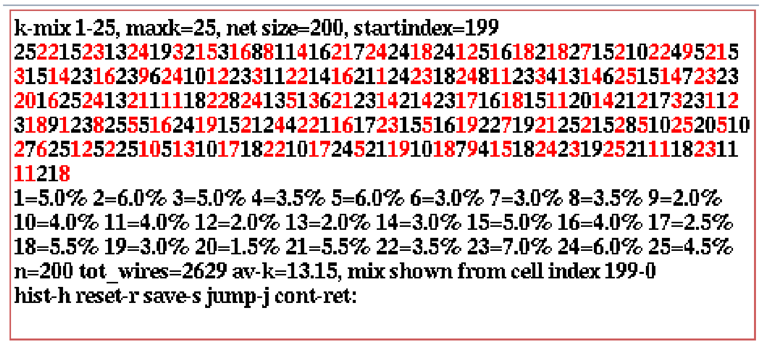
The default k_{max} is the upper bound set previously, for example in section 9.9. However k_{max} can be set to a higher value, up to the upper k_{Lim} listed in section 7.2. The following top-right prompt is presented,

set greater max-k (max 25, def 9): *(if k_{Lim}=25 and k_{max}=9)*

This enables the k values in the network to be subsequenty increased up the new k_{max}. For example, if 100% k=5 is set in section 9.8.1 (a k-mix with uniform k), and k_{max} is set to say 9, cells in the uniform k network may later be reset to any value ≤ 9.

9.12 Reviewing the k-mix

Finally, the k-mix is displayed in a top-right window, or a succession of windows for large networks. For example, figure 9.3 shows a randomly assigned k-mix between 1 and 25 for a network size 200 (in TBO mode), for each cell 199 to 0. To distinguish each (decimal) number, colors alternate between black and red. The percentage and number of different k's in the network is also displayed.



```
k-mix 1-25, maxk=25, net size=200, startindex=199
25221523132419321531688114162172424182412516182182715210224952215
31514231623962410122331122141621124231824811233413146251514723 23
20162524132111118228241351362123142142317161815112014212173231 12
31891238255516241915212442211617231551619227192125215285102520510
27625125225105131017182210172452119101879415182423192521111823 11
11218
1=5.0% 2=6.0% 3=5.0% 4=3.5% 5=6.0% 6=3.0% 7=3.0% 8=3.5% 9=2.0%
10=4.0% 11=4.0% 12=2.0% 13=2.0% 14=3.0% 15=5.0% 16=4.0% 17=2.5%
18=5.5% 19=3.0% 20=1.5% 21=5.5% 22=3.5% 23=7.0% 24=6.0% 25=4.5%
n=200 tot_wires=2629 av-k=13.15, mix shown from cell index 199-0
hist-h reset-r save-s jump-j cont-ret:
```

Figure 9.3: An example k-mix randomly assigned for k=1-25 (TBO mode), for a network size=200. To distinguish each (decimal) number, colors alternate between black and red.

Options are presented at the foot of the k-mix review window,

...
mix shown from cell index 244-0 *(values shown are examples)*
hist-h reset-r save-s jump-j cont-ret:

Enter **h** to show the k distribution histogram in a lower left window, as in figures 9.1 and 9.2. Enter **r** to revert to section 9.4 and reset the k-mix. Enter **s** to save the k-mix in a `.mix` file (see Filing, chapter 35) which can be loaded as in section 9.5. Enter **return** to accept the k-mix. The other options are explained below.

9.12.1 Reviewing the k-mix in large networks

Large networks may require several windows to display the k-mix. In this case the prompt above (section 9.12) includes an option **more-m** to see successive windows (for example),

> **...**
> **mix shown from cell index 7011-4221** *(values shown are examples)*
> **hist-h reset-r save-s more-m jump-j cont-ret:**

9.12.2 Jumping to a new cell index

Alternatively, enter **j** in section 9.12.1 to jump to any cell index in the network, which becomes the first index in the new window. The following prompt is displayed,

> **...**
> **jump to index (9800-0):** *(this example for a 2d network, 99×99)*

9.12.3 Saving the k-mix file

Enter `s` in 9.12 to save the k-mix. A filing prompt box will allow saving a `.mix` file – the default filename is `mymix.mix` (see Filing, chapter 35). The k-mix can also be saved in section 19.5. but can only be loaded in section 9.5. However, information on the k-mix in mixed k networks is contained in the `.w_s`, `.r_s` and `.wrs` files (section 19.2), so does not usually need to be saved separately.

9.13 Reviewing the k-mix within "network architecture"

The k-mix may also be reviewed as described in chapter 17 "Reviewing network architecture". Options in chapter 17 allow changing k, and "neutral" k changes, up to k_{max} set in section 9.11, which may be greater than the maximum k found in the network. This can be done for single cells, for predefined blocks, or for the whole network.

Intentionally Blank

Chapter 10

The local neighborhood, and network geometry

This chapter defines the CA neighborhood, the pseudo-neighborhood for RBN or DDN, the network geometry, and how these are indexed in DDLab.

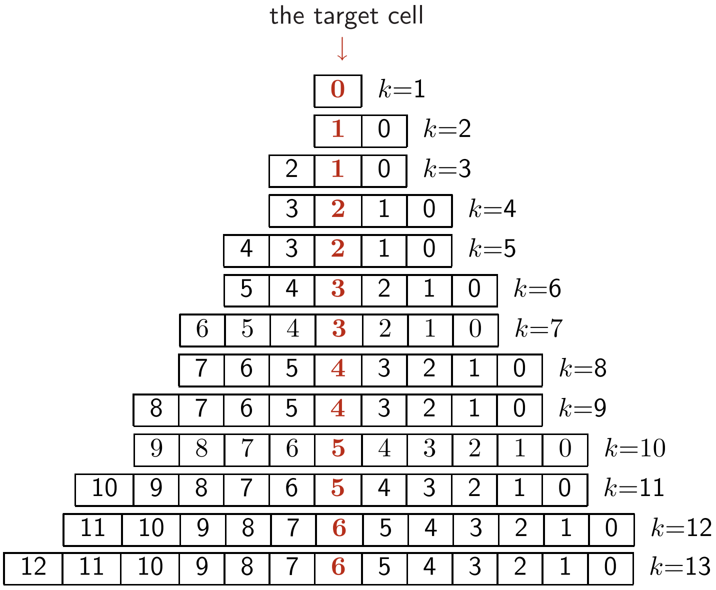
10.1 The CA neighborhood

For CA, a cell updates according to the values of it local neighborhood, which usually includes the cell itself and its nearest neighbors. The relative position of the neighborhood cells in relation to the "target" cell is referred to as the "neighborhood template", or just the "neighborhood". For CA the neighborhood is homogeneous throughout the network, and thus requires periodic boundary conditions where each array edge wraps around to its opposite edge resulting in a ring of cells in 1d, the surface of a torus in 2d, and a 3-torus in 3d. For 1d and 2d the neighborhood templates are defined in DDLab for k=1 to 25 ($k \geq 14$ TFO-mode only), and k=1 to 13 for 3d.

The following sections describe the predefined neighborhoods and how they are indexed. Note that in 2d these neighborhood templates can be square or hexagonal (sometimes both) to run on a square or hexagonal lattices, as shown in figure 10.3. Its also possible to construct any arbitrary neighborhood assign it uniformly throught the network (section 12.5.11).

10.2 The pseudo-neighborhood, RBN and DDN

For networks with non-local (random) wiring, RBN and DDN, DDLab provides methods (chapter 12) to specify how each "target" cell is connected to other cells with respect to its "pseudo-neighborhood", identical to the neighborhood in 1d, 2d and 3d CA. The set of cells that influence a given cell (its actual "neighborhood" to use the term loosely) may be scattered arbitrarily throughout the network. Each scattered cell is wired to one position in the target cell's pseudo-neighborhood. This allows a CA rule to be applied equally to regular CA, RBN and DDN (see chapter 13).



the target cell

0 k=1
1 0 k=2
2 1 0 k=3
3 2 1 0 k=4
4 3 2 1 0 k=5
5 4 3 2 1 0 k=6
6 5 4 3 2 1 0 k=7
7 6 5 4 3 2 1 0 k=8
8 7 6 5 4 3 2 1 0 k=9
9 8 7 6 5 4 3 2 1 0 k=10
10 9 8 7 6 5 4 3 2 1 0 k=11
11 10 9 8 7 6 5 4 3 2 1 0 k=12
12 11 10 9 8 7 6 5 4 3 2 1 0 k=13

Figure 10.1: Neighborhood indexing 1d, k=1 to 13. The central "target" cell is shown in red. The same principle applies for k=14 to 25 (TFO-mode only).

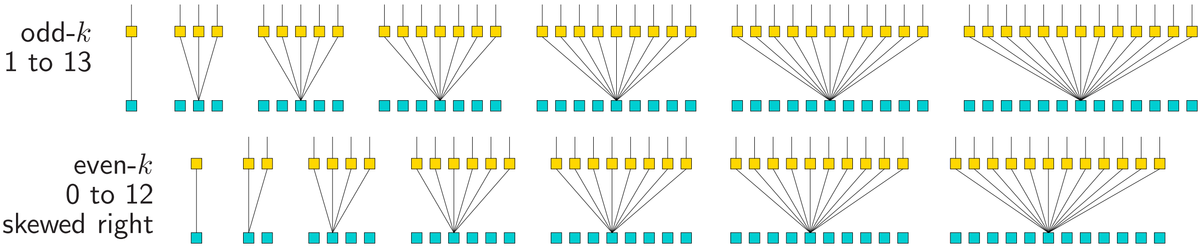


Figure 10.2: 1d neighborhood templates as defined in DDLab, shown up to k=13 – the same principle applies for k=14 to 25 (TFO-mode only). Note that for even k the neighborhood is asymmetric, skewed to the right. For effective k=0 see section 9.3.

10.3 1d neighborhood

The 1d neighborhoods for k=1 to 25 are defined and indexed as shown in figures 10.2 and 10.1. Cells in the neighborhoods are indexed in reverse order, k-1, k-2,..., 0. For odd k the target cell is centered, for even k the neighborhood is asymmetric with an extra cell on the right.

10.4 2d neighborhood

The 2d neighborhoods for k=1 to 25 are defined as shown in figure 10.3. $k \geq 14$ apply in TFO-mode only. To achieve maximum symmetry and for outer-totalistic rules (section 14.2) the target cell itself is sometimes not included in the neighborhood. Cells in the neighborhoods are indexed k-1,k-2,..., 0, in reverse order in sucessive rows, from the top down, as in the example for k=9 in figure 10.4. Note that for a hexagonal lattice to work properly, the height of the network, j, should be even.

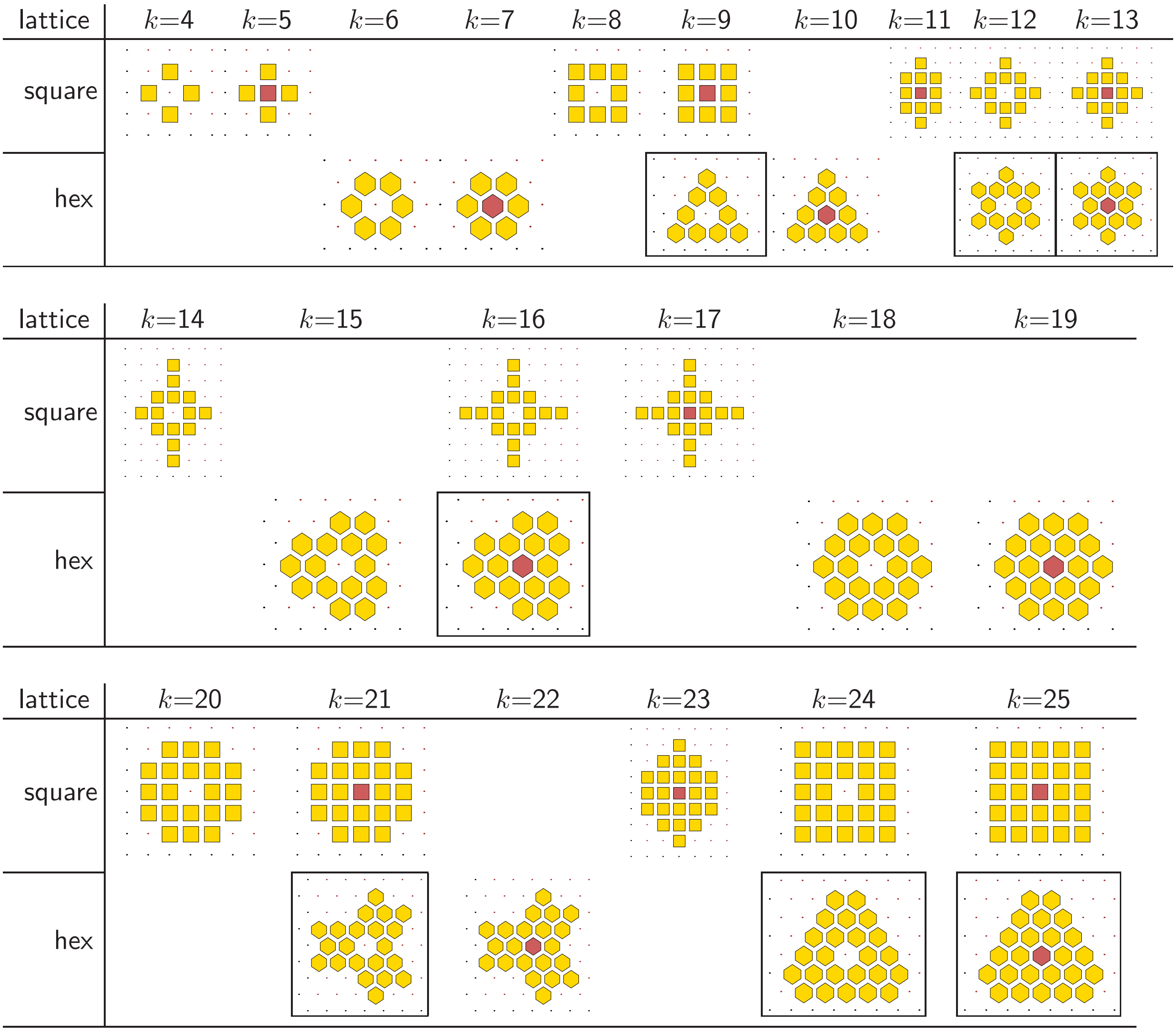


Figure 10.3: 2d neighborhood templates as defined in DDLab, for k=4 to 25. k=1 to 3 are the same as 1d, confined to the i coordinate. $k \geq 14$ apply in TFO-mode only. Each k has a template based on either a square or hexagonal layout, sometimes both, which is designed for best symmetry. If both, either hex or square can be selected – the neighborhoods shown without a black frame are the defaults. The templates may include or exclude the target cell. With a square layout, k=5 is known as the vonNeuman neighborhood and k=9 as the Moore neighborhood – applied in the "game-of-Life" (sections 14.2.2 and 16.10). With a hex layout, k=6 applies to the "beehive rule"[33] and k=7 to the "spiral rule"[34].

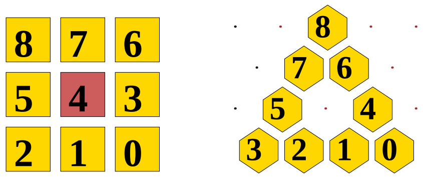


Figure 10.4: 2d neighborhood templates are indexed in reverse order starting with 0 bottom left, as in these examples for k=9 for square and hexagonal neighborhoods. The index numbering is toggled on/off with **index-i** in section 17.4.

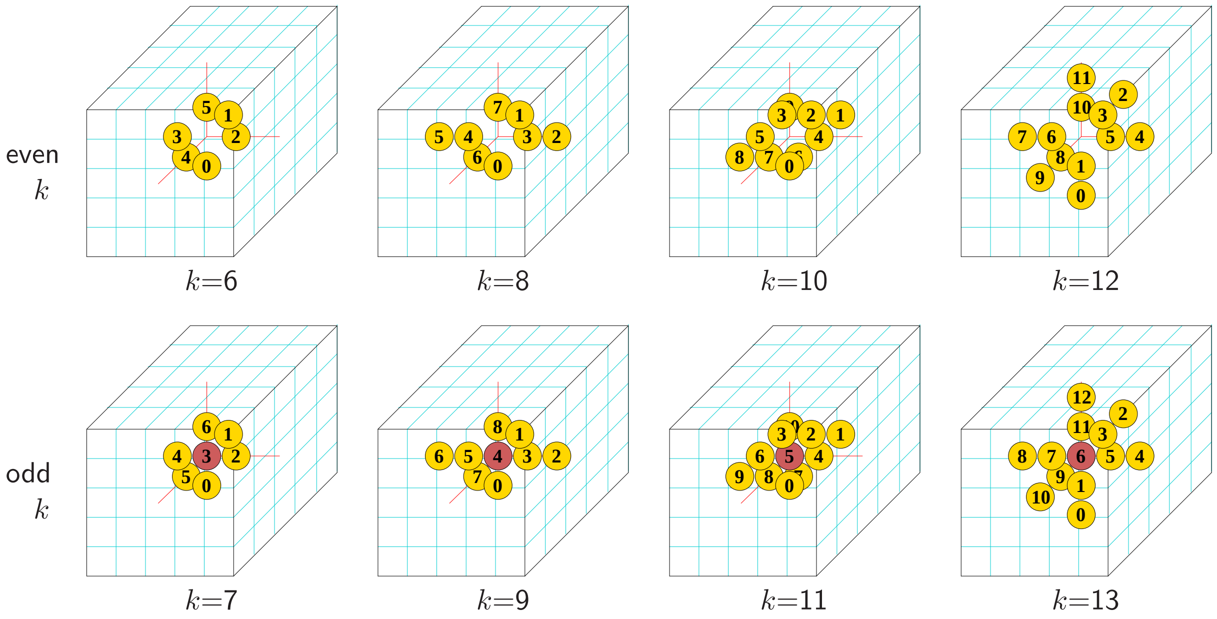


Figure 10.5: 3d neighborhood templates and indexing as defined in DDLab, k=6 to 13. For even k the target cell itself is not included in the neighborhood. The view is isometric as if looking up into a cage. The 3d neighborhood and network is simultaneously displayed in 2d (section 17.7). The index numbering is toggled on/off with **index-i** in section 17.4.

10.5 3d neighborhood

The 3d neighborhoods for for k=6 to 13 are defined and indexed as shown in figure 10.5. To achieve maximum symmetry, for even k the target cell itself is not included in the neighborhood.

In 3d networks, the neighborhoods define a cubic grid on the 3-torus. For $kleq5$ the neighborhoods are the same as for 2d, confined to horizontal i, j layers – for $k \geq 6$ the neighborhood penetrates the vertical h axis. Note that the only fully symmetrical 3d neighborhoods are a 3d nearest neighbor cross – k=6 and 7, and a 3d next-nearest neighbor cross – k=12 and 13.

10.6 Network geometry

The cells in a network size n are indexed and arranged in a regular 1d, 2d or 3d space or lattice, with axial dimensions i for 1d, i, j for 2d and i, j, h for 3d, as shown in figures 10.6, 10.7 and 10.8 – these examples are from the network-graph presentation[1] in chapter 20. This network "geometry" has real meaning for CA, or RBN/DDN where each cell's random inputs are confined close to the cell itself. For networks with fully random wiring the geometry simply allows convenient indexing and representation. The cells are also indexed $n-1 \cdots 0$.

CA have periodic boundary conditions, where opposite edges join, so CA (local) wiring in 1d

[1] The network-graphs were saved as vector PostScript files (chapter 36). The annotations were added within these ascii files and checked in Ghostview.

creates a ring of cells. 2d CA wiring creates a regular square or hexahonal lattice on the torus. 3d CA wiring creates a 3-torus. RBN/DDN where inputs are confined close each target cell also have periodic boundary conditions by default, but this can be suppressed.

10.6.1 1d network indexing

1d networks size n are indexed $n-1 \cdots 0$, where $i = n$, and are usualy laid out horizontally following figure 10.6 (left).

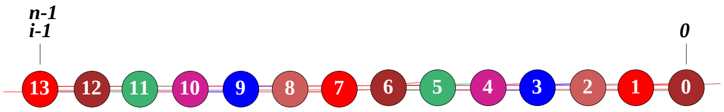
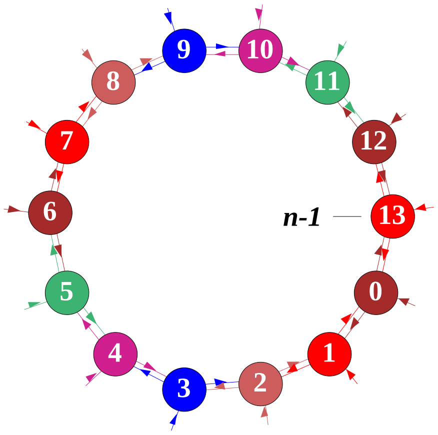


Figure 10.6: A 1d CA, n=14 k=3, showing the 1d network indexing $n-1 \cdots 0$, so $13 \cdots 0$, in horizontal and circle layout.



10.6.2 2d network indexing

2d networks size n are indexed $n-1 \cdots 0$, as in figure 10.7 for n=36, where $[i,j]$=[6,6] with square layout (k=4, left) and hex layout (k=6, right). i is the "width" or "columns", j is the "depth" or "rows". Note that the cell positions shown, x, are 1d indexes, For a 2d network i, j, to convert I, J coordinates to a 1d index, $x = iJ + I$. Converely, $I = x \mod I$, $J = \left\lfloor \frac{x}{I} \right\rfloor$.

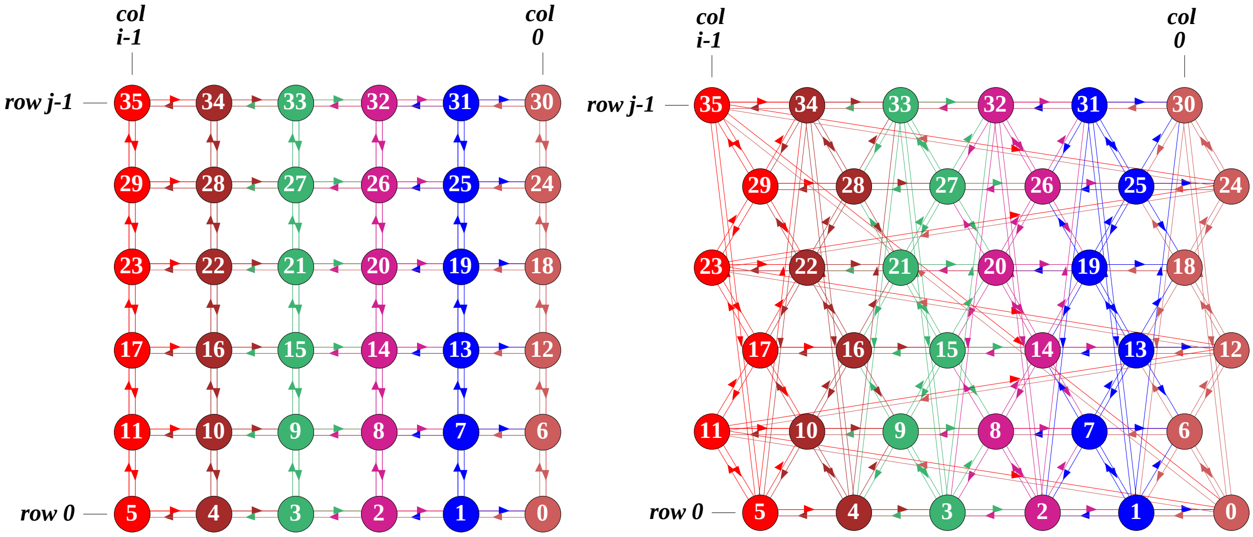


Figure 10.7: A 2d CA, n=36, where $i \times j = 6 \times 6$, showing the 2d network indexing $35 \cdots 0$, with square layout (k=4, left) and and hex layout (k=6, right).

10.6.3 3d network indexing

3d networks size n are indexed are indexed $n - 1 \cdots 0$, an in figure 10.8 for n=64, where $[i, j, h]$=[4,4,4], k=6. Note that cell positions shown, x, are 1d indexes, For a 3d network i, j, h, to convert I, J, H coordinates to a 1d index, $x = ijH + iJ + I$. Converely, $I = x \bmod i$, $J = \left|\frac{x \bmod ij}{i}\right|$, $H = \left|\frac{x}{ij}\right|$.

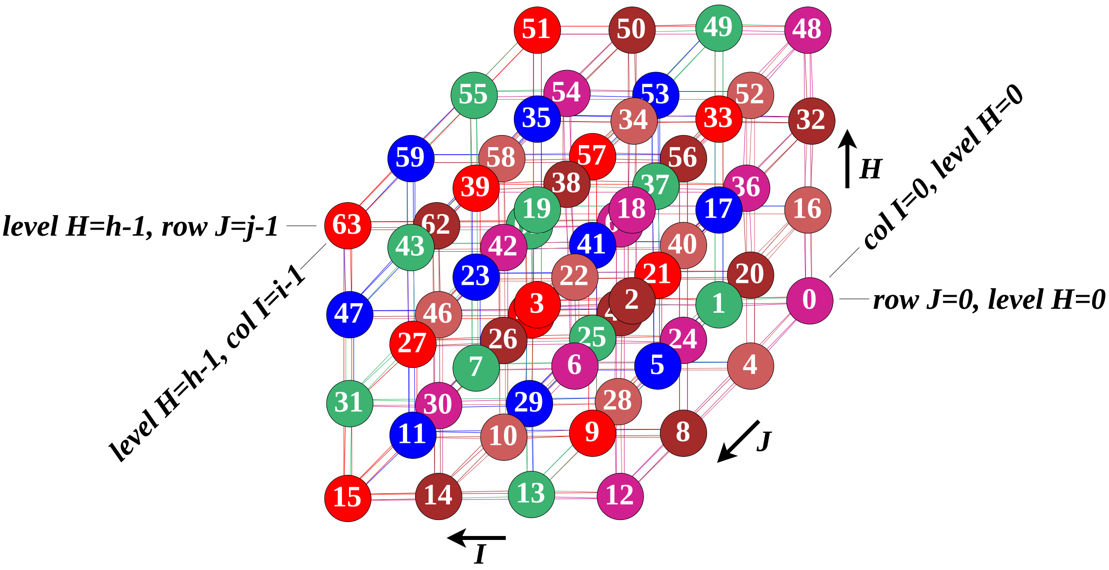
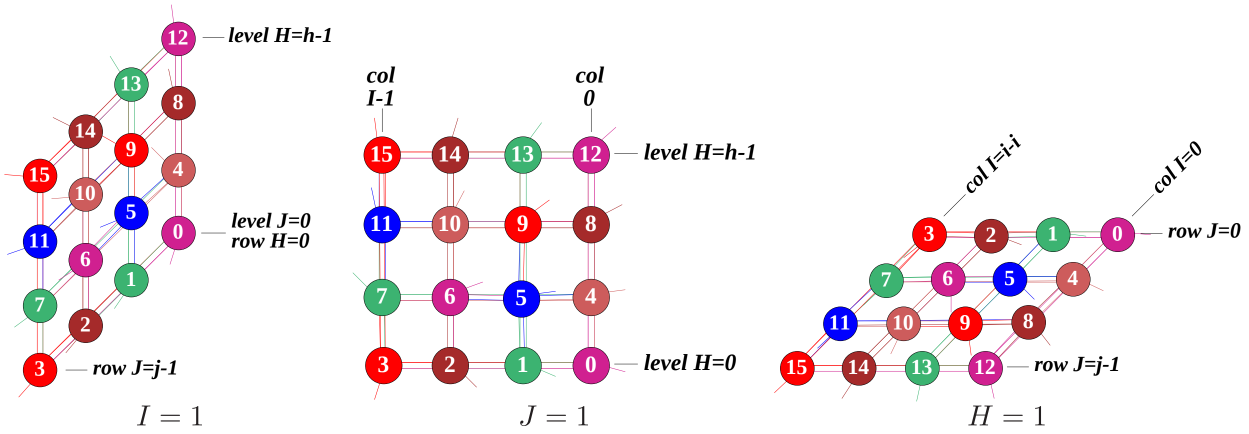


Figure 10.8: A 3d CA, n=64, $[i, j, h]$=[4,4,4], k=6, cells numbered 0-63, shown as a 3d network-graph, (see chapter 20). The graph should be viewed as if looking up from below into a cage, were the bottom layer of cells are numbered 0-15.



$I = 1$ $J = 1$ $H = 1$

Figure 10.9: To clarify 3d network indexing, the network-graphs above show three examples where the width (columns) i, depth (rows) j, and height (levels) h, are made to equal 1 in turn, so from left to right $[i, j, h]$=[1,4,4],[4,1,4],[4,4,1]. k=6, cells numbered 0-15.

Chapter 11

Setting the wiring, quick settings

The main wiring prompts are displayed in context dependent top-right windows. The first wiring prompt gives options for quick wiring settings for CA in 1d, 2d (hexagonal or square) or 3d, random wiring for just 1d, or loading a wiring file (see section 11.3). Alternatively, *special* wiring allows more flexible wiring requiring further wiring prompts, described in chapter 12.

11.1 The first wiring prompt

The first wiring prompt is as follows,

> **WIRING: special-s load-l random-r**
> **regular 3d-3, 2d-2(hex+x square+s), 1d-def:**

11.2 Special wiring

If **s** is selected in section 11.1 above, prompts are presented for various special wiring options, including setting random wiring for 1d, 2d and 3d networks (with various biases), described in chapter 12.

11.3 Loading the wiring scheme

If **l** is selected in section 11.1, filing prompts will allow a `.wir` file to be loaded. Loading such a file resets the network size (see chapter 19).

11.4 Random 1d wiring

If **r** is selected in section 11.1, a wiring scheme will be assigned at random according to the previously selected neighborhood k or k-mix settings. The next prompt allows a graphic representation to be displayed, where the wiring can be reviewed and altered (see chapter 17).

To assign random wiring for 2d or 3d networks, or bias the random wiring in various ways, **s** should be selected in section 11.1, and "special" wiring assigned as described in chapter 12.

11.5 Regular 1d wiring

If **return** (the default) is selected in section 11.1, the wiring is set up as regular 1d, with periodic boundary conditions. The network may be a CA, or may have mixed k. The remaining wiring options will be skipped.

11.6 Regular 2d or 3d wiring

If **2** or **3** is selected in section 11.1, the wiring will be set up as regular 2d or 3d CA wiring according to the default neighborhoods in section 10.4. In both cases boundary conditions are periodic, i.e. the 2d array can be imagined as drawn on the surface of a torus, and the 3d array on a 3-torus. The network can also have a k-mix with CA wiring for each neighborhood. Entering **2x** or **2s** will force the neighborhood, and the initial presentation of the lattice, to be hex or square, overrideing the defaults in section 10.4. Just the hex/square presentation can be toggled later in the program.

As well as setting regular wiring, further prompts will appear - to set the 2d or 3d network size, and k or a k-mix. Previous n and k settings from the main sequence of prompts will be superseded.

11.6.1 Setting 2d and 3d network size

The following option sets the $i \times j$ (width × depth) dimensions of 2d networks, and the $i \times j \times h$ (width × depth × height) dimensions of 3d networks.

2d, enter width (def 40): depth (def 40):
or
3d, enter width (def 9): depth(def 9): height(def 9):

Enter the width, depth (and height), keeping in mind the network size limitations (see section 8.3). For space-time patterns the maximum network size, n_{max}=65025, corresponding to a square 2d network, 255×255. The maximum 3d cube is 40×40×40. The network need not be square or cubic as long as $nleq$65025. The size limits apply so that network positions can be indexed by an unsigned short (16 bits).

If the dimensions selected for a 2d or 3d network exceed 65025, a message such as the following is displayed,

255x256 too big! max size=65025 cont-ret: *(for 2d)*
or
40x40x41 too big! max size=65025 cont-ret: *(for 3d, values shown are examples)*

Pressing **return** restores the network size prompt.

If the intention is to generate attractor basins, the 2d size should be kept small, for eexample 3x3 for quick results, 4x4 for slower results.

11.6.2 Set k, or k-mix

If 2d or 3d was selected, the next top-right prompt resets k, or the k-mix, as was described in chapter 9.

> **Neighborhood size k: kmix-m, or enter 1-25 (def 9):** *(max-k depends on the context: TFO and v. Default k depends on the previous selection)*

Enter a new value to reset k, or **return** to accept the default. Enter **m** for a k-mix and follow further instructions in chapter 9.

11.7 Networks too big for a basin field

If FIELD-mode for a basin field was selected at the first prompt in chapter 6.2, and a network size n in 2d or 3d is set larger than 31, FIELD-mode will be automatically changed to SEED-mode for a single basin or a space-time pattern, with the following message,

> **size too big for FIELD-mode (max 31)**
> **changed to SEED-mode! cont-ret**

The title bar across the foot of the screen (section 5.5) is redrawn to reflect this change. The significance of FIELD-mode, SEED-mode (and TFO-mode) was described in chapter 6.

Intentionally Blank

Chapter 12

Setting special wiring

Selecting **s** in section 11.1 allows a greater range of methods for wiring the network. Selecting special wiring is necessary to set random wiring for 2d or 3d networks. The network can be assigned regular or random wiring in 1d, 2d (square or hex), 3d, or hypercube wiring, and the wiring can be "hand wired" and biased in a variety of ways.

Special wiring also allows the following functions, some of which may be restricted to predefined parts of the network (see also chapter 17).

- Confining random wiring within a set zone, from which some wires may be released.
- Suppressing periodic boundary conditions, selectively for the various axes in 2d and 3d.
- Forcing or disallowing self-wiring.
- Forcing distinct wiring i.e. no duplication,
- Setting the same random wiring template for every cell in the network.
- There are additional functions for 3d networks.
  - Suppress links to particular layers of the network.
  - Force direct links to specific layers.
  - Apportion fractions of available random links between specified layers.

Once the special wiring is set, it can be amended by many flexible methods from the "wiring graphic" described in chapter 17.

12.1 Setting up the network geometry

Entering **s** in section 11.1 gives the first special wiring option, which allows the network geometry to be set as 1d, 2d (square or hex), 3d, and also as a hypercube for appropriate n and k.

hypercube-h, 3d-3, 2d-2 (hex+x), 1d-def
(hypercube-h only if $k = log_2 n$ or $log_2 n + 1$, for example k=3, n=8 or 9)

Sections below examine each option in detail.

12.2 Hypercube wiring

In a fully connected n-dimensional hypercube, each state receives input from its $log_2 n$ one-Hamming distance “neighbors”, and may also receive input from itself. If n=8 and $k = log_2 n = 3$, the hypercube is a simple cube with network states at the vertices and bi-directional links at the edges.

The hypercube wiring option is only active if the network size n is a power of 2, (2, 4, 8, 16...), and $k = log_2 n$, or $k = log_2 n + 1$ where vertices also receive one additional input from themselves. For a simple cube, n=8, k must equal 3 or 4 for the hypercube option to appear.

If **h** for a hypercube is selected in section 12.1 a further option allows degrading the hypercube “wiring”, i.e. pruning uni-directional connections, according to some probability,

```
set % prob (def 100):66 (for example)
```

For example, for an n=8, k=4 hypercube, if 66 is entered, wires are set with a probability of 66%. A further prompt is presented,

```
part hcube:%wire-prob=66 act=66.8 bi=6/12=50.0%
nhood mix=44124322 (values shown are examples)
reset-r save-s cont-ret:
```

This indicates that the actual fraction of remaining wires and the number of and percentage of bi-directional links. Figure 12.1 gives 3d graph examples.

Enter **r** to reset the wiring probability, **s** to save the k-mix (see chapter 19), and **return** to accept. The resulting k-mix is given for the network as in section 9.12.

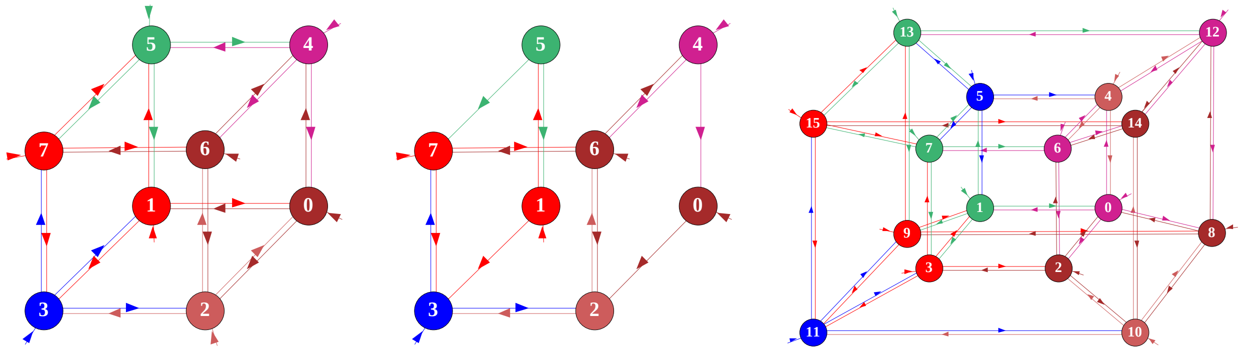


Figure 12.1: *left*: a simple n=8 k=4 hyprecube, which includes self-links, and *center*: a degraded version of the hypercube with some links missing. *right*: a n=16 k=5 hyprecube. The 3d network-graph option (section 20.4) was used for the figures. Output directed links are the same color as the source node. The figures should be viewed as if looking up from below. The graph was rearanged by dragging nodes (section 20.5). To regenerate these layouts automatically, enter **f** in section 20.4 and load the layout files `hyp8.grh` or `hyp16.grh`.

12.3 1d, 2d and 3d special wiring

If **return**, **2** or **3** was entered in section 12.1, a 1d, 2d or 3d network will be selected to receive either regular or random wiring, or "hand wiring" in a wiring matrix "spread sheet".

For regular wiring, the wiring dimension would usually be set to match the network dimension, but its also possible to assign regular 1d wiring to a 2d network, and regular 1d or 2d wiring to a 3d network.

Further network parameters, $i \times j$ ($\times$ h for 3d)), k or the k-mix, and the wiring itself are set subsequently[1]. The following option is presented,

> **hand wire-h,**
> **regular 1d-1, random-def:** *(if 1d was set in 12.1)*
> or
> **regular 2d-2 (hex+x), 1d-1, random-def:** *(if 2d was set in 12.1)*
> or
> **regular 3d-3, 2d-2 (hex+x), 1d-1, random-def:** *(if 3d was set in 12.1)*

Once the network wiring has been set, the wiring of individual cells may be reviewed and altered in a "wiring matrix" or 'wiring graphic" (chapter 17).

12.4 Special wiring, local

Enter **1**, **2**, **2x** or **3** in section 12.3 for local 1d, 2d or 3d (CA) wiring. Enter **2x** to force a 2d hexagonal layout if the default is square - this depends on k (see section 10.4).

Note that for regular 1d wiring in a 1d network, the n, k parameters set previously in chapters 8 and 9 remain valid. For regular 2d and 3d wiring new prompts will be presented to reset these parameters.

12.4.1 Local 1d treated as random

For a 1d network, if **1** for regular 1d wiring was selected in section 12.3, a subsequent option allows the regular wiring to be treated as if it were "random" or non-local (always the case for 2d and 3d networks), allowing the regular wiring to be altered in various ways, and also for wire moves to be allowed in "learning" (see chapter 34). However, the "compression" of attractor basins (section 26.1) may be applied as long as the network retains regular 1d wiring. The following prompt is presented,

> **treat regular 1d as random wiring-1, and compress-2:**

12.4.2 Local 2d and 3d

For 2d or 3d networks, if **2**, **2x** or **3** for local 2d or 3d (CA) wiring was selected in section 12.3, further prompts are presented to set the network size, $i \times j$ ($\times$ h for 3d), as described in section

[1] For 1d, n, k or the k-mix, will remain as previously set in sections 8.2 and 9.1

11.6.1, and neighborhood k, or the k-mix, as described in chapter 9. Note that these settings override those previously set for n and k in chapters 8 and 9.

Regular 2d and 3d wiring is treated as if it were "random" or non-local, allowing it to be altered in any way. This is not the case by default for regular 1d wiring (see section 12.4.1 above).

12.5 Special wiring, random

If random wiring (the default) is selected in section 12.3, (for 1d, 2d or 3d), and n, k, or the k-mix, have been set, a series of context dependent prompts are presented in sequence to allow a variety of biases to the random wiring *(values shown are examples)*,

bias random wiring: confine to local zone (max=14 def=14): CA-c:
release some wires from zone (def 0, max 3): *(depending on k)*
suppress periodic boundary-s: i: j: h: *(i: j: for 2d, i: j: h: for 3d)*
exclude all selfwiring-2, selfwire centre wire-1:
distinct wiring (no duplication)-n:
suppress links to layers (0-8), range1 low: high: range2 low: high: *(3d only)*
force direct link to a layer, enter (0-8): *(depending on h, 3d only)*
force random links to specific layers-y: *(3d only)*
same wiring everywhere-e:

DDLab will do its best to reconcile any contradictory settings. A small top-centre window **accept defaults-d** that at any time further prompts in this sequence can be skipped by entering **d**. Enter **q**, or the right mouse button, to backtrack. The options are explained below.

12.5.1 Applying wiring biases to parts of the network

At this early stage in DDLab's prompts the biases apply to the whole network, but it is important to note that the biases may be applied to just single cells or pre-defined parts of the network, as well as to the whole network, when these same options are accessed from the wiring graphic (section 17.3). In this case the biases only take effect when **r** (for random wiring) is selected in section 17.4 (see also 17.8.5).

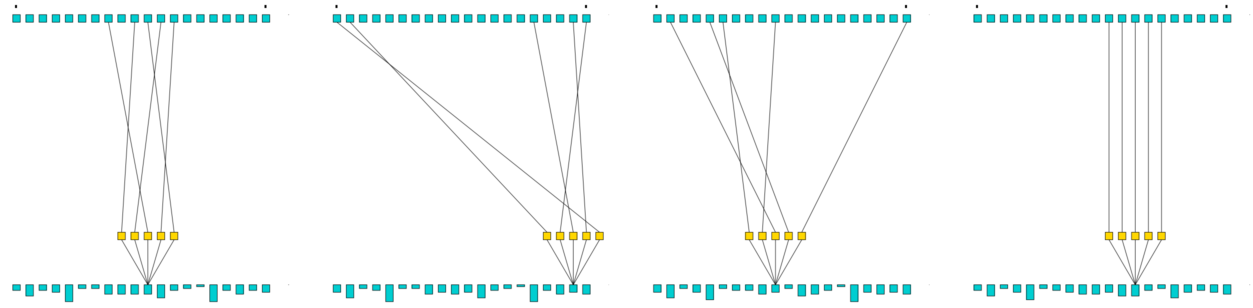
To design particular wiring schemes, especially if the wiring needs to be tailored in specific ways between different parts of the network, its usually easier to set up a dummy wiring scheme at this stage, them revise it in chapter 17.

12.5.2 Confining random wiring to a set zone

The wiring may be confined within a periodic zone of a given diameter relative to each target cell. This diameter relates to the network geometry. Enter the local zone diameter at the prompt

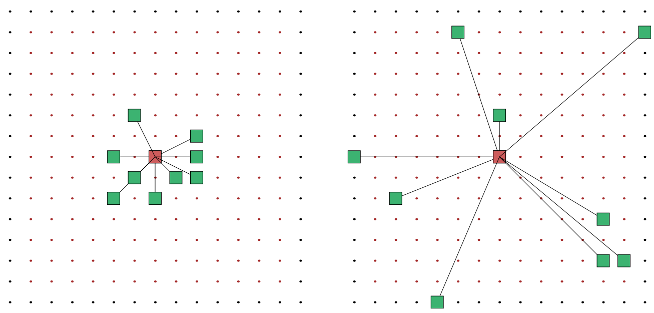
confine to local zone:

in section 12.5. 1d, 2d and 3d examples are shown in figures 12.2, 12.3 and 12.3. which illustrate how network wiring is represented, explained more fully in chapter 17.



(a) random, zone=7 (b) random, zone=7 (c) random (d) CA wiring

Figure 12.2: Confining 1d k=5 random wiring within a local zone of a set diameter. n=15. (a) and (b) show random wiring within a 7 cell local zone, where (b) illustrates periodic boundary conditions. (c) shows fully random wiring, and (d) local, CA type, wiring for comparison. Wiring from the pseudo-neighborhood is shown connected to to the previous time-step (see chapter 17)



(b) a) random, zone=5 (b) fully random

Figure 12.3: Confining 2d random wiring within a 5 cell local zone. n=15×15. (a) random wiring confined within a 5 cell zone. (c) fully random wiring.

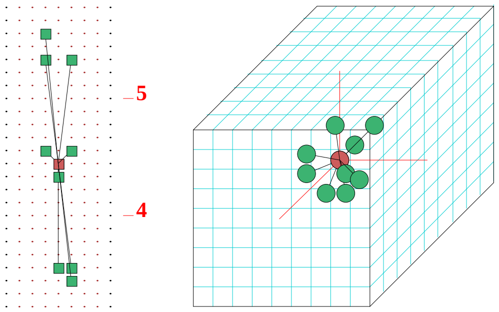


Figure 12.4: Confining 3d k=9 random wiring within 3 cell local zone. n=9×9×9. *left*: 2d showing successive layers. *right*: 3d isometric.

12.5.3 Setting local wiring as random

Enter **c** at the prompt **CA-c:** in section 12.5 to set local (CA) wiring as "random wiring". CA wiring in 1d, 2d or 3d will be set depending on the native dimension of the network. This allows subrequent changes to be made to the CA wiring, for example, releasing some wires described below.

12.5.4 Release wires from zone

Some wire can be released from the local zone set in section 12.5.2 and from the CA wiring set in section 12.5.3. Enter the number of wires to be released at the prompt

> **release some wires from zone (def 0, max 7):** *(values shown are examples)*

in section 12.5. The number of wires specified will be chosen and reassigned at random to any position in the network, unless this is restricted by further biases.

12.5.5 Suppress periodic boundary conditions

For a 1d network, enter **s** at the prompt

> **suppress periodic boundary-s:**

in section 12.5 to suppress periodic boundary conditions.

For 2d and 3d networks further prompts appear,

> **i:**, **j:** (and **h:** for 3d).

Periodic boundary conditions can be suppressed independently for each axis, i, j and h. To do so, enter **s** in response to the prompts.

12.5.6 Self-wiring

Enter **2** or **1** at the prompt

> **exclude all selfwiring-2, selfwire centre wire-1:**

in section 12.5 to exclude self-wiring, or to force target cells to wire to themselves. Self-wiring means that the cell will provide an input to its own pseudo-neighborhood, at the central position if available, or to a nearby position if not. Chapter 10 describes the pseudo-neighborhoods.

12.5.7 Distinct wiring

Wiring may be made distinct to prevent two or more positions in the pseudo-neighborhood to be wired to the same cell. Enter **n** at the prompt

> **distinct wiring (no duplication)-n:**

in section 12.5 for distinct wiring. DDLab will do its best to achieve distinct wiring, but this may conflict with a small “wiring zone” set in section 12.5.2 and with other wiring bias settings.

12.5.8 Suppress links to 3d layers

Links to horizontal layers in a 3d network can be suppressed. Two ranges of layers can be designated. Prompts are presented in sequence. Enter the levels to be suppressed starting with the lowest level. For example, the following entries would suppress links to layers 0,1,2,3 and 6,

> **suppress links to layers (0-8), range1 low:0 high:3 range2 low:6 high:6**

12.5.9 Force a direct link to a 3d layer

Direct links can be “forced” to each cell in a designated 3d layer. A direct link means that cells at position i, j link one wire to position i, j in the designated layer. The wire originates from the pseudo-neighborhood index 0 (see chapter 10).

For example, for a 9 layer 3d network, enter the required layer index at the prompt

> **force direct link to a layer, enter (0-8):**

to which direct links will be forced. By default, if a layer is specified, all cells make a direct link, but this can be restricted to a specific cell, range of cells, or to a specific layer or range of layers from the wiring graphic (see chapter 17).

This option may be used to provide a constant input pattern to a 3d network, for example to model inputs from a “retina”.

12.5.10 Force random links to specific 3d layers

Fractions of the available random links can be assigned to designated 3d layers. Previous biases to confine wiring to a set zone will be respected for the horisontal plane, but the vertical constraints will be overridden.

If **y** is entered at the prompt

force random links to specific layers-y:

in section 12.5, the following series of options are presented in a top-right window, for example in an 9 layer, k=7, 3d network,

k=7, enter n links to layer 8 (7 available):3 *(3 entered, 4 remain)*
k=7, enter n links to layer 7 (4 available):1 *(1 entered, 3 remain)*
. . .
k=7, enter n links to layer 0 (3 available):3 *(none remain)*

The prompts continue until layer 0, or until all links have been assigned and none remain. To miss a layer enter **return** or **0**. If links remain at the end, they will be assigned at random, but still according to the biases specified in section 12.5.

12.5.11 Same random wiring everywhere

The same random wiring “template” can be applied to every cell in the network to create a quasi-CA with periodic boundary conditions. To do this enter **e** at the prompt

same wiring everywhere-e:

in section 12.5. Biases previously specified in section 12.5 and other options will be respected as much as possible. The “same random wiring everywhere” option can of course be restricted to just part of the network from the wiring graphic (chapter 17).

12.6 Wiring by hand

If **h** is selected in section 12.3, a blank wiring scheme is presented in the form of a matrix or “spread sheet”, which may be filled in with the wiring positions for each cell’s pseudo-neighborhood index. A completed wiring matrix can be amended in the same way (section 17.2.2).

Columns give the cell’s pseudo-neighborhood index, K (k-1...0), Rows give the cell network position, N (n-1...0). The 0-0 grid, or the 0-minimum n grid if the whole matrix does not fit within one window, is in the lower right hand corner. Each grid records the position in the network x, (n-1...0), to which the K’th wire of the N’th cell is connected.

Note that positions N and x are 1d indexes, even if the network is 2d or 3d. For 2d $I \times J$, to convert ij coordinates, $x = Ij + i$. For 3d $I \times J \times H$, to convert ijh coordinates, $x = IJh + Ij + i$.

Move around the matrix with the left/right/up/down arrow keys. Enter the new position at the flashing green cursor and complete the entry by moving to another grid with an arrow key or **return**. On a blank grid, or on zero, just **return** gives a random position. An entry outside the network limits will be ignored. Enter **q** to complete. Any undefined grids will be set to position 0. While the wiring matrix is being set, the following reminder is displayed in a top-right window,

> **hand wire/revise: jump-j** *(values shown are examples)*
> **enter wiring positions 0-144 (return on blank/0=random)**
> **move-arrows more/complete-m layout-l font-f quit-q**

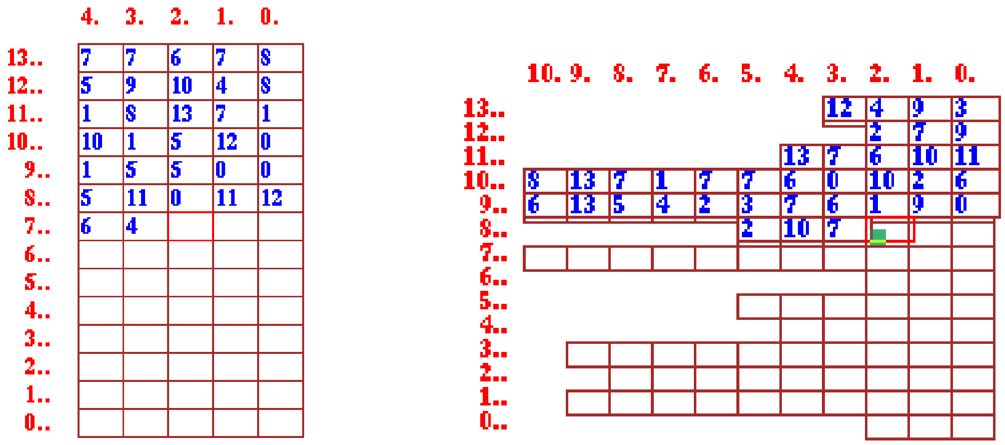
Enter **l** or **f** to alter the presentation of data and the amount visible in the matrix window. **f** is a 3-way toogle for the font size between normal, medium and small. **l** changes the matrix window width. The following top-right prompt is presented,

> **change window width (922-231 now=462):** *(values shown are xamples)*

Enter the new width in pixels within the limits indicated.

Large networks may require several successive windows to display the matrix. Enter **m** to see the next window. Moving beyond the the top or bottom row in the current window with the arrow keys or **return** also brings up the preceding or next window. **return** on the very last (bottom right) entry makes the program continue. Enter **j** to jump to a new cell index, the following prompt is presented,

> **jump to index (1599-0):** *(for a 2d network, 40×40)*



	4.	3.	2.	1.	0.
13..	7	7	6	7	8
12..	5	9	10	4	8
11..	1	8	13	7	1
10..	10	1	5	12	0
9..	1	5	5	0	0
8..	5	11	0	11	12
7..	6	4			
6..					
5..					
4..					
3..					
2..					
1..					
0..					

(a) k=5, n=14

	10.	9.	8.	7.	6.	5.	4.	3.	2.	1.	0.
13..								12	4	9	3
12..									2	7	9
11..							13	7	6	10	11
10..	8	13	7	1	7	7	6	0	10	2	6
9..	6	13	5	4	2	3	7	6	1	9	0
8..						2	10	7			
7..											
6..											
5..											
4..											
3..											
2..											
1..											
0..											

(b) k= 3 to 11, n=14

Figure 12.5: Setting wiring by hand on the blank wiring matrix, shown partly filled. k-1...0, indexes columns, n-1...0, indexes rows. (a) shows an example matrix for a k=5, n=14 network. (b) shows an example matrix for a mixed k network, k=3 to 10, n=14.

Enter the new cell index, which will become the first entry in the top row.

Enter **q** to accept the wiring and go to the prompt in section 12.7 below.

The wiring scheme can be reviewed and revised in the same wiring matrix format, or as a wiring graphic in 1d, 2d or 3d, as described in chapter 17.

12.7 Reviewing wiring

After the special wiring has been set as described in this chapter, various options allow the wiring to be reset, reviewed and ammended from a wiring graphic or wiring matrix.
A prompt similar to the following is presented,

(for a 1d network)
1d network (n=150), review/revise, wiring only - rules not set
graph-g, matrix: revise-m view-M prx-Mp
graphic: 1d:timesteps-1 circle-c, 2d-2:

(for a 2d network)
2d network (40x40), review/revise, wiring only - rules not set
graph-g, matrix: revise-m view-M prxt-Mp
graphic: 1d:timesteps-1 circle-c, 2d-2:

(for a 3d network)
3d network (15x15x15), review/revise, wiring only - rules not set
graph-g, matrix: revise-m view-M prxt-Mp
graphic: 1d:timesteps-1 circle-c, 2d+3d-3:

These options are described in chapter 17. They provide very flexible methods to review and amend the wiring, and also the rules once set. It may be preferable to set up a suitable dummy network initialy, then tailor it with these options.

If just *quick* wiring was set in chapter 11, these options do not appear at this point in the program, but they appear in any case after the rules have been set (chapters 13-16).

Intentionally Blank

Chapter 13

Rule types

Rules in DDLab can be set according to a number of types (possibly overlapping): full lookup-table (rcode), k-totalistic (kcode)[1], t-totalistic (tcode), outer totalistic, and reaction-diffusion, so before a rule is actually selected in chapter 16, options are presented to select the type, and also whether a rule-mix is required. These options are described below, and the chapter goes on to explain the rule types in detail. Further prompts in chapters 14 and 16 describe the rule-mix options and the wide variety of rule sub-types within the chosen type.

13.1 Selecting the rule type

After the network wiring has been set or defaults accepted (in chapters 11 and 12), the next prompt in the main sequence selects the type of rule. The prompt differs between rules based on full lookup-tables (SEED-mode or FIELD-mode) which may be rcode, kcode or tcode, and just totalistic rules based on only the shorter kcode and tcode (TFO-mode).

13.1.1 select full lookup-tables

In SEED-mode or FIELD-modes, all rule types rely on full lookup-tables. Totalistic rules and reaction-diffusion rules will be automatically transformed into rules with a full lookup-table. The following prompt is presented,

totalistic: tcode-t kcode-k (def-rcode):

enter, **return** ... for rcode (allows one method for reaction-diffusion))
k ... for kcode, a k-totalistic rule
t ... for tcode, a t-totalistic rule

13.1.2 select totalistic lookup-tables

In TFO-mode for kcode or tcode lookup-tables, the following prompt is presented,

totalistic only: outertot-o/+o, tcode-t, (def-kcode):
(outertot-o/+o does not apply for a k-mix, set in sections 9.1 or 11.6.2)

[1] Suggested by Antonio Lafusa[3].

enter, **return** ... for kcode, a k-totalistic rule
t ... for tcode, a t-totalistic rule
o ... for an *outer* k-totalistic rule (allows another method for reaction-diffusion)
to ... for an *outer* t-totalistic rule

If *outer*-totalistic is selected, refer to section 14.2 for further options.
Note that for v=2, tcode and kcode lookup-tables and the resulting dynamics are identical.

13.2 Rule types and combinations

The rest of this chapter explains the various types of rule, or systems of logic, that act on the neighborhood, or pseudo-neighborhood, to determine a cell's output, and the convensions in DDLab for constructing and ordering rule-tables. The rules described first are Boolean functions as in older versions of DDLab, where the value-range v=2 consists of just [0,1]. Then the rules are generalised for multi-value logic, where the value-range (the number of colors) may be anything from v=2 to 8.

In DDLab there are several types of rules, which are expressed as lookup-tables, as follows:

- **full lookup-tables**: referred to as "rcode" listing the outputs of all possible neighborhood patterns (not TFO-mode)
- **k-totalistic rules**: referred to as "kcode" – lookup-tables listing the outputs for all possible combinations of totals (or frequencies) of the values in the neighborhood. kcode can be selected in TFO-mode, or in SEED-mode or FIELD-mode where the kcode is also expressed as a full lookup-table.
- **t-totalistic rules**: referred to as "tcode" – lookup-tables listing of the outputs of all possible totals when the values in the neighborhood are simply added. tcode can be selected in TFO-mode, or where the tcode is also expressed as a full lookup-table.

  For v=2, kcode and tcode are identical.

In addition DDLab offers combinations of rules as follows::

- **rule-mix:** where each cell in the network can have a different rule, or where a restricted set of rules is distributed throughout the network, described in detail in chapter 14. A network with mixed-k must have a rule-mix.
- **outer-totalistic rules:** (TFO-mode only, and not for mixed-k) a number v of totalistic rules (tcode or kcode) can be employed to create outer-totalistic rules, where a different rule applies according to the value of the center cell (see section 13.7). For v=2 the game-of-Life can be set in this way, though it can also be set as a full lookup-table in section 16.10.
- **reaction-diffusion rules:** or excitable-media[7]. Cells may be either resting, excited, or refractory, and change according to thresholds and values (see section 13.8). The resulting dynamics produces waves, spirals and related patterns that can resemble the BZ reaction and other types of excitable media. A reaction-diffusion rule can be set from a full lookup-table (section 13.8.2) or from an outer-kcode (section 14.2.1). Note that outer-kcode allow a greater range of $[v, k]$ than a full lookup-table, but the latter allows a rule-mix that can include more than one reaction-diffusion rule in the network.

13.2.1 Lookup-table size implications

The lengths of lookup tables, S, depend on v and k.

Rule-tables are the longest . . . $S = v^k$
followed by kcode . . . $S = (v + k - 1)!/(k! \times (v - 1)!)$
followed by tcode . . . $S = k(v - 1) + 1$

This has implications on the k_{Lim} (section 7.2). In SEED-mode or FIELD-mode, kcodes and tcodes will be implimented as the equivalent full lookup tables-tables, whereas in TFO-mode the shorter kcode and tcode allow a greater k_{Lim} for a given value-range v (see also section 6.1).

Various subcategories of rcode and kcode can also be selected in chapter 16 and at a later stages in DDLab, including majority, Altemberg, chain, and isotropic. The same rule may apply to all cells in the network as for CA, or the network may have a mix of different rules.

13.3 Binary full lookup-table – rcode

A full lookup-table (rcode) for binary systems (v=2) has 2^k entries. Table 13.1 shows how the size of the rule-table increases exponentially with k.

k	1	2	3	4	5	6	7	8	9	10	11	12	13
size of lookup table 2^k	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192

Table 13.1: The size of rule-tables k=1 to 13

The convention in DDLab is to list neighborhood configurations from left to right in reverse Boolian value order[2] following Wolfram[17], so that the all 1's neighborhood is on the left, as set out below. Each neighborhood is assigned an output [0,1], and the resulting bitstring defines one of the 2^{2^k} possible rcodes. The most significant bit in the bitstring corresponds to the all 1's neighborhood.

The rules may also be expressed in decimal (if applicable – section 16.6) or in hexadecimal. $k \geq 5$ rules are usually refered to by their hex rule numbers, $k \leq 3$ rules are usualy refered by their more familiar decimal rule numbers, for example, the k=3 binary rcode 60 below,

```
   7     6     5     4     3     2     1     0    - Boolean value = rcode index
  111   110   101   100   011   010   001   000   - k=3 neighborhood
   0     0     1     1     1     1     0     0    - output string = rcode
```

In DDLab the neighborhood configurations are rotated and displayed verticaly for compactness making a so called "neighborhood matrix", showing also the k-index, as in these example for k=3, and k=5 below. A bit-string can be assigned in the corresponding order to make the rcode.

```
               7......0 - rcode index
               :      :
           2 - 11110000
  k index  1 - 11001100
           0 - 10101010
               --------
               11000001 - rcode, decimal 193, or hex c1
```

[2] This can be inverted – section 18.3

```
              31...... ........ ........ .......0 - rcode index
              :                                 :
           4 - 11111111 11111111 00000000 00000000
           3 - 11111111 00000000 11111111 00000000
   k index 2 - 11110000 11110000 11110000 11110000
           1 - 11001100 11001100 11001100 11001100
           0 - 10101010 10101010 10101010 10101010
              -------- -------- -------- --------
              11111100 01100010 10001000 00110111 - rcode
                                                    fc 62 88 37 in hex
                                                    4234315831 in dec
```

13.3.1 The binary neighborhood matrix

In the main sequence of prompts, a graphic of the binary neighborhood matrix is displayed (but not for mixed k or in TFO-mode). Figure 13.1, gives examples for k=1 to 9. The matrix can be rescaled and different parts shown (see section 14.11).

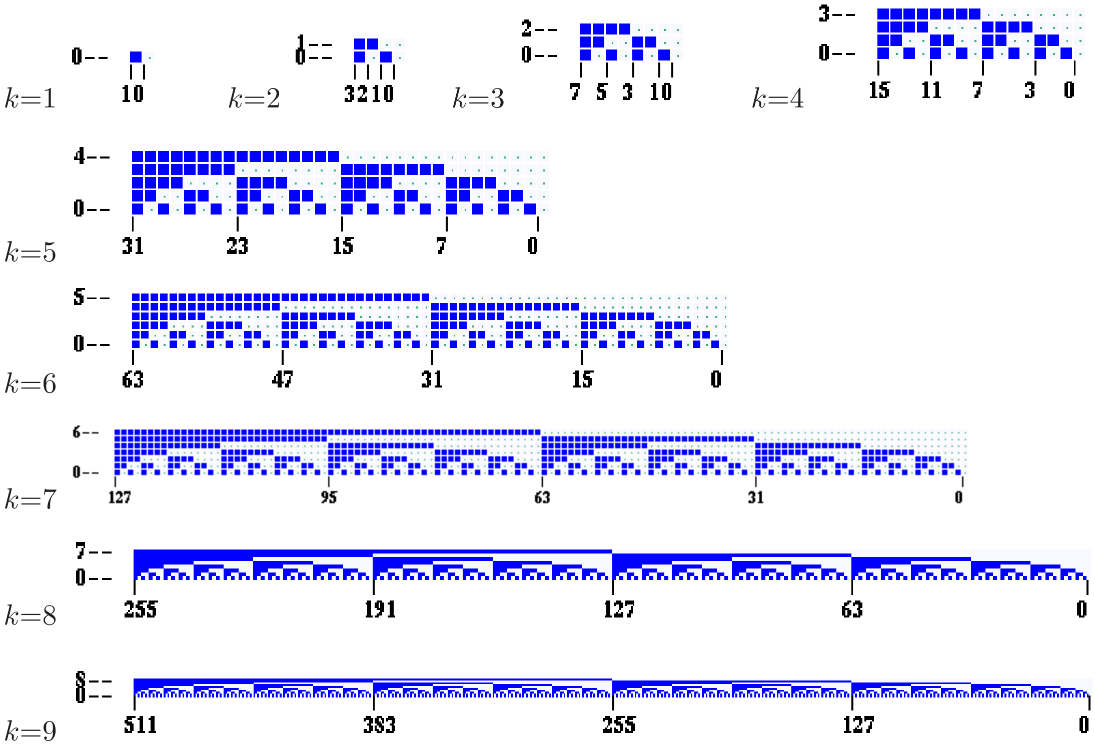


Figure 13.1: The binary neighborhood matrix k=1 to 9, as displayed in DDLab.

13.4 Binary totalistic rules

For binary (v=2) totalistic rules a cell's value depends only on the sum of 1s in its neighborhood - tcode and kcode are identical in binary (see section 13.6). Totalistic tcode is defined by a lookup table of k+1 possible totals set out in reverse value order from k to 0. Each total is assigned an output [0,1]. The resulting bit string defines one of the 2^{k+1} possible tcodes. If DDLab is not in TFO-mode, it automatically transforms the tcode into the full lookup-table rcode format as in section 13.3.

Here is an example for k=5,

```
5 4 3 2 1 0 - totals
- - - - - -
1 1 0 0 1 0 - output = code 50 in dec, 32 in hex,
              rcode = 11101000100000011000000100010110
              or 3900801302 in dec, e8818116 in hex
```

Totalistic codes are also useful for setting threshold functions, for example, the totalistic code `111000` gives the k=5 majority rule.

13.5 Multi-value full lookup-table – rcode

A full lookup-table, rcode, for multi-value systems (v=2 to 8 in DDLab) has v^k entries. Table 13.1 shows how the size of the rule-table increases exponentially with v and k, and also the limits of k relative to v allowed in DDLab - these limits are more generous for totalistic rules in TFO-mode (section 13.6).

k	1	2	3	4	5	6	7	8	9
$v = 3$	3	9	27	81	243	729	2187	6561	19683
$v = 4$	4	16	64	256	1024	4096	16384		
$v = 5$	5	25	125	625	3125	15629			
$v = 6$	6	26	216	1296	7776				
$v = 7$	7	49	343	2401	16807				
$v = 8$	8	64	512	4096					

Table 13.2: The size of full lookup-tables for v=3 to 8, against k, showing the limits of k in DDlab.

The convention in DDLab is to list neighborhood configurations from left to right in reverse n-ary value order[3] so the all-max neighborhood is on the left. Each neighborhood is assigned an output [0,1,..v-1], and the resulting n-string defines one of the v^{v^k} possible rules. The rules may also be expressed in decimal (if applicable – section 16.6) or in hexadecimal. An example for a v=3, k=3 rule is shown below, where the central row shows all possible neighborhoods, the top row their trinary values (the rcode index), and the bottom row the rcode itself,

```
 26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
222 221 220 212 211 210 202 201 200 122 121 120 112 111 110 102 101 100 022 021 020 012 011 010 002 001 000
  0   0   2   1   2   1   0   1   2   2   1   2   1   1   0   2   1   1   0   0   0   2   0   2   2   0   1
```

DDLab treats the n-string as a bitstring, assigning the minimum number of bits for each value; 1 bit if v=2, 2 bits if v=3 or 4, and 3 bits if v=5 to 8. The resulting bitstring can be expressed in hexadecimal, in this case the rule is 026469949408a1.

In DDLab the neighborhood configurations are rotated and displayed verticaly for compactness making a so called "neighborhood matrix", showing also the k-index. A n-ary string can be assigned in the corresponding order to make the rcode.

[3] This can be inverted – section 18.3

```
           26.........................0 - rcode index
            |                          |
        2 - 222222222111111111000000000
k index 1 - 222111000222111000222111000
        0 - 210210210210210210210210210
            |||||||||||||||||||||||||||
            002121012212110211000202201 - output string = rcode
```

13.5.1 The multi-value neighborhood matrix

In the main sequence of prompts, a graphic of the multi-value neighborhood matrix is displayed (but not for mixed k or in TFO-mode). Figure 13.2 gives examples. The matrix can be rescaled and different parts shown (see section 14.11).

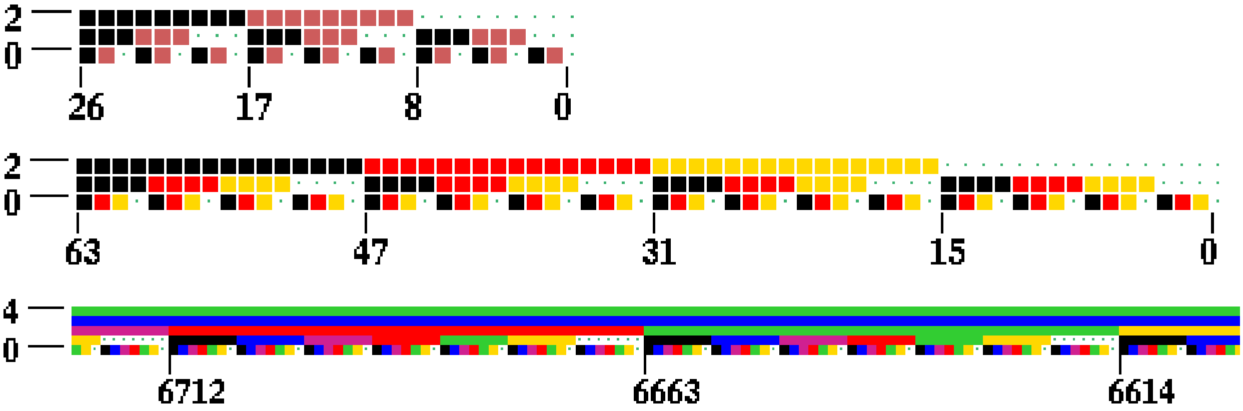


Figure 13.2: Examples of the the multi-value neighborhood matrix, colored according to the value color key (see section 7.1). top: $v3k3$, center: $v4k3$, bottom: $v5k7$ central part only.

13.6 Multi-value totalistic rules, tcode and kcode

For multi-value systems, if $v \geq 3$, totalistic rules seperate into two types: k-totalistic rules (kcode) t-totalistic rules (tcode). For v=2, kcode and tcode are identical.

13.6.1 k-totalistic rules - kcode

```
               27..........................0 <--kcode index
                |                          |
           > 2: 6554443333222221111110000000  < frequency strings       6    0
    values > 1: 0102103210432105432106543210  < of 2s, 1s, 0s,     from 0 to 0
           > 0: 0010120123012340123450123456  < shown vertically        0    6
                ||||||||||||||||||||||||||||
                0022000220022001122200021210 <--kcode, outputs [0,1,2]
```

The kcode lookup-table is a list of the outputs for all possible combinations of value-frequencies in the neighborhood. The combinations are represented by a string of length v (shown vertically from

v-1 down) giving the frequencies of the values v-1 to 0, which must add up to k, so the last row of frequencies is redundant and could be omitted. The ordering of the pattern strings themselves depend on their v-ary value, with the higher value on the left. This can be effectivly inverted to allow kcode expressed in the opposite convention to run as intended (section 31.2).

In the example above for the $v3k6$ "Beehive rule"[33] (also shown as a matix in section 13.6.2) the kcode with outputs [0,1,2] are listed in reverse order of the kcode index. The kcode may also be expressed in decimal (if applicable – section 16.6) or in hexadecimal. If DDLab is not in TFO-mode, it automatically transforms the kcode into the full lookup-table, rcode (section 13.5).

k-totalistic rules are isotropic because they depend only on the frequency of each value (or color) in the neighborhood – the positions of values irrelevant, so that symmetric space-time pattern must conserve their symmetry.

The size of kcode lookup tables, $S = (v + k - 1)!/(k! \times (v - 1)!)$, are much shorter than full lookup-tables. S for different v and max-k allowed in DDLab, is shown in table 13.3 below. Table 13.4 shows all sizes of S for increasing with v and k.

v	max-k	kcode size
2	25	26
3	25	351
4	25	3276
5	25	23751
6	17	26334
7	13	27132
8	11	31824

Table 13.3: kcode lookup-table size for v and max-k available in DDLab.

v \ k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
2:	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
3:	3	6	10	15	21	28	36	45	55	66	78	91	105	120	136	153	171	190	210	231	253	276	300	325	351
4:	4	10	20	35	56	84	120	165	220	286	364	455	560	680	816	969	1140	1330	1540	1771	2024	2300	2600	2925	3276
5:	5	15	35	70	126	210	330	495	715	1001	1365	1820	2380	3060	3876	4845	5985	7315	8855	10626	12650	14950	17550	20475	23751
6:	6	21	56	126	252	462	792	1287	2002	3003	4368	6188	8568	11628	15504	20349	26334								
7:	7	28	84	210	462	924	1716	3003	5005	8008	12376	18564	27132												
8:	8	36	120	330	792	1716	3432	6435	11440	19448	31824														

Table 13.4: The size of kcode-tables for v=3 to 8, against k, showing the limits of k in DDLab.

13.6.2 kcode v=3 matrix

$$\begin{array}{cc|ccccccc}
 & & & & & & j & & \\
 & & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
 & 0 & 0 & 1 & 2 & 1 & 2 & 0 & 0 \\
 & 1 & 0 & 2 & 2 & 2 & 1 & 1 & \\
 & 2 & 0 & 0 & 2 & 2 & 0 & & \\
i & 3 & 0 & 2 & 2 & 0 & & & \\
 & 4 & 0 & 0 & 2 & & & & \\
 & 5 & 2 & 0 & & & & & \\
 & 6 & 0 & & & & & &
\end{array}
\qquad
\begin{array}{cc|cccccccc}
 & & & & & & j & & & \\
 & & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
 & 0 & 0 & 1 & 2 & 1 & 2 & 2 & 2 & 2 \\
 & 1 & 0 & 2 & 2 & 1 & 2 & 2 & 2 & \\
 & 2 & 0 & 0 & 2 & 1 & 2 & 2 & & \\
i & 3 & 0 & 2 & 2 & 1 & 2 & & & \\
 & 4 & 0 & 0 & 2 & 1 & & & & \\
 & 5 & 0 & 0 & 2 & & & & & \\
 & 6 & 0 & 0 & & & & & & \\
 & 7 & 0 & & & & & & &
\end{array}$$

Figure 13.3: Matrix representations of v=3 k-totalistic rules. Left: $v3k6$ Beehive rule[33] and Right: $v3k7$ Spiral rule[34], both on a 2d network with hexagonal tiling[33, 34].

For ternary (v=3) systems, kcode can usefully be show as a 2d matrix[34], where the output state of each neighborhood is given by the row-index i (the number of neighbors with value=2) and column-index j (the number of neighbors with value=1). The number of neighbors with value=0 is given by $k-(i+j)$ so is not required. Two examples are given in figure 13.3 for k=6 and k=7.

13.6.3 t-totalistic rules - tcode

t-totalistic rules relate only to the sum of all the values in the neighborhood. The rules are not necessarily isotropic (as in kcode, section 13.6.1) because different sets of values can make up the same total.

The size of a tcode lookup-table, $S=k(v-1)+1$, is much shorter than kcode (section 13.6.1), but max-k in DDLab is nonetheless is set at the same level as for kcode. S for max-k and v is shown in table 13.5 below,

v	max-k	kcode size
2	25	26
3	25	51
4	25	76
5	25	101
6	17	86
7	13	29
8	11	78

Table 13.5: tcode lookup-table size for v and max-k available in DDLab.

To construct tcode, each total, set out in reverse value order, S-1 to 0, is assigned an output [v-1,v-2,..0], which is the tcode value-string. Here is an example for $v7k5$.

```
30.............................0 - totals
 |                             |
 6301462664052202461530202656513 - tcode, outputs [0,1,2,3,4,5,6]
```

The tcode may also be expressed in decimal (if applicable – section 16.6) or in hexadecimal, as in section 13.5. If DDLab is not in TFO-mode, it automatically transforms the tcode into the full rule-table format (section 13.3).

13.7 Outer-totalistic rules

TFO-mode only, and not for mixed-k

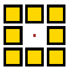
Outer-totalistic rules apply in TFO-mode. The rules (kcode or tcode) depend on the value of the center cell, so a number of rules, v, need to be specified.

If **o** for kcode or **to** for tcode is entered at the prompt in section 13.1.2, the rules will subsequently be assigned in the same way as for a rule-mix (chapter 14), but only to the first v cells in the network (network index 0 to $v-1$) representing the values of the target cell, which will determine the rule to be applied. So this method just utilises the rule-mix functionality, it is not a proper rule-mix.

The method works with any k, but makes most sense if the central cell is not wired to itself - is empty in the neighborhood (see section 10).

For the classic binary game-of-Life (see section 14.2.2), two rules are required, and can be set “by hand” as shown below,

for v=0: 000001000 - birth: exactly 3 live neighbours
for v=1: 000001100 - survival: 2 or 3 live neighbours



However, there are automatic methods for setting the game-of-Life and other Life-like rules, in both outer-kcode (section 14.2.2) and as a full lookup-table, rcode (section 16.10).

13.8 Reaction-Diffusion dynamics

Reaction-Diffusion or excitable media dynamics[7], can be generated with a type of CA or DDN with 3 cell qualities: resting, excited, and refractory (or substrate, activator, and inhibitor). There is usually one resting type, one excited type, and one or more refractory types. In DDLab these correspond to the values $0, 1$, and values≥ 2, which cycle between each other[4].

A resting cell becomes excited if the number of excited cells in its neighborhood falls within the threshold interval (t). An excited cell type changes to refractory. A refractory cell type changes to the next refractory type (if there are more than one) and the final refractory type changes back to resting, completing the following cycle,

$$\text{resting(0): if within (t)} \rightarrow \text{excited(1)} \rightarrow \text{refractory}(2 \rightarrow 3 \rightarrow \cdots \rightarrow v-1) \rightarrow \text{resting(0)}$$

In DDLab, reaction-diffusion can be set as outer-kcode in TFO-mode, which allows greater $[v, k]$, or as a full lookup-table – rcode.

The resulting dynamics in 2d can produce waves, spirals and related patterns that can resemble the BZ reaction and other types of excitable media. As well as the threshold interval, the dynamics is sensitive to the initial state and its density of non-resting types (non-zero values) – usually low for best results. This density (λ parameter) can be set in section 16.3.1. Interesting results can be achieved not only for 2d CA, but also for 2d DDN were the random wiring is confined to a tight local zone (section 12.5.2 and figure 12.3a) as illustrated in figure 13.4.

13.8.1 Reaction-Diffusion from outer-kcode

Select outer-kcode (section 13.7) by entering **o** in section 13.1.2. Subsequent top-right prompts allow the selection of reaction-diffusion dynamics (see section 14.2.1).

The required rules are automatically assigned to the first v cells of the network (network index 0 to $v-1$) in a quasi-outer-kcode[5]. The way this works is as follows, where resting=0, excited=1, refractory≥2: kcode(0) holds the threshold interval to change the cell from resting to excited (0→1). kcodes(1, 2, 3,.., v-1) store the colors, so whatever the neighborhood, the cell cycles to the next color, then back to 0, as described in section 13.8.

[4] For binary networks (v=2) the refractory type would be missing, so this would not be reaction-diffusion in its proper sence, though the option is still present in DDLab.

[5] As noted in section 13.7, outer-totalistic rules are themselves a quasi-rule-mix, using the rule-mix functionality.

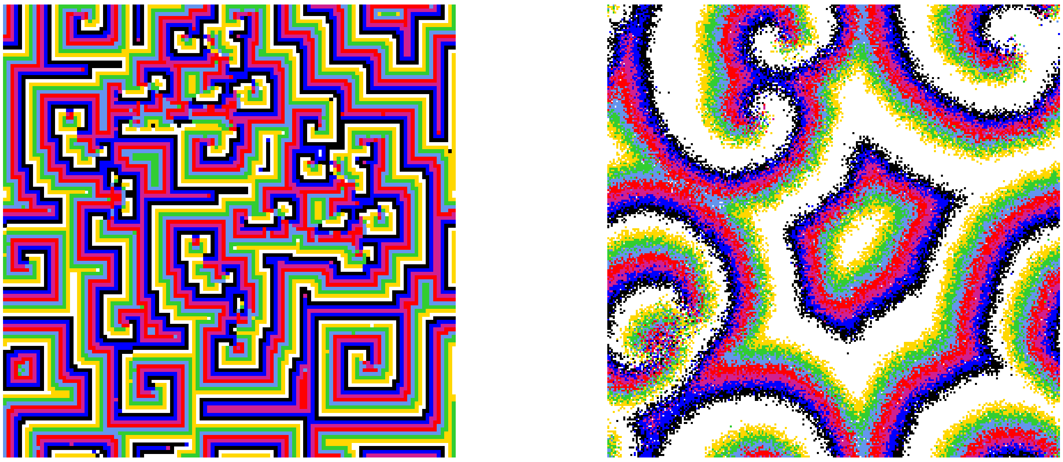


Figure 13.4: Reaction-Diffusion dynamics. Left $v8k8$, threshold interval is 1 to 6, 122x122 square grid. Right: $v8k11$, threshold interval is 2 to 7, 255x255 square grid, random connections within a 24 diameter local zone.

13.8.2 Reaction-Diffusion from a full lookup-table – rcode

Select rcode which impliments any logic, by entering **return** in section 13.1.1. Then enter **R** (. . . **RD-R**. . .) at the next main sequence prompt (section 16.1.1) for a reaction-diffusion rcode. After the threshold is set in section 13.8.3 below, the rule will be set automatically to emulate reaction-diffusion dynamics. Note that a reaction-diffusion rcode can be part of a rule-mix.

13.8.3 Selecting the threshold interval

Once reaction-diffusion dynamics has been selected (in sections 13.8.1 or 13.8.2) a subsequent top-right prompt sets the threshold interval. This example shows the threshold interval set between 1 and 6 in a k=8 network,

thresholds: lower (0-8):1   upper (1-8):6 *(if v=8)*

This would make a resting cell change into an excited cell if there were between 1 and 6 excited cells in its outer neighborhood, otherwise it would remain at rest.

Chapter 14

Rule-mix options

All cells in a network may have the same rule as in classical CA, or cells may have different rules. This is referred to as a rule-mix, which may a rcode-mix, kcode-mix or tcode-mix. If the network has heterogeneous neighborhoods sizes (a k-mix, described in chapter 9 and section 11.6.2) then there must be a rule-mix, which is set by default – in this case section 14.1 would not apply. For just one rule enter **return** at the prompt in section 14.1 and continue with chapter 16.

This chapter describes the rule-mix options, and also outer-totalistic rules in TFO-mode which share the same methods. Outer-kcode provides alternative methods for reaction-diffusion and Life-like dynamics[1].

A rule-mix can be assigned from ...

- the whole of rule-space (rule-spaces for a k-mix)
- a limited subset of rules to restrict the range of rules in the rule-mix. The number of rules in the subset can be anwhere from 1 to n (system size), but with an upper limit of 200 rules. This subset option does not apply to a k-mix, or to outer-totalistc rules in TFO-mode.

Rules in a rule-mix or a limited subset can be set automatically at random with a pre-defined density-bias of non-zero outputs (λ-parameter), and rules can be confined to various sub-categories. Alternatively rules can be set individually "by hand" allowing almost any combination and the rule-mix can be loaded from a previously saved file.

This chaper also describes altering the presentation of the neighborhood matrix (section 14.11), and listing Post-functions, which are included in the initial rule-mix prompt.

14.1 Single rule or rule-mix, and other options

not for a k-mix (unless all k's equal, section 14.6.7), or outer-totalistc rules in TFO-mode

After selecting a rule type in section 13.1, the next top-right prompt selects a single rule or a rule-mix, and includes some other options. The top line of the prompt shows a reminder of the current mode, FIELD, SEED or TFO (section 6.1) and an option to preset the density-bias (λ-parameter). The prompt also differs somewhat according to the rule type,

[1] Reaction-diffusion rules or Life-like rules can also be set as a full lookup-table, which allows inclusion in a rule-mix.

For rcode, kcode and tcode based on full lookup-tables selected in section 13.1.1

FIELD mode, bias-density (def 50.00%)-s: *(or SEED mode, def 50.00% – for v=2)*
RULES: single rcode (def), load rulemix-l, list Post-P, nhood-matrix-a
mix: no limit-n, or set limit up to 200: *(or single tcode/kcode above)*

In TFO mode, based on short lookup-tables selected in section 13.1.2

TFO mode, bias-density (def 83.33%)-s: *(or def 83.33% – for v=6)*
KCODE: single kcode (def), load kcodemix-l *(or TCODE: single tcode..)*
mix: no limit-n, or set limit up to 200:

Note that the maximum size of the limited subset of rules (**set limit**) is 200, unless the system size $n < 200$, in which case the maximum **set limit** is n.

14.1.1 Density-bias (λ-parameter)

Enter **s** in section 14.1 to change the density-bias – the fraction of non-zero values in the rcode-table (the λ-parameter) but expressed as a percentage. This is applied probabilistically, so each rule in a rule-mix will not necessarily have exactly the given density-bias. The following top-right prompt is presented,

density-bias: enter % (def 66.67%):
(for v=3, the default is an equal probability of each value)

Enter the new percentage density-bias – the prompt in section 14.1 will reappear with the revised setting. The density-bias will be applied when setting the following at random,

- single rules (section 16.3.1).
- a rule-mix or subset of rules (sections 14.5, 14.4.2).
- a rule-mix or subset of rules "by hand" (section 14.6).

Note that for single rules set at random, the density-bias can be reset, either exactly or probabilistically at later stages in DDLab (section 16.3.1).

14.1.2 Summary of rule-mix and other options

Enter one of the following at the promp in section 14.1,

- **return** for a single rcode, kcode or tcode, so *not* a rule-mix, skipping everything else in this chapter. See chapter 16 for further options.
- **l** to load a `.r_s` rcode-mix (rule-scheme), `.r_v` for a kcode-mix, and `.r_t` for a tcode-mix. Chapter 19 explains the loading prompts.
- **P** to list Post functions, section 14.12 (not in TFO-mode).
- **a** to amend the presentation of the neighborhood matrix, section 14.11 (not in TFO-mode).
- **n** for a rcode-mix, kcode-mix or tcode-mix, taken from the whole of rule-space (section 14.4.1).
- a number (1 to 200) or (1 to n if $n < 200$) to set the size of a limited subset of rules (rcodes, kcodes or tcodes) from which the mix will be assigned to the network (section 14.4.2).

14.2 Outer-totalistic kcode or tcode

TFO-mode only

If outer-totalistic (kcode or tcode) was selected in TFO-mode in section 13.1.2, a quasi-rule-mix with v rules is the automatic choice, and a top-right reminder similar to this is presented,

> **TFO-mode,**
> **OuterKcode: number of kcodes to be set=value-range=3**
> **cont-ret:** *(or OuterTcode/tcodes, values shown are examples)*

The next prompt will be similar to setting a rule-subset (KcodeSet or TcodeSet) in section 14.4.2, but where the subset size=v. The prompts are as follows,

> *for kcode (and v=8)*
> **OnterKcode(8): select by hand-h, RD-R Life-L maj-m/u Alt-A rnd-(def):**
> *for tcode (and v=4)*
> **OuterTcode(4): select by hand-h, maj-m rnd-(def):**

The rules will be allocated to the first v cells in the network (index 0 to $v-1$) representing the values of the target cell, which determines the rule. The rules are set either at random with various biases, or by hand, as for a normal rule-mix, but although the method utilizes the rule-mix functionality, this is not a proper rule-mix (see section 13.7).

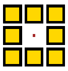
14.2.1 Setting reaction-diffusion by outer-kcode

for reaction-diffusion from a full lookup see section 13.8.2

Enter **R** in the **OuterKcode**... prompt in section 14.2 above to initiate a reaction-diffusion rule by outer-kcode (see sections 13.8, 13.8.1 and figure 13.4). Then set the threshold interval as described in section 13.8.3. An alternative method for reaction-diffusion is from the full lookup-table – rcode, which also allows reaction-diffusion rules to form part of a rule-mix, see sections 13.8.2.

14.2.2 The game-of-Life and other Life-like rules by OuterKcode

for Life from a full lookup table see section 16.10

If k=8 on a 2d square lattice, giving this neighborhood  – John Conway's game-of-Life[5] can be set as an outer-kcode. Any other Life-like[2] rule from the "Life family" can also be set, as well as Life-like rules with neighbohoods other than the Moore neighbohood, or with $v > 2$. This can also be done as a full lookup table in section 16.10, but the outer-kcode method provides a greater range of v and k.

To set the classic game-of-life (but with possibly v colors), enter **L** in section 14.2, the following top-right prompt (for k=8) appears,

[2] http://en.wikipedia.org/wiki/Life-like_cellular_automaton

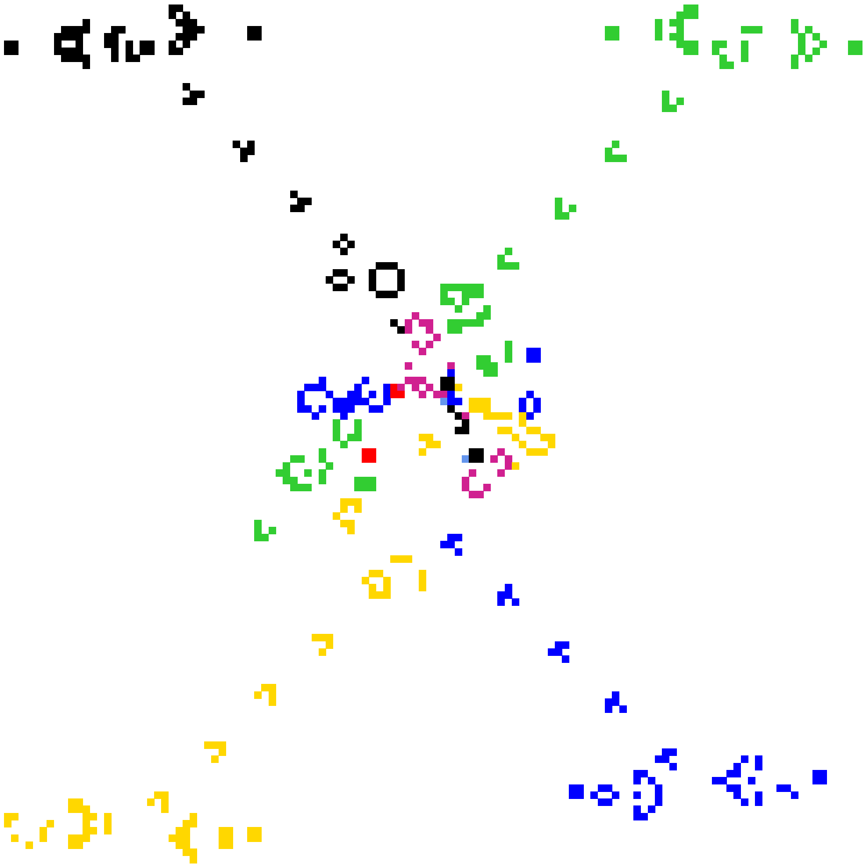


Figure 14.1: The game-of-Life (23/3) appled to a v=8 CA. The algorithm in DDLab generates 8 outer kcodes giving the same dynamics as classical binary Life, but including 7 colors + background. In this example 4 glider guns are located in each corner on a 122×122 lattice, shooting different colored gliders tovards the centre.

Life k=8 (def: survival 2,3, birth 3,) accept-ret amend-a:

Enter **return** to accept the default. If **a** is entered to amend, the following further prompts are presented,

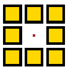
accept-ret, or enter number+ret, max entries=9, max value=8:
enter survival (def=2.3,): *followed by ...*
enter birth (def=3,):

Enter the new values for survival, followed by the new values for birth. After each number enter **return** for the next number. There may be up to k entries – their order, or repeats, are not significant. **return** without a number concludes the entries. **q** reverts to the first Life prompt, but with the defaults possible altered.

The Life option is available for $v \geq 2$, and any k as well as the k=8 neighborhood, for 1d and 3d as well as 2d, and for a rule-mix by hand. For $v > 2$, for a (23/3) Life setting, the algorithm in DDLab generates an equivalent lookup table giving the same dynamics as the classical binary Life, but including v-1 colors plus the background, as in figure 14.1.

As well a using the promps above, the game-of-Life as an outer totalistic rule, can also be set "by hand". For example, select v=2, k=8, and a 2d square lattice. Then in section 13.1.2 select outer-totalistic (kcode or tcode). Two outer totalistic rules need to be set (because v=2), the first for a central cell of 0, the second for a central cell of 1. Select 'by hand", then select **b** for bits in the single rule prompt, similar to prompt 14.6.2 or 16.1.2. Set each rule (kcode or tcode) in turn as below,

for v=0: 000001000 - birth: exactly 3 live neighbours
for v=1: 000001100 - survival: 2 or 3 live neighbours



← k=8 outer-neighborhood

14.3 Setting the neighborhood, k=0

If a k-mix was selected in section 9.1 or 11.6.2, DDLab identifies any self-wired k=1 cells, and gives an option to set the rule (or kcode or tcode) at these cells to make them effectivly k=0 (see section 9.3). The following prompt is presented,

single self wirings=4, set rule for k=0 equiv -0:
(shows the number of k=1 self-wired rules found)

Enter **0** accept.

Note that if the effective wiring to a cell is k=0, it will appear disconnected in the 1d wiring graphic (see section 17.5), as illusrtated in figure 14.2.

Figure 14.2: Effective neighborhood k=0, as shown in the 1-d wiring graphic (see section 17.5)

14.4 Methods for setting the rule-mix or rule-subset

A rule-mix, rule-subset (or quasi-rule-mix) will be set if,

- **n** for *no limit* was selected in section 14.1 – this sets a rule-mix
- a number 1-n (upper limit 200) for as rule-subset was selected in section 14.1 – this sets the subset size.
- the network has mixed-k (set in chapter 9 or section 11.6.2) – this forces a rule-mix.
- outer-totalistic was selected in section 13.1.2, followed by the outer-totalistic reminder in section 14.2 – this sets a quasi-rule-mix of v rules.

There are two strategies for creating a simple rule-mix.

- The direct strategy (the only one for a k-mix, or enter **n** for *no limit* in section 14.1) - allows any rule from rule-space(s) to be included, so if randomly assigned a repeat of the same rule is unlikely for a large rule-space.
- The indirect strategy (enter a number: 1-n, upper limit 200). This first creates a limited subset of rules from which rules will be assigned at random to the network, so the final rule-mix can be restricted to just a few rules - or just one rule (see section 14.4.3).

The options below, which differ only slightly between the direct and indirect strategies, allow rules to be set automatically at random (section 14.5) with various constraints depending on the type of rule, or individually (by hand) by the various methods described in chapter 16.

14.4.1 Setting a rulemix directly

If **n** for *no limit* was selected in section 14.1, or for a k-mix, the rulemix will be assigned directly to the network with the following prompts, which depend on the type of rule.

For full rule-tables and rcode, the prompt is,

> **RuleMix: select by hand-h, maj-m/u Alt-A chain-c iso-i Post-P canalyzing-C/+C, rnd-(def):**

In TFO mode, or if kcode or tcode was selected for full lookup-tables in section 13.1.1, the prompt is,

> **KcodeMix: select by hand-h maj-m/u Alt-A rnd-(def):** *(for kcode)*
> *or*
> **TcodeMix: select by hand-h rnd-(def):** *(for tcode)*

14.4.2 Setting a rulemix indirectly - specify a rule-subset

does not apply to a k-mix

A number (1 to 200) or (1 to n if $n < 200$) selected in section 14.1 specifies the size of the subset of rules. Rules from this subset will be automatically assiged at random to the network. The prompts are similar to section 14.4.1 but with some prompts missing – **maj-m/u**, **Alt-A** and **canalyzing**.

For rcode the prompt is,

> **RuleSet(22): select by hand-h** *(for a subset size 22)*
> **Alt-A chain-c iso-i Post-P rnd-(def):**

In TFO mode, or if kcode or tcode was selected for full lookup-tables in section 13.1.1, the prompts are,

> **KcodeSet(5): select by hand-h, Alt-A rnd-(def):** *(for kcode, for subset size 5)*
> *or*
> **TcodeMix(33): select by hand-h rnd-(def):** *(for tcode, for subset size 33)*

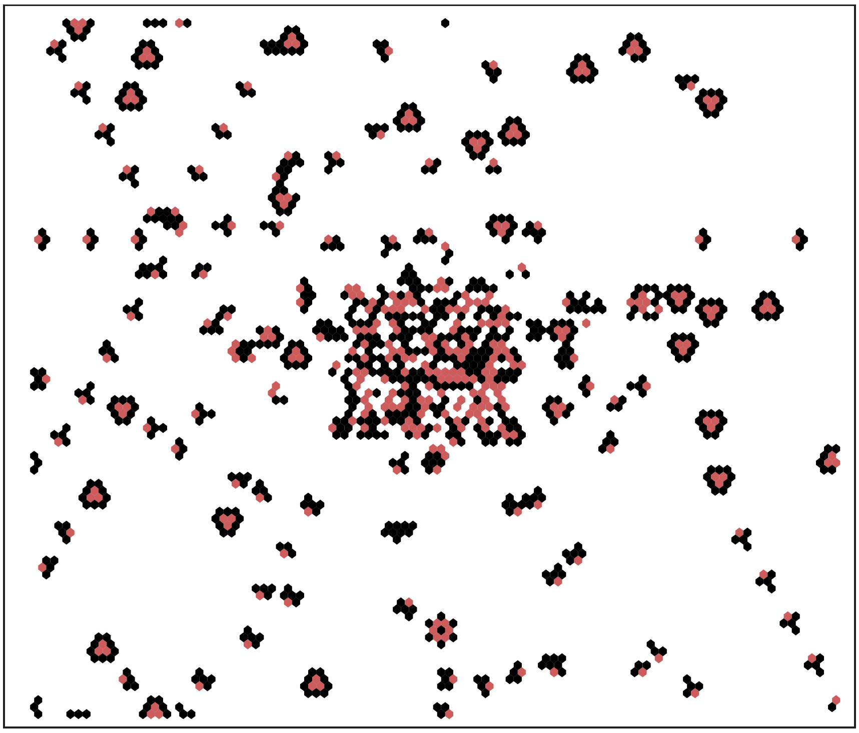


Figure 14.3: A 2d CA set up with the spiral rule[34] 100×100 perturbed by a central 20×20 homogeneous block of chain rules that have been inserted. To do this first set up as a ruelmix consisting of just one rule, then create a rule block (see sections 17.6.5 and 17.8.10).

14.4.3 A rule-mix with just one rule

To set up any network (CA, RBN or DDN) with a homogeneous rule that allows later additions and revisions as if it had a rule-mix (as in figure 14.3), select a rule-mix with a limited subset of rules consisting of just one rule, i.e. enter **1** at the prompt in section 14.1.2, and set the rule *by hand* (section 14.6). This fools DDLab into assuming the network has a "rule-mix" when in fact all rules are the same.

14.5 Rule-mix - random

The following options in section 14.4 assign rules either entirely at random, or at random from some subcategory of rules. The program continues with options to review network architecture described in chapter 17.

option ...	*what it means*
rnd-(def) ...	Enter **return** to assign rules entirely at random from the whole of rule-space, for rcode, kcode or tcode. The density-bias (λ-parameter) can be preset in section 14.1.1
maj-m/u ...	Enter **m** to assign majority (voting) rules, see section 14.7. Enter **u** (*not for tcode*) to assign majority rules but with the outputs from uniform neighborhoods flipped or shifted.

does not apply to tcode

Alt-A . . . Enter **A** to assign rcode or kcodes at random from the subset of "Altenberg" rules, see section 16.9.

for rcode only

chain-c . . . Enter **c** to assign rcode at random from the subset of "chain" rules, see section 16.11.

iso-i . . . Enter **i** to assign "isometric" rcode at random, where rotated and reflected nhoods (in 1d, 2d and 3d) have the same output.

Post-P . . . Enter **P** to assign rcode at random from the subset of "Post functions" (section 14.12). A prompt to restrict the Post value will first be presented as in section 14.12.2.

canalizing-C/+C . . . Enter **C** to see and amend the canalising inputs in the rcode-mix, or enter another subcategory followed by **C**: **mC**, **uC**, **AC**, **cC**, **iC**, or **PC**. Canalizing inputs are described in chapter 15.

14.6 Rule-mix by hand

A succession of rules can be set individually, by hand, to create a rule-mix, rule-subset, or outer-totalistic. Each rule is set just like a single rule in chapter 16 – all the relevant options apply depending on the rule type.

A top-right prompt gives the current rule index (starting with 0), and allows jumping to any other valid index, and changing the current default method of rule selection. The selected rule and other details (and options) for the rule will appear as for a single rule in chapter 16. At any stage, the remaining rules can be set at random with valid biases.

14.6.1 by hand reminder

If **h** is entered in 14.2 or 14.4, a top-right reminder appears, for example,

by hand: subset of 22 rcodes *(or kcode, tcode)*
or
by hand: kcodemix of 150 kcodes *(or rcodes, tcodes)*
or for outer-totalistic
by hand: outertot of 6 kcodes *(or tcodes)*
or for a kmix – lower and upper values of k are indicated
by hand: rcodemix of 150 rcodes (kmix 3-6) *(or kcodemix, tcodemix)*

14.6.2 by hand single rule prompt

At the same time as the "by hand" reminder above (section 14.6.1), the main sequence single rule prompt appears (heavily context dependent – see section 16.1) to accept or revise the default method of rule selection (and other options), for example,

```
Select v2k5 rcode (S=32): empty-e fill-f maj-m Alt-A life-L chain-c rnd-r
ReDiff-R iso-i bits-b hex-h dec-d rep-p load-l values-v prtx-x (def-r):
```

Enter **return** to accept the default rule selection method, or change it with any another valid entry, and set the rule for the current rule index. The rule selection methods are described in detail in chapter 16 and summarised in section 16.2.

14.6.3 by hand options

The rule at each cell (or subset, or outer totalistic) index is set in turn, starting from index 0. A top-right prompt keeps track of the rule index, the current value of k, and the current selection method, for example,

for a k-mix
index=7 k(3-9) (v3k4 rcodemix), same as last-s, all remaining-a:
(if a rule has already been set for the current k, otherwise)

index=7 k(3-9) (v3k4 rcodemix) *(or kcodemix, tcodemix)*
hand=r, change-(any valid entry) or index-q: *(**r** is the current selection method)*

for homogeneous k
index=12 (v3k4 rcodemix)
hand=A, change-(any valid entry) or index-q, rnd all-a:
([rnd all-a] if hand=[r, m, A, c, i] only)

Enter **return** to set the rule, kcode or tcode, according to the current selection method, possibly with intermediate steps. The methods are similar to setting a single rule in chapter 16. Once set, each rule (in bits/values and hex) may appear below the single rule prompt (section 14.6.2) and in the bottom "rule window" (section 16.19). Enter "any valid entry" or **q** to change the current selection method or index – see section 14.6.4.

For a k-mix enter **s** or **a** to copy the rule (see section 14.6.6). For homogeneous k enter **a** to complete random rules automatically (see section 14.6.5).

14.6.4 change the selection method or rule index

To change the selection method,

- enter any valid entry from prompt 14.6.2.
- enter any valid entry (from prompt 14.6.2) in prompt 14.6.3.
- enter **q** in prompt 14.6.3, which also allows jumpimg to a previous cell index.

If **q** is entered in prompt 14.6.2, the following top-right prompt appears (for example),

change: index(16)-i, start again-a, selection-s, cont-ret:

Enter **a** to restart, from index 0.
If **i** is eneterd above, the following top-right prompt appears (for example),

this index=16, enter new index 0-16:

Enter a number ≤ to the current index to change the rule index.

If **s** is entered in **selection-s** above, the following top-right prompt appears, where the lower line depends on the type of rule and k, and corresponds to the main sequence single rule prompt 14.6.2 (see also section 16.1), for example,

> **index=16 (v5k4), change selection method (now r), abort-q**
> **as main sequence choises - e/f/x/m/u/A/L/c/r/i/R/b/h/d/p/l/v:** *(for rcode)*
> *or*
> **as main sequence choises - e/f/x/m/u/A/r/b/h/d/p/l/v:** *(for kcode)*
> *or*
> **as main sequence choises - e/f/x/m/r/b/h/d/p/l/v:** *(for tcode)*

Enter any one of the listed choises to change the selection method which becomes the new default. These methods are described in detail in chapter 16 and summarised in section 16.2.

14.6.5 complete the rule-mix automatically

For a rule-mix with homogenious k, and if the current selection method is **hand=r, hand=m, hand=A, hand=c, or hand=i**, where the rules are set at random but with constraints (see section 16.2), the following option is included in prompt 14.6.3

> ... **rnd all-a:** *(if hand def=[r, m, A, c, i] only)*

Enter **a** to abandon setting rules by hand, and complete setting rules at random automatically for the rest of the network according to the current method and constraints, for example the density-bias (λ-parameter) if **hand=r** (see section 16.3.1).

14.6.6 copy rules automatically for a k-mix

While setting rules by hand for a k-mix, an optinon allows a rule set previously with a given k to be copied, or automatically copied to all the remaining cells in the network with the same k.

The following options are included in prompt 14.6.3

> ... **same as last-s, all remaining-a:** *(if a rule for the current k was set previously)*

Enter **s** to copy once, or **a** to copy to all remaining cells with the same k, or enter **return** to ignore this option and continue.

14.6.7 Mixed k where all k's (and rules) are the same

To create a k-mix network where all k's are the same (k=3 in this example), at the prompt for setting the k-mix in section 9.4,

> **set k-mix: load-l hand-h specify-s rnd-(def):**

select **s** to set the percentage of k's required (section 9.8.1), then enter **return** until k=3, then set 100%,

enter % k=3 (100.0% left) back-b:100

To make it possible in the future to increase k for some cells, k-max can be set at some higher value (section 9.11) for example,

. . . **set greater max k-max (def 3, limit 9):7**

After further options, DDLab will recognise if all k's are the same, and present the single rule or rule-mix prompt (section 14.1) – for a single homogenious rule enter **1**.

14.7 Rule-mix - majority

If **m** is selected, in section 14.4.1, a "majority" (voting) rule will be set at each cell. For v=2 the majority rule outputs 1 for a majority of 1s in the neighborhood, 0 for a majority of 0s. In the case of even k, and a tie between 1s and 0s, the output is set according to the value of the central neighborhood index, as defined in chapter 10.

For $v > 2$ the majority rule outputs the majority value in the neighborhood. In the case of a tie one of the most frequent values is chosen at random.

14.8 Rule-mix - majority with shifted uniform outputs

not for tcode – see also section 16.8 for a single rule

If **u** is selected in section 14.4.1, "majority" (voting) rules are set as in section 14.7 above, but the uniform (unanimous) neighborhoods have their outputs shifted by -1, except for 0 which becomes v-1.

For random wiring, this can result in some interesting bi-stable, tri-stable, v-stable, dynamics, as in figure 14.4. For v=2 this is the same as flipping the "end bits" (all-0s and all-1s), as before in binary DDLab (figure 16.8).

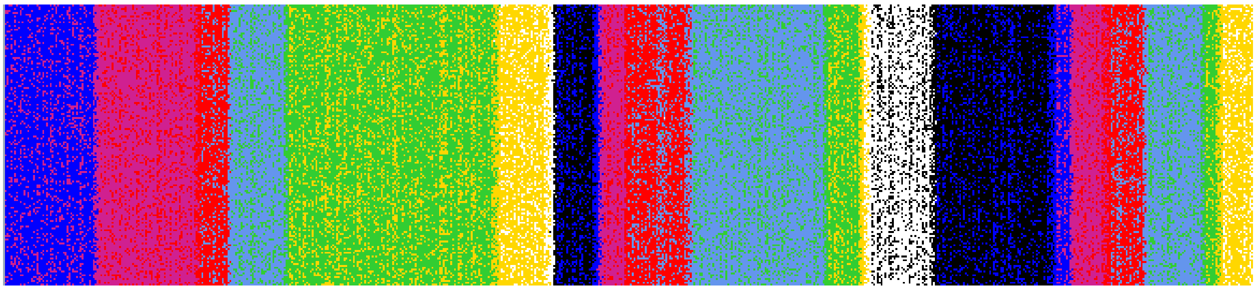


Figure 14.4: Shifted majority kcode-mix with random wiring, $v8k11$, where uniform neighborhoods have their outputs shifted to a different color. A n=150 1d space-time pattern (rotated by 90° so the time flows from left to right) shows zones of color density that remain stable for an unpredictable number of time-steps before flipping to a different quasi-stable regime.

14.9 Rule-mix for large networks, or large k

Assigning the rule-mix to large networks, especially for large k, may take some time. For $n \geq 5000$ or ($n \geq 500$ and $k \geq 10$), the assignment is monitored by a progress bar near the top right hand corner of the screen. When the assignment is complete, the program will continue with options to review network architecture described in chapter 17.

14.10 The all 0s output

A rule-mix set at random may be biased to have the "all 0s" neighborhood output in every rule reset to 0 – the default for rcode, or any rule may be allowed – the default for kcode and tcode. The following prompt is presented,

for rcode-mix
rcodemix: allow any rule-a, or force all 0s->0-(def):

for kcode-mix or tcode-mix
kcodemix: force all 0s->0-0:

For rcode, enter **return** to force all 0s->0, or **a** to leave the rule-mix as is. For kcode or tcode, enter **0** to force all 0s->0, or **return** to leave rule-mix as is.

In a locally wired network, forcing all 0s->0 ensures that a small zone of non-zero cells can grow at no more than the network's "speed of light", otherwise non-zero cells can appear at the first time-step throughout the network.

14.11 Amending the neighborhood matrix

for full lookup-tables only, not in TFO-mode

If **a** to amend the neighorhood matrix (k-matrix) presentation (see section 13.5.1) is selected in section 14.1 the following prompts are presented in sequence,

nhood-matrix: exit-q, % of index to show (def 100%):33
scale in pixels (def 2, max 30):8 *(values shown are examples)*

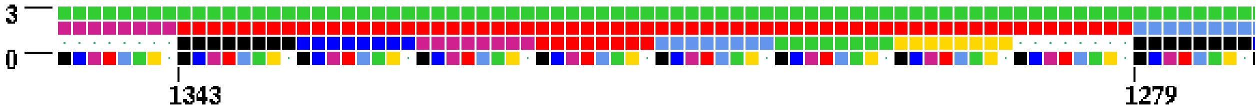


Figure 14.5: Part only of the multi-value neighborhood matrix, colored according to the value color key (see section 7.1), this example for $v8k4$, set at 33%, and scale=8.

Enter the percentage to show just part of the matrix, which sets the maximum rule-table index on the left. Then set the scale in pixels. When the revised neighorhood matrix is displayed as in figure 14.5 below (or 13.2), the **nhood-matrix:...** prompt reappears for further amendment. Enter **q** to exit and continue.

14.12 List Post functions

for full lookup-tables only, not in TFO-mode

Post functions[16] are rules that belong to certain classes of Boolean functions that are closed under composition, and which play a role in the emergence of order in Boolean networks, a "softer" version of canalizing functions. In DDLab, Post functions are generalised for multi-value, but as their theoretical relevance has only been demonstrated for Boolean functions, any multi-value results should be treated with caution.

Very briefly, if u is the Post value from $[v-1, v-2, ..., 0]$ or just $[1, 0]$ in the Boolean case, and the Post class is $[2, 3, 4, ..., i]$, where i stands for *all* or "infinity", a rule-table (function) belongs (non-exclusively) to $A[u]2$, $A[u]3$,..., $A[u]i$, if any [2, 3,..., *all*] neighborhoods where the rule-table output=u have a common u component (DDLab only computes class 2, 3, and i).

In the literature the neighborhood (string) is the "vector", and the rule-table is the "function". By this definition, if a value u occurs just once in the rule-table or not at all, the function belongs to Post class $A[u]i$. These classes are nested; $A[u]i \in A[u]3 \in A[u]2$. For Boolean functions the class $A[u]i$ must also be canalizing. Note also that totalistic rules, kcode and tcode, are much more likely to include Post functions than rules in general.

14.12.1 initial Post-function prompt

```
         _tcode index
        /    _tcode-table (totalistic)
       /    /      _rule-table
      /    /      /       _Post value and class
     /    /      /       /      _Canalizing
    /    /      /       /      /  _P-parameter
   /    /      /       /      /  /  _Z-parameter
  /    /      /       /      /  /  /
15: 1111 11111111 A[0]i   C=3 P=1 Z=0
14: 1110 11111110 A[0]i   C=3 P=0.875 Z=0.25
13: 1101 11101001 A[0]2   C=0 P=0.625 Z=0.75
12: 1100 11101000 A[1]2 A[0]2   C=0 P=0.5 Z=0.5
8: 1000 10000000 A[1]i   C=3 P=0.875 Z=0.25
4: 0100 01101000 A[1]2   C=0 P=0.625 Z=0.75
0: 0000 00000000 A[1]i   C=3 P=1 Z=0
v2k3: Post count=7=43.750\% (2=3 3=0 i=4) all tcode-space=16 (Post only)
                                            \                   \_or (full list)
                                             \_or tcode-sample=x
```

Table 14.1: Post function list and data output for tcode-space $v2k3$, no restriction, with explanatory notes. Only rule-tables ≤128 are displayed. The last line is the data summary.

Enter **P** in section 14.1 to find and list all, or just samples, of Post functions in the terminal window (including data on canalizing, and the P and Z parameters) and/or to save the list to a `.dat` file. A prompt similar to this, depending on the context, is presented,

> **Post functions: sample-s, rcode-space (2^8=256)-p, all-a:** *(for rcode* $v2k8$ *or kcode/tcodes-space, the option* **-p** *is removed if the table-size>* 2^{32}

```
43539:  00011100011100000010000011000000 A[1]2   C=0 P=0.71875 Z=0.507812
41744:  10000100111010000100000010001000 A[1]2   C=0 P=0.71875 Z=0.4375
40536:  01110010111111000100000010000000 A[1]2   C=0 P=0.625 Z=0.5625
38222:  11011101000000000110111000000000 A[1]i   C=1 P=0.65625 Z=0.570312
38161:  01011101110001101000100000000000 A[1]2   C=0 P=0.65625 Z=0.617188
28130:  00000100100000000110000001100010 0 A[1]i   C=1 P=0.78125 Z=0.390625
23020:  00011000011100001100000010000000 A[1]2   C=0 P=0.75 Z=0.5
21167:  10010011000010000110010000000000 A[1]2   C=0 P=0.75 Z=0.5
12990:  11000100000100111000000000000000 A[1]i   C=1 P=0.78125 Z=0.4375
11724:  00101010000000010100000000100010 00 A[1]i   C=1 P=0.78125 Z=0.4375
5419:  00010010010000101000100010000000 A[1]2   C=0 P=0.78125 Z=0.4375
v2k5: Post count=11=0.022\% (2=7 3=0 i=4) rule-sample=50000 (Post only)
```

Table 14.2: Post function list and data output for a 50000 sample of $v2k5$ rcode-space, restricted to Post value 1, the sample index is on the left. For explanatory notes see table 14.1.

```
1553: 2212221212222121211 02  A[0]3   C=0 Z=0.26749
1418: 2000000000002001002 20  A[1]i   C=0 Z=0.152263
1145: 2000000020220220000 20  A[1]i   C=0 Z=0.251029
924: 1221121222121011110 00  A[0]2   C=0 Z=0.419753
656: 0220000220000221000 20  A[1]i   C=0 Z=0.27572
473: 1111222212212112220 21  A[0]2   C=0 Z=0.358025
422: 1212101011010101011 11  A[2]2   C=0 Z=0.366255
414: 0202000002120220020 00  A[1]3   C=0 Z=0.283951
222: 0102120001110010001 01  A[2]2   C=0 Z=0.366255
202: 2222220202210221120 22  A[1]2   C=0 Z=0.407407
197: 0220222022012022022 00  A[1]2   C=0 Z=0.432099
v3k5: Post count=11=0.550\% (2=6 3=2 i=3) kcode-sample=2000 (Post only)
```

Table 14.3: Post function list (multi-value) and data output for a 2000 sample of $v3k5$ kcode-space, unrestricted. Only the kcode-table is shown because the rule-table size>128. The sample index is on the left. For explanatory notes see table 14.1.

Enter **p** for whole of rule-space, only possible if its size$\leq 2^{32}$ (then the size is shown in the prompt). Alternatively enter **s** for just a sample of rule-space. Add **a** in either case (i.e. **pa** or **sa**) to show *all* the functions, not just the Post functions, useful for finding parameter data on rules in general. **q** will exit the prompt, **return** has no effect.

14.12.2 restrict Post-functions

If **a** for all functions was *not* selected in section 14.12.1, a top-right prompt to restrict the Post value is presented,

> **restrict Post value (0-1):** *(for Boolean functions. (0-2), (0-3) etc for multi value)*

Enter the Post value $[v-1, v-2, ...0]$. Enter **return** for no restriction.

14.12.3 set Post-function sample size

If **s** to select a sample was selected in section 14.12.1, a top-right prompt to set the sample size is presented,

> **sample (def 10000):**

Enter the sample size or **return** to accept the default size.

14.12.4 final Post-function prompt

The following top-right prompt, with exact wording depending on the context, is presented to print the data to the terminal, save it to a `.dat` file with (default file name `post_data.dat`), or both,

> **rule sample (2000) (Post only): print xterm-(def), save-s, both-b:**
> *possible alternatives: all rules/kcode/tcodes, rule/kcode/tcode sample, (Post only)/(full list)*

To cut short a lengthy computation in progress enter **q**. When complete, the program returns to section 14.12.1. Enter **q** to exit Post-functions, or start a new listing.

Intentionally Blank

Chapter 15

Setting Canalization in a random rcode-mix

This chapter apples to full lookup-tables only – not TFO-mode.

The dynamics on a random rcode-mix can be biased towards order by including varying proportions of "canalizing" inputs, with applications in modeling genetic regulatory networks[9, 25].

A canalizing input may be defined as follows: if a particular value (0,1,. . . ,v-1) on an input to a cell (referred to as a "gene" in this context) determines the gene's output irrespective of its other inputs, that input is said to be canalizing. A given gene may have from zero to k canalizing inputs – the outputs of all of these must be the same. As k increases, the fraction of rule-space with canalizing rules (having at least one canalizing input) deceases exponentially (section 24.13), so the probability of setting such a rule at random deceases accordingly.

Traditionally, canalization applies to Boolean rules, but the same definition has been generalised for multi-value networks and is now applied in multi-value DDLab. However, the examples in this chapter remain focused on Boolean rules.

The methods bias the randomly assigned rules by varying or tuning the fraction of canalizing rules, or inputs, in the network. The algorithms for tuning canalization in DDLab are designed to minimize secondary biases in the distribution of canalizing inputs. Rule tables will be amended at random for the required degree of canalization.

15.1 Selecting Canalizing

Canalizing can be selected at two alternative places in DDLab.

When setting rcode-mix directly in section 14.4.1, at the prompt,

RcodeMix: select by hand-h, maj-m Alt-A chain-c iso-i Post-P canalyzing-C/+C, rnd(def): *(canalyzing-C/+C not for kcode or tcode)*

Enter **C** to apply canalyzing to random rules, or [**mC, AC, cC, iC**] (but not **PC**) to apply canalyzing to biased random rules. The (biased) rules will be set first, then modified by cumulative random mutation to achieve the required canalization.

Alternatively, once the rule-mix has been set, canalizing can be reached from the wiring graphic prompts described in chapter 17. At the prompt,

```
...
... rule: rev/trans-v/t ...
...
```

Enter **t** to get the *transform rule* options described in chapter 18. A top-right prompt similar to the one below will be presented,

```
transform rule: solid-o invert-v comp-c neg-n ref-r canal-C (all+a)
equiv>k (4-7), max k-m, eff k: all-K this-k, save-s:
```

Enter **Ca** to select canalizing for the whole network, described in this chapter, or just **C** to set canalizing for a single rule described in section 18.5.

15.2 The first canalizing prompt

A series of top-right prompts allow the canalization to be set and reviewed. The first canalizing prompt is as follows,

```
set canalizing genes-g inputs-(def):
```

Enter **g** to specify canalizing genes, i.e. the fraction of network elements, n, with at least one canalizing input.

Enter **return** (the default) to specify canalizing inputs (wires), where the total number of wires is $n \times k$ for homogeneous k, or the sum of all k's for a mixed k network.

15.3 Canalizing percentage or number

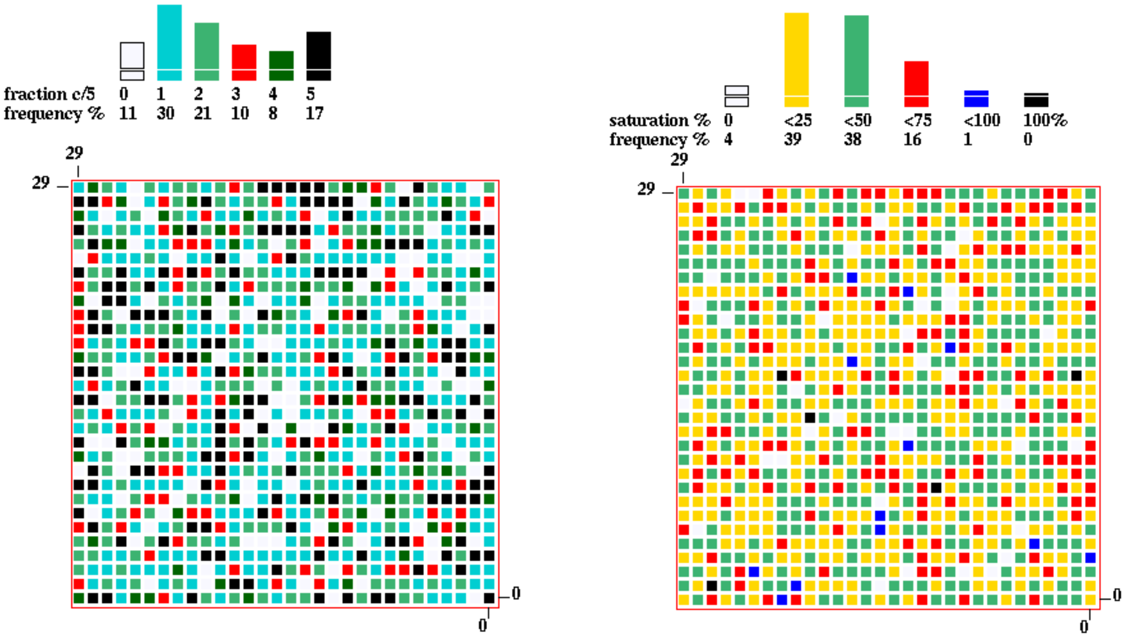
For a network with homogeneous k, according to whether "genes" or "inputs" were selected in section 15.2 above, the next prompt is as follows,

c-inputs: redo-q number-n %-(def): *(genes/inputs depends on the earlier choice)*
or
mixed k, c-inputs: redo-q number-n %-(def): *(for mixed k networks)*

Enter **return** (the default) to set a percentage of the total genes/inputs as canalizing, or enter **n** for a specific number. From this point on, the options for networks with homogeneous k, and mixed k, are somewhat different.

15.4 Canalizing - homogeneous k

The next prompt gives the fraction and percentage of the current canalizing genes and inputs, and asks for the required canalizing settings, for example,



k=5, canalizing fraction, canalizing inputs set to 45%.

k=10, canalizing saturation, canalizing inputs set to 20%.

Figure 15.1: Canalizing frequency/saturation for homogenious k networks, n=30×30. *left*: The fraction of canalizing inputs of each gene in the network, k=5, and a histogram of the frequency of each fraction. The canalizing inputs set to 45%. *right*: The saturation of canalizing inputs of each gene in the network, k=10, and a histogram of the frequency of each saturation. The canalizing inputs set to 20%. The colors in the network and histogram correspond.

```
c-inputs=0/4500=0.0% genes=0/900=0.0% (for a 2d network, k=5, 30×30)
set new % c-inputs, redo-q accept-ret:
(number/% and genes/inputs depends on the earlier choices)
```

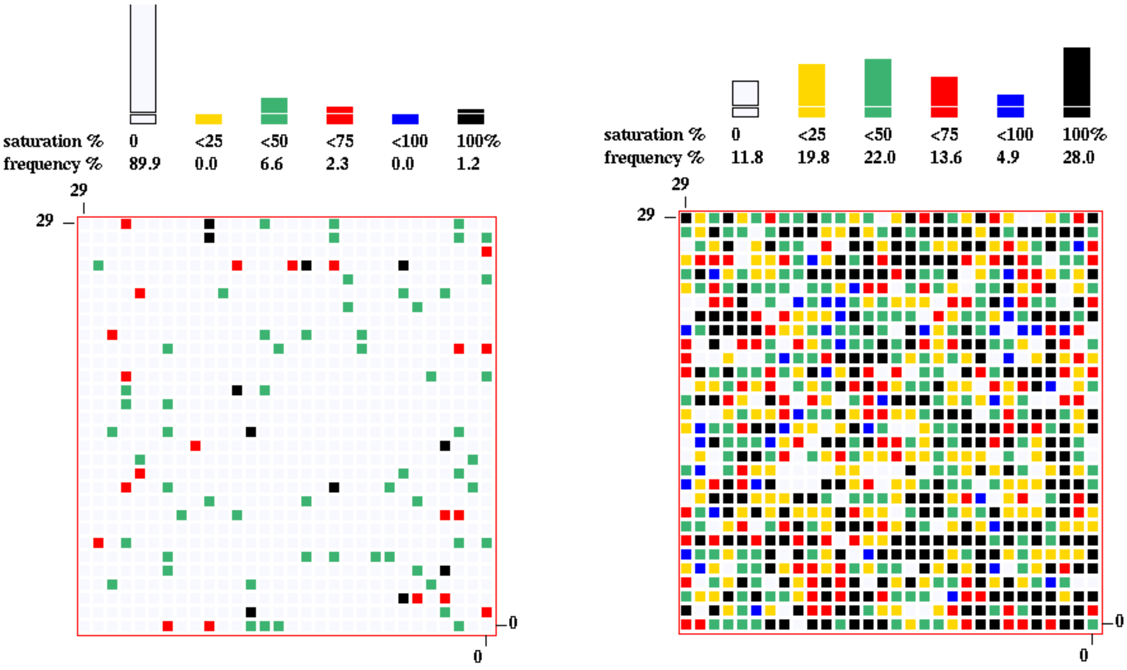
If, say, **45** (for 45%) is entered. The prompt will reapper with updated information,

```
c-inputs=2025/4500=45.0% genes=793/900=88.1%
set new % canalizing inputs, redo-q accept-ret:
```

You may enter a new canalizing setting, or enter **return** to accept, the program will continue with options to review network architecture described in chapter 17. Enter **q** to revert to the first canalizing prompt (15.2).

At the same time as the prompt appears, a window in the lower right hand corner of the screen gives a graphical display of network canalization (figure 15.1). This shows a 2d representation of the network (irrespective of its native dimension) showing the degree of canalisation of each gene.

Above this is a histogram of canalizing frequency for $k \leq 9$, or canalizing saturation for $k \geq 10$ (also for mixed k, see section 15.5).



Initial canalizing inputs, 3.0%.

Canalizing inputs set to 45%.

Figure 15.2: Canalizing saturation for two networks with the same (roughly evenl) k-mix, 3-7, as shown in section 15.5. n=30×30. *left*: The initial network showing initial, expected, canalizing. *left*: The canalizing inputs set to 45%. The colors in the network and histogram correspond.

The frequency histogram shows each degree of canalization, 0-k (see figure 15.1). The saturation histogram shows the frequency of different percentages of canalization, for 0%, 100%, and intervals of 25% in between, i.e. $0, < 25, < 50, < 75, < 100, 100$ (see figure 15.2).

15.5 Canalizing - mixed k

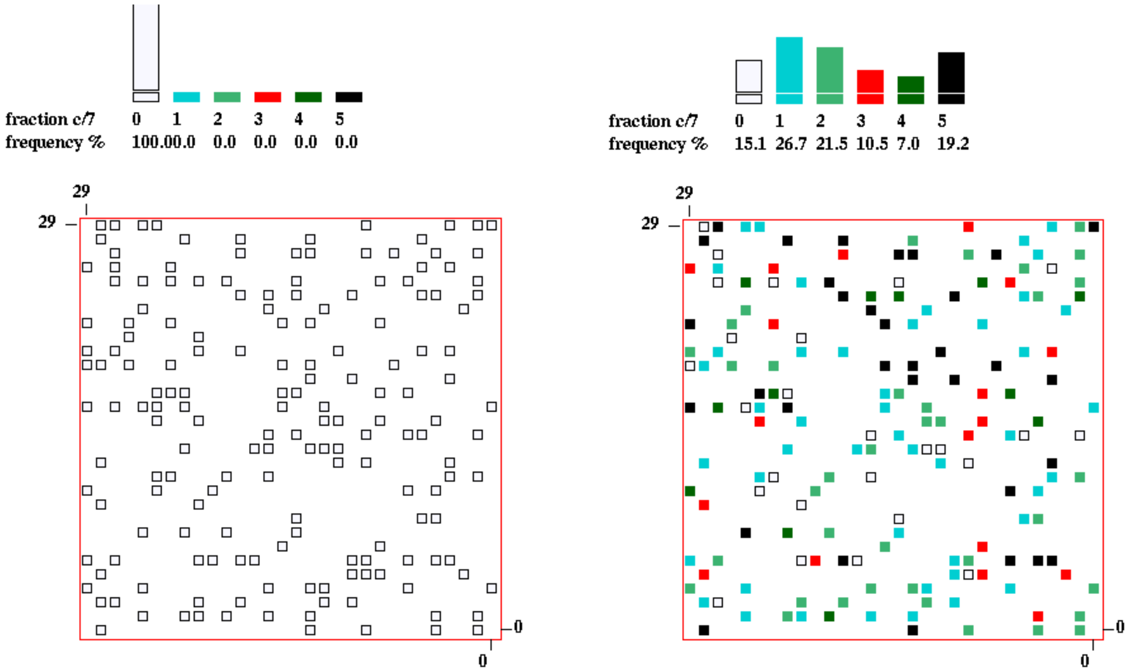
For a network with mixed k, following the prompts in sections 15.2 and 15.3, the next prompt shows the percentage of different k's (see also section 9.8.1), and offers an option to reset the canalization for just a subset of the network having a particular k, or for the whole network,

```
...
%k: 3=19.9 4=19.7 5=19.1 6=21.0 7=20.3
enter k for just one nhood, all-(def) (genes/inputs depends on the earlier choice)
```

15.5.1 Canalizing for the whole network - mixed k

If **return** is entered at the prompt in section 15.5 above, the next prompt gives the fraction and percentage of the current canalizing genes and inputs, and asks for the required canalizing settings.

For example, the prompt may appear as follows, in this case for a 30×30 network, with a k-mix of 3 to 7,



The initial k=5 genes with no canalizing inputs, as expected.

Canalizing inputs to the k=5 genes set to 45%.

Figure 15.3: Just one k in a mixed k network. Canalizing saturation for two networks with the same (roughly even) k-mix, 3-7, as shown in section 15.5. n=30×30. Only the selected k=5 genes are shown. An empty outline indicates no canalizing inputs and gaps are genes where $k \neq 5$. *left*: The initial network showing k=5 genes with no canalizing inputs, as expected. *right*: Canalizing inputs to the k=5 genes set to 45%. The colors in the network and histogram correspond.

c-inputs=134/4530=3.0% genes=91/900=10.1% *(2d network, 30 × 30)*
set new % canalizing inputs, redo-q accept-ret:
(number/% and genes/inputs depends on the earlier choices)

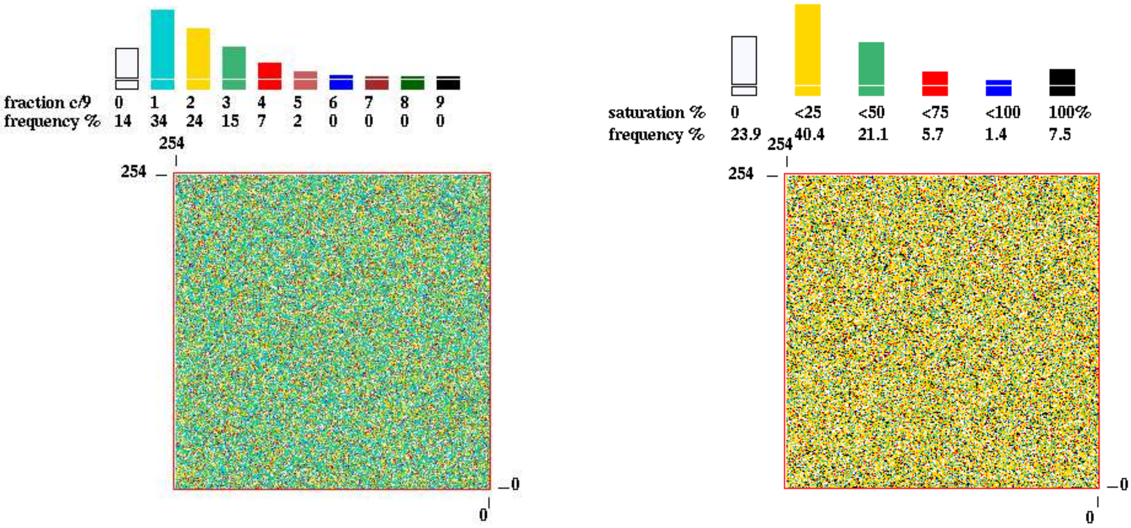
The reason for the initial canalising degree is that about 23.5% of inputs in randomly assigned k=3 rules are likely to be canalising. For k=4 rules the figure is about 1.5(section 24.13 describes how these rule-space properties are derived).

If, say, **45** (for 45%) is entered. The prompt will reapper with updated information,

c-inputs=2034/4530=45.0% genes=794/900=88.2% *(2d network, 30 × 30)*
set new % canalizing inputs, redo-q accept-ret:
(number/% and genes/inputs depends on the earlier choices)

You may enter a new canalizing setting, or enter **return** to accept, the program will continue with options to review network architecture described in chapter 17. Enter **q** to revert to the first canalizing prompt (15.2).

At the same time as the prompt appears, a window in the lower right hand corner of the screen gives a graphical display of network canalization This shows a 2d representation of the network (irrespective of its native dimension) showing the degree of canalisation of each gene.



A homogeneous k=9 network.

A mixed k network, k=3-13.

Figure 15.4: Examples of two large networks, $n = 255 \times 255$, with large k. The canalizing inputs were set to 20%. Several tries with small increments were required to reach this setting.

Above this is a histogram of canalizing saturation showing the frequency of different percentages of canalization, for 0%, 100%, and intervals of 25% in between, i.e. $0, < 25, < 50, < 75, < 100, 100$ (figure 15.2).

15.5.2 Canalizing for a particular k in a mixed k network

The canalizing setting of those genes in a mixed-k network with a just one specific k can be set in isolation from the rest of the network.

Enter the required value of k at the prompt **enter k for just one nhood, all-(def):** in section 15.5. It must be a valid value that exists in the k-mix.

For example, if k=5 is selected for the same network as shown in section 15.5.1, the following prompt will appear,

k=5 (19.1%) c-inputs=0/860=0.0% genes=0/172=0.0%)
set new % canalizing inputs, redo-q accept-ret:

This indicates that 19.1% of the network consists of k=5 genes, and that none of the inputs to those genes are canalizing, which is to be expected.

If, say, **45** (for 45%) is entered. The prompt will reapper with updated information,

k=5 (19.1%) c-inputs=387/860=45.0% genes=146/172=84.9%)
set new % canalizing inputs, redo-q accept-ret:

You may enter a new canalizing setting, or enter **return** to accept, the program will continue with options to review network architecture described in chapter 17. Enter **q** to revert to the first canalizing prompt (15.2).

At the same time as the prompt appears, a window in the lower right hand corner of the screen gives a graphical display of canalization (as in section 15.5.1) but for just the k=5 genes selected. An empty outline indicates no canalizing inputs and gaps are genes where $k \neq 5$.

Above this is a histogram of canalizing frequency for just the k=5 genes. Because we are dealing with just one k, the canalizing frequency is shown for $k \leq 9$, and the canalizing saturation for $k \geq 10$., as described in section 15.4,

15.6 Canalizing for large networks, or large k

For large networks, or large k, the algorithm for assigning canalizing may take some time. Also, the degree of canalization achieved may be less than requested. In this case more tries should be made (in sections 15.4, 15.5.1 and 15.5.2) to reach the required setting, increasing the setting by small increments.

A top center window monitors the percentage of canalizing inputs set so far. A progress bar near the top-right hand corner of the screen also monitors the assignment of rules to the network, as in section 14.9

Intentionally Blank

Chapter 16

Setting a singe rule

This chapter describes the alternative methods of setting a single rule, rcode, kcode or tcode. The methods also apply to setting rules "by hand" for a rule-mix in section 14.6.

16.1 The first single rule prompt

If one of the following selections was made at previous prompts,

return ... (the default) for a single rcode, kcode or tcode in section 14.1,
h ... for **by hand-h** for various kinds of rule-mix: (see also section 14.6)

- setting a rulemix directly (section 14.4.1),
- setting a rule-subset (section 14.4.2),
- outer-totalistic kcode or reaction-diffusion (section 14.2).

... then a main sequence prompt (sections 16.1.1 or 16.1.2 below) is displayed to accept or revise the default method of rule selection (and other options). The prompt shows a reminder of the value-range v, the neighborhood size k, the rule type (rcode, kcode or tcode), the size of the corresponding rule-table S, and the list of options which are context dependent and may vary considerably.

16.1.1 full lookup-table single rule prompt examples

For full lookup-tables (not for TFO or a k-mix) a graphic of the binary neighborhood matrix is displayed (sections 13.3.1 and 13.5.1), and the prompt is presented as follows, for example,

for rcode
Select v2k5 rcode (S=32): empty-e fill-f prtx-x val-v rnd-r bits-b
hex-h dec-d maj-m/u Alt-A life-L chain-c RD-R iso-i rep-p load-l (def-r):

for kcode
Select v3k4 kcode (S=15): empty-e fill-f prtx-x val-v rnd-r bits-b
hex-h dec-d maj-m/u Alt-A rcode-R rep-p load-l (def-r):

for tcode
Select v4k5 tcode(S=16): empty-e fill-f prtx-x val-v rnd-r bits-b
hex-h dec-d maj-m rule-R rep-p load-l (def-r):

16.1.2 TFO-mode single rule prompt examples

for kcode
Select v3k4 kcode (S=15): empty-e fill-f k+-k prtx/swap-x val-v rnd-r bits-b hex-h dec-d maj-m/u Alt-A rep-p load-l (def-r):

for tcode
Select v4k5 tcode (S=16): empty-e fill-f prtx-x val-v rnd-r bits-b hex-h dec-d maj-m rep-p load-l (def-r):

16.2 Methods for setting a rule

The meaning of the prompts for setting a rule[1] in section 16.1 are summarised below. More details are given in the rest of this chapter. These options also apply to a rule in a rule-mix set "by hand" in section 14.6, where the selected method stays as the default unless changed in section 14.6.4. Some options do not apply to particular cases as noted. **rnd-r** is the initial default in the main sequence of prompts, **bits-b** is the default if a rule has already been selected, or if revising a rule at later stages (sections 30.5.1 and 32.14.2).

options . . . *what they mean*

empty-e . . . to reset all rule-table entries to 0.

fill-f . . . to fill the lookup table with any valid value with the top-right prompt,

fill with value 0-4 (def 4): *(if v=4)*

This allows a clean slate for setting bits or values in section 16.4.

k+-k . . . *(only for kcode in TFO-mode)* to create and save a kcode with an increased neigborhood $k+$ by inserting a random string of the correct length within the current kcode (section 16.17).

prtx-x . . . show the rule in the terminal (section 16.18).

prtx/swap-x . . . *(for kcode only)* as above, but also allows swapping output values to make an equivalent kcode (section 16.18.5).

val-v . . . to save or load the rule(`*.tbl`), as an ascii string. The following top-right prompt is presented (for example),

ascii table (v=4 k=3 S=55) as ascii string, load/save-l/s:

note . . . *After the prompts above the program reverts to the first single prompt in section 16.1. The prompts below set a rule and the program comtinues.*

rnd-r . . . to set the rule at random, for a given density bias or λ parameter (section 16.3).

bits-b . . . to set the rule as bits or values – "draw" the rule-table on a 1d or 2d graphic array, using the mouse or keyboard (section 16.4).

[1] Some of these methods are similar to setting a seed in section 21.2.

hex-h . . . to set the rule in hexadecimal, in a mini "spread sheet" (section 16.5).

dec-d . . . *(if applicable)* to set the rule in decimal (section 16.6).

maj-m/u . . . Enter **m** to set a "majority" (voting) rule (section 16.7).
Enter **u** *(not for tcode)* to assign a majority rule but with the outputs from uniform neighborhoods flipped or shifted (section 16.8).

Alt-A . . . *(not for tcode)* to set an "Altenberg" rule, where the output of each neighborhood is set probabilistically according to the frequency of the values in that neighborhood (section 16.9).

Life-L . . . *(for rcode only, and if $k \geq 5$)* to set to the "game-of-Life" or quasi "Life" (section 16.10).

chain-c . . . *(for rcode only)* to set a maximally chaotic rule, where $Z_{left} = 1$ or $Z_{right} = 1$, but not both. For binary, In addition both Z_{left} and Z_{right} are greater than 0.5 (section 16.11).

RD-R . . . *(for rcode only)* to select a reaction-diffusion rule see section 13.8.2.

rcode-R . . . *(for tcode or kcode and full lookup-table – not TFO-mode)* to show a tcode or kcode as a full lookup table, which can then be modified.

iso-i . . . *(for rcode only)* to set the rule at random but as an isometric rule, where rotated and reflected nhoods (in 1d, 2d and 3d) have the same output.

rep-p . . . to repeat the last rule. (16.13).

load-l . . . to load a rule. from a `*.rul`,`*.vco`, or `*.tco`, file.

Once a single rule have been set, except after **rnd-r** or in a rule-mix "by hand" (section 14.6), the **bits-b** option is activated automatically. Once accepted the rule is reconfirmed as a bit/value pattern (in decimal if applicable) and in hex, and simultaniously displayed in the rule window (section 16.19). At any stage enter **q** to backtrack and revise.

16.3 Setting the rule at random

The density of non-zero values – the λ parameter – in the random rule-table (expressed as a percentage) is displayed in a lower top-right density window. The default is an equal probability of each value, so 50% 1s for binary. This example is for v=8, a 2d seed $[i, j]$=[20,20], and a block [14,14],
If **r** is selected in section 16.1, a random rule is generated and displayed as a rule-table bit/value string, with the ability to tune the density bias of non-zero values – the λ parameter. Data on the random rule is simultaneously displayed in top-right windows (section 14.1.1). Initially there is an equal probability of each value, unless this was reset.

Various changes and transformations[2] can be made – repeatedly until **return** or **q** is entered – to accept the rule. The following subsequent reminder is shown,

tog gaps-g, rotate-l/r another-n density-s Z-u/d comp-m back-q accept-ret *(**Z-**rcode only)*

[2] These option are similar to setting a random initial state or seed in section 21.3.

options . . . *what they mean*

tog gaps-g . . . to toggle gaps between sucessive blocks of 8 bits/values. This is the default for binary rules (i.e. between chars) as in figure 16.10.

rotate-l/r . . . enter **l** or **r** to rotate rule-table left or right by one output, with the edge outputs wrapping around.

another-n . . . for another random rule with the same density bias – λ parameter.

density-s . . . to change the density bias – λ parameter (section 16.3.1).

Z-u/d . . . *(for rcode only)* enter **u** or **d** (or keep the key pressed) to progressively force the Z-parameter up or down, by selectively mutating the rcode. This is done by flipping bits/values at random positions, and only retaining the flips that produce the desired change in Z. The algorithm for forcing Z down can get stuck – if so lower the density bias with **density-s**.

comp-m . . . to toggle/transform the rule into its compliment. For binary this changes 1s to 0s and vice versa. For $v > 2$ each value a is changed to its compliment $a_c = (v-1) - a$ (see section 16.20).

back-q . . . to backtrack to the first single rule prompt (section 16.1/ – the latest rule is remembered, and can be amended with **bits-b** or **hex-h**.

accept-ret . . . to accept the rule. Once accepted, the rule is also displayed in decimal (if applicable), and in hex.

Some of these options are described in greated detail below.

16.3.1 Random rule density-bias (λ parameter)

Enter **s** in section 16.3 to change the density-bias – the fraction of non-zero values in the rcode-table (the λ-parameter) but expressed as a percentage. The initial default is an equal probability of each value, but this may have been reset in section 14.1.1. The lower top-right window shows information about the density, in this example for rcode $v5k6$,

(the density window)

```
density (4-1s)exact=12500/15625=80.000%, bias-80.000%
```

where "density (4-1s)" indicates the non-zero values, "prob" the current method of setting the density as opposed to "exact", 15625 is the size of the rcode-table, then the actual density, and the bias requested.

There are two alternative methods to change the density-bias, **exact** or **prob** (**exact** is the initial default). The following two stage top-right prompt is presented, for example,

(for v=2, 22% entered, with **prob** *as the default method)*

bias-density: enter % (def 50.000% prob):22 exact-e: *(enter* **e** *to change to* **exact***)*

(for v=8, 33% entered, with **exact** *as the default method)*

bias-density: enter % (def 87.500% exact):33 prob-p: *(enter* **p** *to change to* **prob***)*

Enter the new density-bias as a percentage – the result updates the density window. For binary this is the probability of setting 1s. For $v > 2$ it is the probability of setting non-zero values – set without bias. The new density-bias becomes the new default. If the **prob** method is active the density-bias requested is applied as a probability, whereas the **exact** method will apply the requested bias as closely as possible. To change the default method between **prob** to **exact**, enter **e** or **p** as indicated. The density-bias and method will be maintained for further random rules generated with **another-n** in section 16.3 updating the density window.

16.3.2 Random rule parameters

not applicable in TFO mode

For each alternative rule created in section 16.3, current rule parameters (as finally appear in the rule window – section 16.19) are shown in an upper top-right rule parameters window. This example is for a $v2k5$ rcode,

(the rule parameters window)

*C=1/5=*3*** zl=0.1875 zr=0.1875*
ld=0.09375 ld-r=0.1875 P=0.90625 Z=0.1875

abbreviations ...	*what they mean*
C=1/5 ...	the number of canalizing inputs[9], in this case 1 out a possible 5 (k=5).
*3*** ...	if there are canalizing inputs, this shows which are canalizing, in this case one input at index 3 (from a possible 0 to 4, see chapter 10).
zl ...	Z_{left}.
zr ...	Z_{right}.
ld ...	the λ parameter[14].
ld-r ...	the λ ratio[19].
P ...	the P parameter[9].
Z ...	the Z parameter[19, 27].

16.4 Setting the rule as bits or values

If **b** is selected in section 16.1, a bit or value pattern representing the current rule-table (initially just 0s) is displayed. This consists of a single row or multiple rows for longer rule-tables. For v=2 the default has gaps between sucessive blocks of 8 bits (i.e. between chars) but this can be toggled. The default scale of the pattern depends on the size of the rule-table, S, but can be changed. The maximum rule-table index, S-1, is in the top left hand corner, the zero index in the lower right

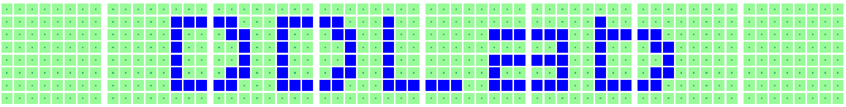


Figure 16.1: Setting bits starting with all 0s, with default gaps and white divisions. $v2k9$, S=512.

hand corner of the pattern. The current position on the pattern is indicated by a small flashing cursor (initially top left as in figures 16.1-16.2), and also in a top-right inset window.

The colors correspond to values (chapter 7) but 0s are initially colored light green. If a rule is already current, it can be reset to all-0s with **empty-e** or to any uniform value with **fill-f** in sections 16.1.1 or 16.1.2 to provide a clean slate for setting bits/values. Alternatively, use the bit/value setting method for fine adjustments to rule-tables set by other methods. Figure. This also illustrates alternative presentations which can be changed on-the-fly.

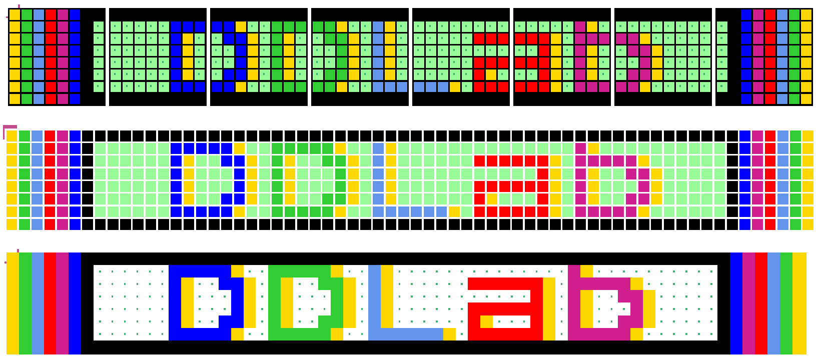


Figure 16.2: Setting values – alternative presentations, $v8k3$, S=512.
top. gaps, black divisions, 0s light green with dots.
centre. no gaps, white divisions, 0s light green, no dots.
bottom. no gaps, no divisions, 0s white with dots.

16.4.1 Rules: bits/values reminder window

Bits or values in the rule-table are set with the mouse and keyboard. Similar methods apply for the seed in secton 21.4.1. During the procedure, a top-right window, and inset display, gives reminders of various options and current settings (summarised below), this example for a $v5k6$ kcode where the rule-table size is 210,

the inset while drawing with the mouse → left button: draw 1s width=1

the inset with current settings ↘

keys: set/select/draw 4-0, vert-v count=0 kcode index=209, scale=5 rot=1
mouse: move-click draw-drag draw-w move-arrows, PScript-P file-F
tog: gaps/0color/dots/divs/divcolor-g/∧/./i/! exp/contrect-e/c
rotate-l/r/+/- flip-h comp-m back/accept-q/ret

options . . . *what they mean*

keys: select/draw 4-0 . . . enter a valid number to select the value/color, and keep the key pressed to draw a horizontal line.

vert-v. . . enter **v** for a vertical line, and keep **v** pressed to draw a vertical line downwards in the selected value/color.

mouse: move-click . . . left click on the rule pattern to reposition the small cursor and activate drawing – sometimes the right button also needs to be clicked (or right-left a few times) to activate.

draw-drag . . . draw the selected value/color by dragging the cursor with the left mouse button pressed – release the button to stop. The right mouse button draws the complimentary value/color (section 16.20) in the same way.

width-w . . . enter **w** to change the width (initially 1) of the line to be drawn, where the max width is the number of rows. The following top-right prompt is presented (for example),

reset line width (now 1) max 4:

The width is shown in the drawing inset, and stays for the current drawing session until revised.

move-arrows . . . all four arrow keys move the cursor around the seed rule-table (up/down arrows for multilpe rows).

PScript-P . . . to save the rule bit/value pattern as a vector PostScript image (section 16.4.3).

file-F . . . to save the rule pattern as a 1d seed file (section 16.4.3).

tog: gaps-g . . . enter **g** to toggle gaps between sucessive blocks of 8 bits/values (figure 16.3).

tog: 0color-∧ . . . to toggle the zero value color between light green and white.

tog: dot-. . . . enter a dot to toggle a dot at the centre of each zero cell (figure 16.2).

tog: divs/divcolor-i/! . . . enter **i** to toggle division lines between the cells. Enter **!** to toggle the division line color between white and black (figure 16.2).

exp/contrect-e/c . . . enter **e** to expand, **c** to contract the scale by one pixel.

rotate-l/r/+/- . . . enter **l** or **r** to rotate the rule-table left or right by the rotation interval (initialy 1) with the edge outputs wrapping around. Enter **+** or **-** to increace or decreace the rotation interval – shown in the current settings inset.

flip-f . . . to flip (reflect) the rule table.

comp-m . . . to toggle the rule into its compliment. For binary this swaps 1s and 0s. For $v > 2$ the values are shifted; each output a changes to a_m by subtraction from the maximum value $v-1$, so $a_m = (v-1)-a$, and the maximum value changes to zero (section 16.20.

back/accept-q/ret . . . enter **ret** to accept the rule, **q** to backtrack.

count=0 kcode index=209, scale=5 rot=1 ← *the inset with current settings, for example*

current settings . . . *what they mean*

count= . . . the current cursor position, starting from 0 in the top left.

kcode index= . . . (or rcode index, or tcode index) the current cursor position according to the rule index, where where the bottom right is 0, and the top left is S-1, where S is the size of the the rule-table.

scale= . . . the current scale in pixels.

rot= . . . the current interval by which the rule-table can be rotated.

16.4.2 Rules: setting bits/values with the keyboard and mouse

The active position (to be updated) in the rule-table is highlighted by a small flashing cursor initially in the top left. Its position is displayed in the top-right inset window. The flashing cursor is repositioned with the mouse by clicking either the left or right button on the new position[3], or moved with the left/right/up/down arrow keys.

Setting or drawing values on the rule-table pattern is done with the keyboard or mouse (figures 16.1, 16.3). To set (and activate) a value at the cursor position, press a valid number key (without return) – the cursor will then advance to the right by one unit. To draw a horizontal line towards the right keep the key pressed; eventually the line will continue to the next row or jump back to the top left. To draw a vertical line downwards with the active value, press **v**.

To draw the active value with the mouse, drag the cursor anywhere over the rule-table pattern with the left button pressed – release to stop. To draw the compliment of the active value (see section 16.20) drag with the right button pressed. While a mouse button is pressed, the inset in the reminder window changes to show which button, the current active value, and the current width of the line, for example,

left button: draw 3s width=2 *or* right button: draw 4s width=3

Initially the left button draws the value v-1 and the right draws 0, so for binary 1 and 0. To activate drawing with the mouse it is sometimes necessary to click the left and right button alternately. To accept the rule-table enter **return**. Once accepted, the rule is also displayed in decimal (if applicable), in hex, and in the rule window (section 16.19).

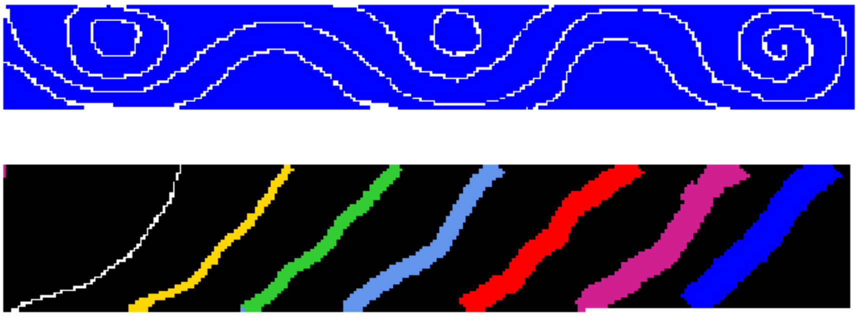


Figure 16.3: Drawing bits or values on the rcode pattern by dragging the mouse cursor.
top: drawing value 0 with line width=1 on an all-1s bit pattern, $v2k13$ rcode, S=8192.
bottom: drawing values 0,1,2,3,4,5,6, with line width 1,2,3,4,5,6,7, on an all-7s value pattern, $v8k9$ kcode, S=11440.

16.4.3 Rules: save as 1d seed or PostScript

The rule-table can be saved as a vector PostScript image, or as a 1d seed file, by selecting *PScript*--**P** or *file*-**F** in section 16.4.1 – both options follow the same methods as the 1d seed options in section 21.4.7. The following prompts appear in a top-right window,

for PostScript, PScript-**P**
PostScript RCODE: save all-a, save patch-p: *(or kCODE, TCODE)*

for a 1d seed file, file-**F**
KCODE as 1d SEED: save all-a, save patch-p: *(or RCODE, TCODE)*

[3] There is a slight difference between Linux-like systems and DOS. In Linux the mouse cursor changes direction within the rule-table pattern, pointing north-west instead of north-east, and within the pattern, left or right mouse clicks reposition the flashing cursor. In DOS the mouse cursor is confined within the rule-table pattern and clicking the left or right buttons sets values as well as repositioning the flashing cursor.

Enter **a** for the whole rule-table. If **p** is entered, successive prompts are presented in a top-right window to define a patch – the default is set by the last two mouse clicks on the rule-table. The mouse click index shows up in the inset panel in section 16.4.1. This patch prompt example is for v=8 RCODE,

1d: max i=511, revise coords-ret, accept patch 71-278-p: *(patch = last 2 mouse clicks)*
start i: **end i:** *(... if* **ret** *was entered above)*

Enter **p** to accept the default patch. Enter **return** to set the start/end values manually – the defaults are the mouse click coordinates.

16.4.4 Rules: PostScript prompt

When creating a PostScript image of a rule, there are a variety of presentation possibilities as illustrated in this chapter, and in section 21.4.9 for a seed where similar methods apply.

Enter **a** (or **p** for part of the seed) at the prompt in section 16.4.3 to save the rule-table image as a vector PostScript (`*.ps`) file (default filename `my_sPS.ps`). Various top-right options will be presented for the exact appearance of the image. This example is for v=8 RCODE,

create a PostScript image for 1d, 64x8
symbols-s greyscale-g color-c exit-q (def-c):

These options are summarised below (then subsequent options continue),

options ...	*what they mean*
symbols-s ...	to show values as symbols (as in figures 21.7 and 21.8 for a seed).
greyscale-g ...	to show values in greyscale (as in figures 21.7 for a seed).
color-c ...	*(the default)* to show values in color.

Once these options have been selected, the prompt to amend other setting is presented,

cellscale=5.00 dots(on)=0.70 divs(off), amend settings-a: *(for example)*

Enter **return** to accept the defaults, or enter **a** for the following prompts presented in sequence,

change: cellscale: togdots-x: dotscale: divs(0,1,2):

Enter changes required or **return** to accept defaults, which follow the current bits/values presentation. The options are summarised below,

options ...	*what they mean*
cellscale ...	enter a new cell width in pixels.
togdots-x ...	enter **x:** to toggle zero dots on/off.
dotscale ...	*(if dots are on)* enter a new width for zero dots in pixels, which can be a decimal number.
divs(0,1,2) ...	the initial default division (color) depends on the bits/values presentation, and is shown in the prompt: (**none**, **1**, **2**) where **none** means that there is no division and adjoining cells touch. To change, enter **0** for none, **1** for black, and **2** for white. The new designation becomes the default.

Cells with value zero (0color) are initially colored light green in the bits/values presentation (section 16.4), which can be togged to white with *0color-∧* in section 16.4.1 – the PostScript file will follow the current 0color.

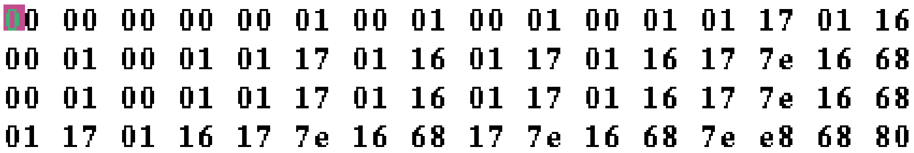
Once these choices are complete, the `*.ps` file is saved from the filing prompt (section 35.2). Section 36.1 explains how to view, edit, and crop the PostScript image.

16.5 Setting the rule in hex

If **h** is selected in section 16, the rule is defined in hexadecimal (hex), which is a shorter way of denoting a rule than the rule-table itself. The method is the same as setting a seed in hex in section 21.5. The hex expression of the current rule (initially just 0s) is displayed. Each hex character (0 – 9 and a – f) shows the value of 4 bits, and the characters are displayed in one byte pairs. During the hex setting procedure, a top-right reminder window displays the following,

enter hex, arrows-move hex count=45 ← *inset shows the current hex position, from the top left*
rotate-l/r, accept-ret

The hex character to be updated is highlighted by a flashing cursor, initially in the top left. Its position is displayed in the top-right inset window. The flashing cursor is moved with the arrow keys, left/right and up/down for more than one hex line. To overwrite, enter a hex character from the keyboard, without **return**. This automatically moves the cursor one position to the right. Hex characters entered which exceed the current value-range will be automatically corrected downwards to the maximum value after the hex string has been accepted. Enter **l** or **r** to rotate the rule-table by one bit or value as in sections 16.3 and 16.4.1, which will be immediately reflected in the hex presentation. To accept the rule as expressed in hex, enter **return**, whereupon the rule will be presented as bits/values (section 16.4), where it can be reconfirmed or amended.



```
00 00 00 00 00 01 00 01 00 01 00 01 01 17 01 16
00 01 00 01 01 17 01 16 01 17 01 16 17 7e 16 68
00 01 00 01 01 17 01 16 01 17 01 16 17 7e 16 68
01 17 01 16 17 7e 16 68 17 7e 16 68 7e e8 68 80
```

Figure 16.4: Setting rcode in hex, showing the "game-of-Life", $v2k9$.

16.6 Setting the rule in decimal

applicable only for a limited range of v and rule-table size S

The decimal option is useful for binary rules with small neighbourhoods, such as the $v2k3$ "elementary rules" with their well known decimal rule numbers[17, 19] or for v=2 tolalistic rules.

However the decimal option remains valid so long as the rule table (rcode, kcode or vcode) is within the limits listed in table 16.1 which lists the the maximum length of the bit/value string S and and the corresponding maximum decimal rule number, for each value-range v. The same limits apply when setting a seed in decimal in section 21.6.

v	max-S	max decimal
2	32	4294967294
3	20	3486784400
4	16	4294967294
5	13	1220703124
6	12	2176782335
7	11	1977326742
8	10	1073741823

Table 16.1: Rules in decimal – rule-table size limilations – the maximum bit/value string S for each value-range v, and the corresponding maximum decimal number.

If **d** is an available option and is selected in section 16.1, the rule can be specified by its decimal equivalent. The following prompt is displayed,

k=3 rule, enter 0-255 (def-rnd-dec): *(for example)*

Enter a decimal number, or enter **return** for a random number within the permitted range, which will be displayed. If the number entered is outside the permitted range, the program presents the message,

k=3 rule, enter 0-255 (def-rnd-dec):333 - too big! back-ret: *(for example)*

Enter **return** to revert to first single rule prompt (section 16.1). Once successfully selected, the decimal rule is presented again as bits/values (section 16.4), where it can be reconfirmed or amended.

16.7 Setting a majority rule

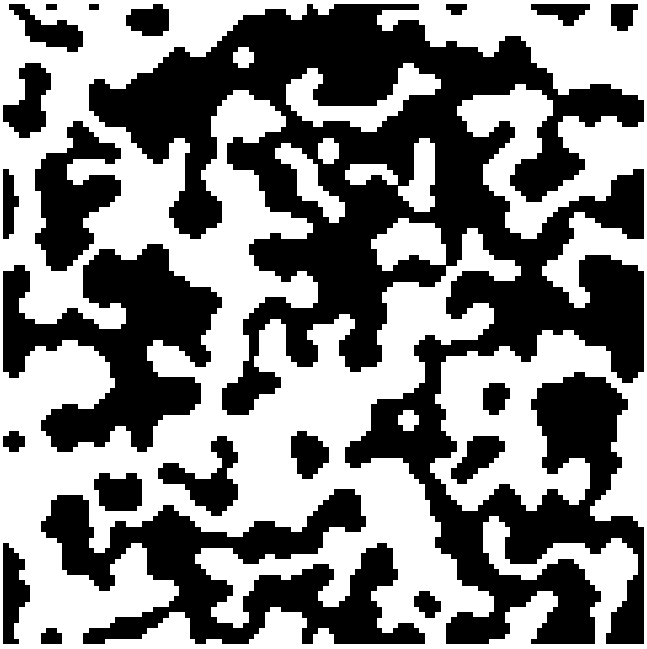
If **m** is selected, in section 16.1, the "majority" (voting) rule will be set – an example was shown in see figure 2.2. The algorithm differs between rcode, kcode and tcode, and also between $v = 2$ and $v > 2$. For rcode or kcode the majority value in the neighborhood becomes the output.

In case of a tie,

- for rcode and $v = 2$ the central cell wins, as before in the old binary version of DDlab.
- otherwise for kcode or $v > 2$ one of the majority values is picked at random (see figure 16.6).

For tcode, the tcode-table is divided into v approximately equal sectors, and the values are allocated to the sectors in decending order from the left (see figure 16.7). Once selected, the majority rule will be presented again as bits/values (section 16.4), where it can be reconfirmed or amended.

Figures 16.5 – 16.7, and 4.14. show examples of the majority rule bit/value pattern, and evolved snapshots of 2d spacetime patterns, for rcode, kcode and tcode.



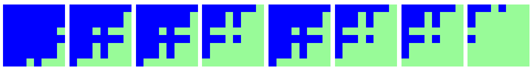
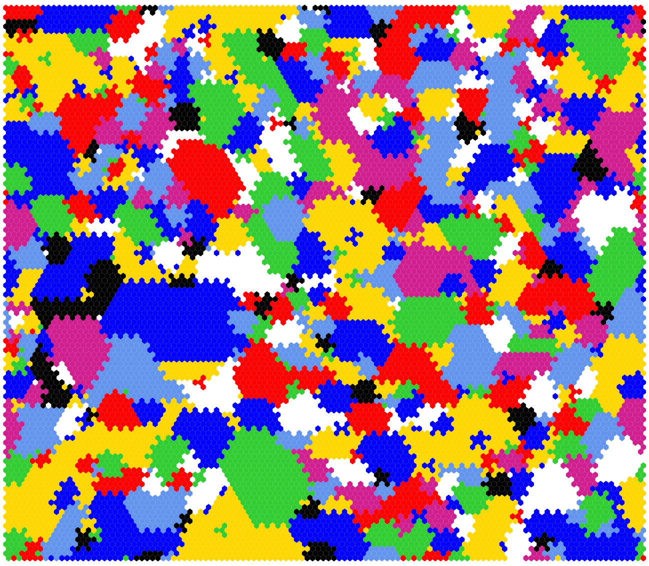


Figure 16.5:
above: Majority rcode $v2k9$, shown as a bit pattern of S=512 bits.

left: Space-time snapshot from a random initial state on a 120×120 square lattice, where the dynamics has stabilised on an attractor.



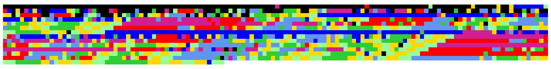
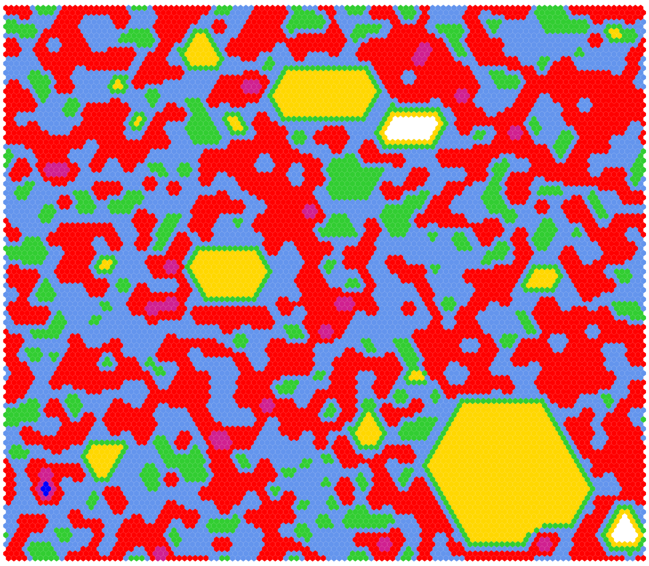


Figure 16.6:
above: Majority kcode $v8k6$ shown as a value pattern of $S = 1716$ values.

left: Space-time snapshot from a random initial state on a 120×120 hexagonal lattice, where the dynamics has stabilised on an attractor.



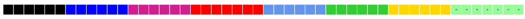


Figure 16.7:
above: Majority tcode $v8k7$ shown as a value pattern of S = 50 values.

left: Space-time snapshot from a random initial state on a 120×120 hexagonal lattice, where the dynamics evolves into competing patches.

16.8 Majority with shifted uniform outputs

not for tcode – see also section 14.8 for a rule-mix

If **u** is selected in section 16.1 a "majority" (voting) rule is set as in section 16.7 above, but the uniform (unanimous) neighborhoods have their outputs shifted by -1, except for 0 which becomes v-1. For v=2 this is the same as flipping the "end bits", as before in binary DDLab, so that unanimity gives the opposite vote, otherwise the majority wins – the tcode is 011001.

For random wiring, this can result in some interesting bi-stable, tri-stable, v-stable, dynamics, for example in figure 16.8 for v=2, and for a rule-mix of kcode majority rules, v=8, in figure 14.4.

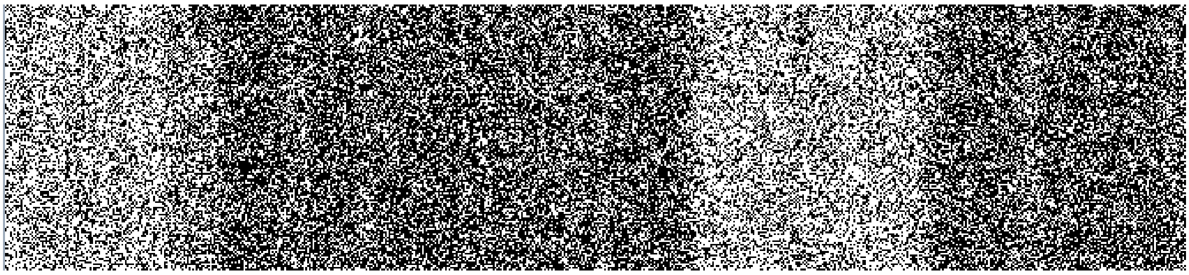
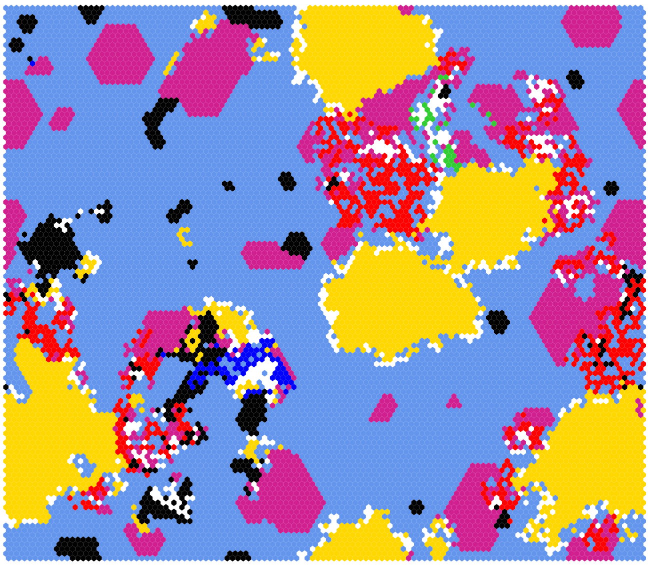


Figure 16.8: Flipped (shifted) majority rcode with random wiring, $v2k5$, where uniform neighborhoods (all-0s and all-1s) have their outputs flipped to the opposite color. A 1d (n=150) space-time pattern (rotated by 90° so the time flows from left to right) shows bi-stable pattern density, where the duration of the two density regimes is unpredictable.



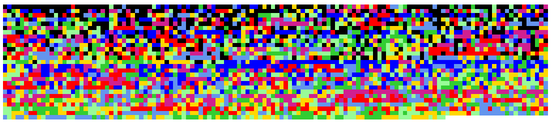


Figure 16.9:
above: Altenburg kcode $v8k7$, shown as a value pattern of $S = 3432$ values.

below: Evolved space-time snapshot from a random initial state on a 120×120 hexagonal lattice.

16.9 Setting Altenberg rules

not applicable to tcode

Enter **A** to set an "Altenberg" rule (suggested by Lee Altenberg) where the output of each neighborhood is set probabilistically according to the frequency of the values in that neighborhood. This is a sort of probabilistic majority rule, and results in mobile ordered zones in the dynamics. An example kcode and 2d snapshot is shown in figure 16.9, and a 1d space-time pattern in figure 4.11.

16.10 The game-of-Life and other Life-like rules – rcode

for rcode (full lookup-tables) and $k \geq 5$ – (for Life in TFO-mode see section 14.2.2)

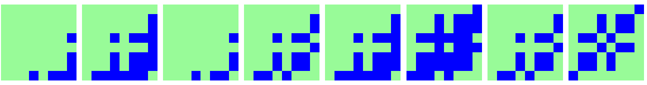


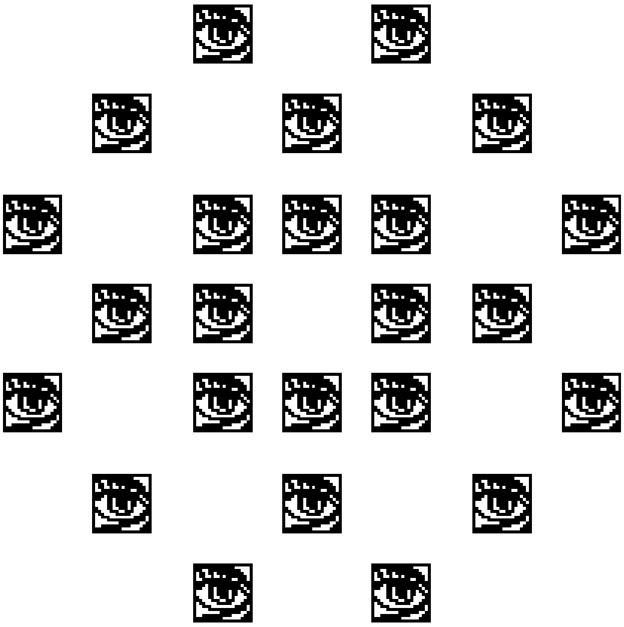
Figure 16.10: Conway's game-of-Life (rule 23/3) shown as a 512 bit rcode.

For $v2k9$ on a square lattice, the "Moore" neighborhood ▦ John Conway's game-of-Life[5] can be set – see figures 4.12 and 4.13 for examples of space-time patterns. Alternatively any other Life-like rule from the "Life family" can also be set, with different values of v and k.

The Life rules can also be set as outer-totalistic rules in TFO-made (section 14.2.2 and figure 14.1), which allows a greater range of neighborhoods sizes, up to k=25 in 1d and 2d, but this section describes the method employing the full lookup table, rcode.

Classical Life is specified as (23/3) in the Mirek's Cellebration notation[4]. A cell is either alive (1) or dead/empty (0). The first part of (23/3) defines the survival of a cell, requiring 2 or 3 live

[4] http://en.wikipedia.org/wiki/Life-like_cellular_automaton



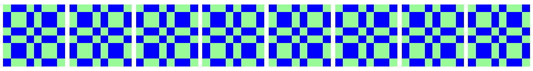


Figure 16.11:
above: A rule in the Life family, Fredkin's replicator, specified by 1357/1357 in the conventional notation. The $v2k9$ rcode, shown as a bit pattern of $S = 512$ bits.

left: Starting with one central "eye" as the initial state, replicated patterns repeatedly pop out from disorder, this at time-step 193. 222×222 on a square lattice.

neighbors, the second part of (23/3) defines birth, requiring 3 live neighbors, otherwise the cell is dead (by overcrowding or exposure). This notation when entered in DDLab is automatically translated into the full binary lookup-table, rcode (figure 16.10). Any other Life-like rule can be specified in this notation, for example 1357/1357 for Fredkin's replicator in figure 16.11.

16.10.1 Setting Life-like rules – rcode

When **L** is entered in section 16.1 the following top-right prompt is presented,

> **Life k=9 (def: survival 2,3, birth 3,) accept-ret amend-a:** *for k=9*

Enter **return** to accept the default. If **a** is entered to amend, the following further prompts are presented,

> **accept-ret, or enter number+ret, max entries=9, max value=8:**
> **enter survival (def=2.3,):** *followed by . . .*
> **enter birth (def=3,):**

Enter the new values for survival, followed by the new values for birth. After each number enter **return** for the next number. There may be up to k entries – their order, or repeats, are not significant. **return** without a number concludes the entries. **q** reverts to the first Life prompt, but with the defaults possibly altered. The Life option is available for $v \geq 2$, and any $k \geq 5$ as well as the k=9 neighborhood, for 1d and 3d as well as 2d, and for a rule-mix by hand. For $v > 2$, for a given Life setting, the algorithm in DDLab generates an equivalent rcode giving the same dynamics as binary Life, but including v colors. For example, v=3 and (23/3) gives the same game-of-Life dynamics but with two colors plus the background, as in figure 16.12. To make changes to particular bits in Life-like rule tables, amend bits or values as in section 16.4.

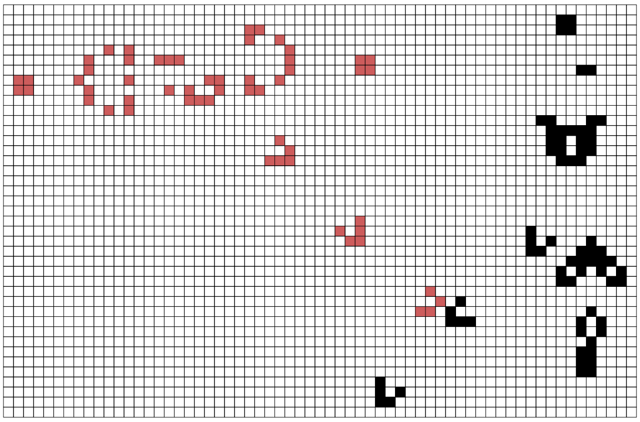


Figure 16.12:
The game-of-Life (23/3) appled to a v=3 CA. The algorithm in DDLab generates an equivalent rcode giving the same dynamics as the classical binary Life, but including 2 colors + background. In this example two glider guns have been constructed, one shooting black gliders SE, the other shooting red gliders SW. Composite red/black gliders also exist.

16.11 Setting a chain-rule

rcode (full lookup table) only

Enter **c** in section 16.1 to set a chain-rule.

Chain-rules are maximally chaotic rules, where $Z_{left} = 1$ or $Z_{right} = 1$, but not both. The global dynamics of chain-rules exhibit extremely low convergence in subtrees. For larger systems, states usually have just one pre-image, so subtrees form a "chain". Garden-of-Eden density becomes order zero with increasing system size. Chain-rules comprise approximately the square root of rule-space (figure 24.10), and the CA reverse algorithm is especially efficient for generating their subtrees. These characteristics make chain-rules suitable for encryption[35], for example, by running "information" backwards to encrypt, forwards to decrypt as in figure 16.13. The methods apply equally for $v > 2$, but for binary, the lesser Z is set greater than 0.5 to minimize structure that is close (in time-steps) to the "information".

16.12 Setting reaction-diffusion – rcode

for reaction-diffusion by by outer-kcode see section 14.2.1

Enter **R** in secton 16.1 above to initiate a reaction-diffusion rule with the full lookup table – rcode. (see sections 13.8, 13.8.2 and figure 13.4). Then set the threshold interval as described in section 13.8.3. An alternative method for reaction-diffusion is from outer-kcode, which allows greater $[v, k]$, see sections 14.2.1.

16.13 Repeating the last rule

If **p** is selected in section 16 (for a single rule – not part of a rulemix), the last rule that was set for the given $[v, k]$ is repeated by automatically loading the "last rule" file with the same v, k and rule type parameters (see section 16.15). The file would have been automatically saved when the

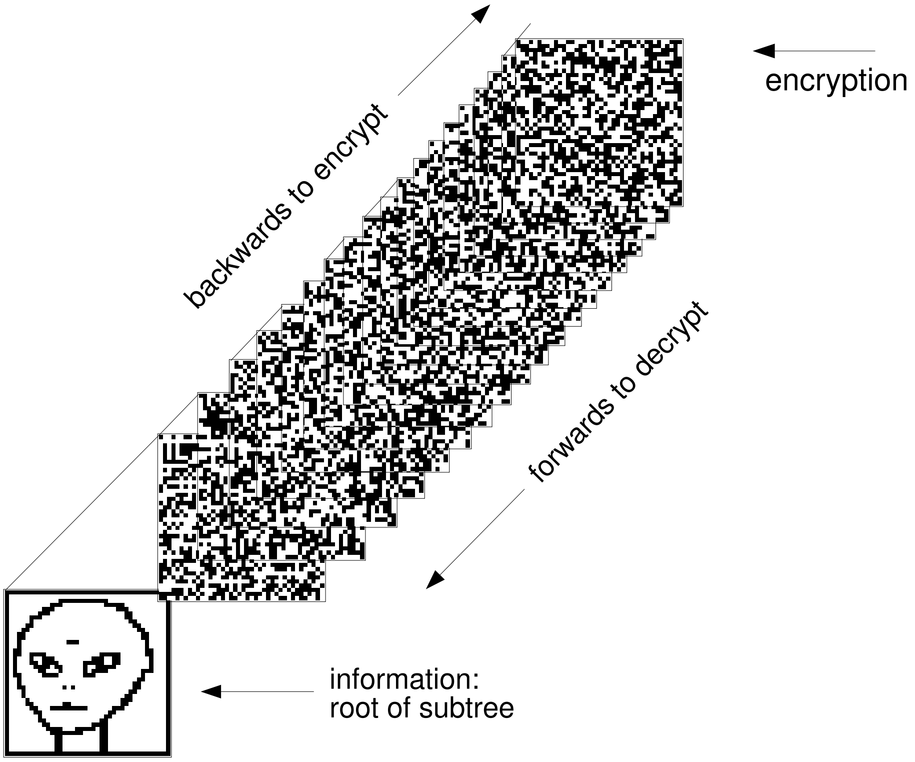


Figure 16.13: A subtree applied to encrypt information at its root state (the seed), set to stop after 19 backward time-steps, where the state reached is the encryption. The root state is a 1d bit-pattern, here displayed in 2d (n=1600, 40×40). The "alien" seed was drawn with the drawing function in DDLab. The seed could also be an ASCII file, or any other form of information. A k=7 chain-rule (rcode) was set at random, and the subtree was generated with the CA reverse algorithm. Note that the subtree has not branched – branching is highly unlikely because of the large system size. To decrypt, apply the same rule to run forwards by the same number of time-steps.

last rcode, kcode or tcode was selected. Once selected, the repeated rule is presented again as bits/values (section 16.4), where it can be reconfirmed or amended.

In a rule-mix, where a sequence of rules is entered by hand (see section 14.6), the "last rule" file is not relevant – selecting **p** repeats the previously entered rule from RAM. For a k-mix the last rule with the current k is repeated.

16.14 Loading a single rule

If **l** is selected in section 16, a top-right filing prompt (section 35.2) is presented allowing a single rule to be loaded from a file. Enter the filename (see chapter 35 for filename limitations). The defaults are `myrul_`vv`.rul` for rcode, `myvco_`vv`.vco` for kcode, and `mytco_`vv`.tco` for tcode, for example `myrul_v2.rul`, where v is the value range. If successfully loaded, the rule is presented again as a bit/value pattern (section 16.4) where it can be reconfirmed or amended.

Note that both the value-range v and neighborhood size k in the file must be the same as in the current setup. If not, error messages will appear as described in section 35.3.

16.15 Automatic saving of last rule

After a single rule has been set, it will be automatically saved as the "last rule" of that rule type, with a different filename for rcode, kcode or tcode, and $[v, k]$. The "last rule" filename has the following format: `l_vxky.ext` where the extension is `.rul` for rcode, `.vco` for kcode and `.tco` for tcode, for example, `l_v4k5.rul`.

16.16 Single rule file encoding

The binary file defining a single rule is encoded[5] starting with 2 leading bytesas follows: Byte 0 = value-range v, byte 1 = neighborhood k. The rest of the rule-table size S (from 0 to S-1) is set as bits in successive bytes.

S depends on the rule type, v and k as follows:

$$\begin{aligned} \text{rcode} \ldots \quad & S = v^k \\ \text{kcode} \ldots \quad & S = (v+k-1)!/(k!(v-1)!) \\ \text{kcode} \ldots \quad & S = v \times k \end{aligned}$$

The number of bits required for each rule-table output (V_{bits}) is as follows:

$$\begin{aligned} v{=}2 \ldots \quad & \text{1 bit } (V_{bits} = 1) \text{ as in the old binary version of DDLab.} \\ v{=}\ 3 \text{ or } 4 \ldots \quad & \text{2 bits } (V_{bits} = 2) \\ v{=}\ 5, 6, 7, 8 \ldots \quad & \text{3 bits } (V_{bits} = 3) \end{aligned}$$

A single rule encoding therefore requires 2 leading bytes plus the rule-table bytes R, where $R = \left\lceil \frac{S \times V_{bits}}{8} \right\rceil$ bytes, the upper absolute value, minimum 1 byte.

16.17 Create a similar kcode with increased k

only for kcode in TFO-mode

Enter **k** in section 16 to create and save a kcode with an increased neigborhood (k+) by inserting a random string of the correct length centrally within the current kcode. This may preserve some aspects of the original dynamics, as in figure 16.14. The following top-right prompt is presented,

create similar to k6 kcode and save, enter greater k(7-25): *(if the current k=6)*

[5] In the binary version of DDLab, the old style encoding started with the neighborhood k at byte 0. In the new style encoding byte 0 is reserved for the value-range v, and subsequent bytes are displaced by one. Old style files are nevertheless compatible for loading in the present version of DDLab. For wiring+rulemix encoding see section 19.3 and for seed encoding see section 21.9.

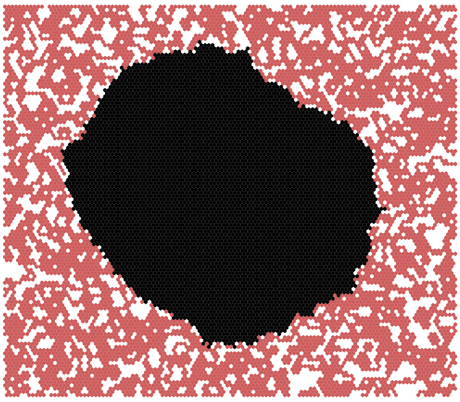


Figure 16.14:
A $v3k7$ kcode, created by inserting a random string within a k=6 kcode majority rule.The figure shows an evolved space-time snapshot from a random initial state on a 122×122 hexagonal lattice. Some aspects of a majorty rule are retained. Black blobs may emerge that either shrink and disappear, or slowly expand and take over.

The two rules are printed in the terminal as shown below, and the new string is automatically saved as the last kcode (`l_v`xky`.vco`), for example `l_v3k7.vco`, which can be loaded with **repeat-p** after backtracking and re-selecting the new k. The program reverts to the prompt in section 16.1.

k-code rule-mix – for example

```
v_tablecodemax[6]=27
2222221220110001110001111000
v_tablecodemax_new=35 nhood_new=7
222222122011000020120001110001111000
```

16.18 Show the rule in the terminal

Select **x** in section 16.1 to show the rule data immediately in the terminal, and to present a top-right prompt (section 16.18.3 for the rule data in more detail. For kcode, this includes a further option to swap values (colors) to make an equivalent kcode (section 16.18.5).

16.18.1 Immediate rule data

Immediate rule data will appear in the terminal as follows (including the cell index for a rulemix),

rcode – for example

```
v3k3 rcodeSize=27
(hex) 02 2a 94 06 62 15 11
(rcode-table:2-0)
002022221100012120201110101
vfreq=8+9+10=27 ld=0.63 ld-r=0.944 zl=0.555556 zr=0.444444
```

kcode – for example

```
v3k5 kcodeSize=21
(hex) 00 12 81 51 89 15
(kcode-table:2-0)
001022001110120210111
vfreq=4+9+8=21
```

tcode – for example

```
v3k7 tcodeSize=15
(hex) 22 06 42 26
(tcode-table:2-0)
202001210020212
vfreq=6+3+6=15
```

Table 16.2: Examples of the immediate rule data in the terminal (xterm) for rcode, kcode and tcode.

16.18.2 immediate rule data for a rule-mix by hand

If a rule-mix is being set by hand in section 14.6 (including a rule-subset, or outer-totalistic), enter **x** in section 14.6.3 to show the immediate rule data for the last rule defined. The cell index will be added to the data as in the example below,

k-code rule-mix – for example

```
v3k3 kcodeSize=10 cindex=5
(hex) 02 64 28
(kcode-table:2-0)
0212100220
vfreq=4+2+4=10
```

16.18.3 rule data in more detail – vertical layout

Simultaneously with the immediate rule data, a prompt is presented for more detail, showing in particular how the neighborhoods and the neighborhood index relates to the rule-table outputs in a vertical layout. For kcode there are additional options for a horizontal layout, and for a matrix layout if the value-range v=3.

for rcode or tcode – for example

print rcode-table(S=19) to xterm (vert)-v: *(or tcode-table)*

for kcode – matrix-m if v=3 only – for example

swap values-s, print kcode-table(S-21) to xterm (vert/horiz/matrix)-v/h/m:

Enter **v** to show the rule-table details with a vertical layout – the same rules as in table 16.2.

rcode v3k3

```
 26: 222 -> 0
 25: 221 -> 0
 24: 220 -> 2
 23: 212 -> 0
 22: 211 -> 2
 21: 210 -> 2
 20: 202 -> 2
 19: 201 -> 2
 18: 200 -> 1
 17: 122 -> 1
 16: 121 -> 0
 15: 120 -> 0
 14: 112 -> 0
 13: 111 -> 1
 12: 110 -> 2
 11: 102 -> 1
 10: 101 -> 2
  9: 100 -> 0
  8: 022 -> 2
  7: 021 -> 0
  6: 020 -> 1
  5: 012 -> 1
  4: 011 -> 1
  3: 010 -> 0
  2: 002 -> 1
  1: 001 -> 0
  0: 000 -> 1
   \  \      \
    \  \    outputs
     \  neighborhoods
     index
```

kcode v3k5

```
  0: 0 0 5  -> 1
  1: 0 1 4  -> 1
  2: 0 2 3  -> 1
  3: 0 3 2  -> 0
  4: 0 4 1  -> 1
  5: 0 5 0  -> 2
  6: 1 0 4  -> 0
  7: 1 1 3  -> 2
  8: 1 2 2  -> 1
  9: 1 3 1  -> 0
 10: 1 4 0  -> 1
 11: 2 0 3  -> 1
 12: 2 1 2  -> 1
 13: 2 2 1  -> 0
 14: 2 3 0  -> 0
 15: 3 0 2  -> 2
 16: 3 1 1  -> 2
 17: 3 2 0  -> 0
 18: 4 0 1  -> 1
 19: 4 1 0  -> 0
 20: 5 0 0  -> 0
  \    \        \
   \    \     outputs
    \  neighborhoods
    index
```

tcode v3k7

```
14 -> 2
13 -> 0
12 -> 2
11 -> 0
10 -> 0
 9 -> 1
 8 -> 2
 7 -> 1
 6 -> 0
 5 -> 0
 4 -> 2
 3 -> 0
 2 -> 2
 1 -> 1
 0 -> 2
  \    \
   \    outputs
   totals=index
```

Table 16.3: Examples of the vertical layout for rcode, kcode and tcode, explanatory notes, for the same rules as table 16.2.

16.18.4 Additional rule data options for kcode

For kcode only, there are two additional layout options at prompt 16.18.3: horizontal layout (section 13.6.1) – enter **h**, and matrix layout (section 13.6.2) which applies only if v=3 – enter **m**. The results in the terminal are shown below (table 16.4),

kcode horizontal layout v3k5

```
2:544333222211111000000
1:010210321043210543210 --neighborhoods
0:001012012301234012345
  |||||||||||||||||||||
  001022001110120210111 --outputs
```

kcode matrix layout v3k5

```
1 1 1 0 1 2
0 2 1 0 1
1 1 0 0
2 2 0
1 0
0
```

Table 16.4: Examples of (*left*) horizontal, and (*right*) matrix, layout for the same kcode as table 16.3.

16.18.5 Swapping kcode values

Enter **s** in section 16.18.3 to swap pairs of values (colors) creating an equivalent kcode with equivalent dynamics, so both attractor basins and space-time patterns would be equivalent, which requires reordering the kcode as well as swapping values. The following top-right prompts are presented,

example for v=3
swap 2 values: enter first (0-2):1 *(1 was entered, followed by)*
first=1, enter second (0-2):2 *(2 was entered)*

The prompt then reverts back to section 16.18.3 – swap more pairs or enter **return** to continue. In this $v3k5$ example, the original kcode is 001022001110120210111 from sections 16.18.1 – 16.18.4. The swapped kcode is shown in table 16.5 in some of the same layouts as can be seen in the terminal. Figure 16.15 shows an example of the equivalent space-time patterns.

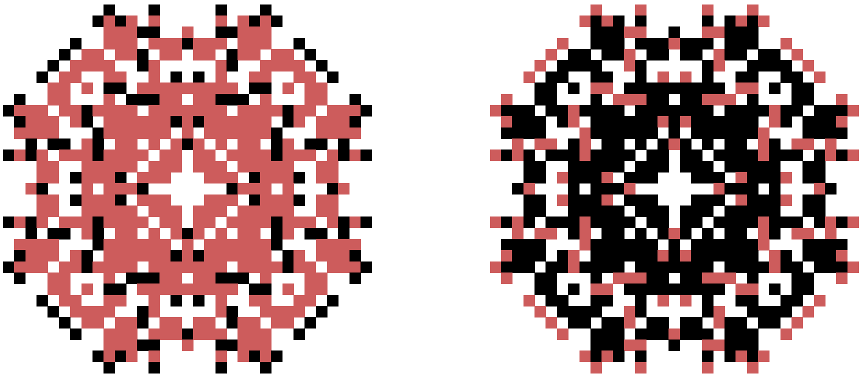


Figure 16.15: Space-time snapshots of two equivalent kcodes with swapped values 1 and 2 (red and black). The initial state was a single central cell on a 40×40 square lattice. The snapshot was taken at timestep 54.

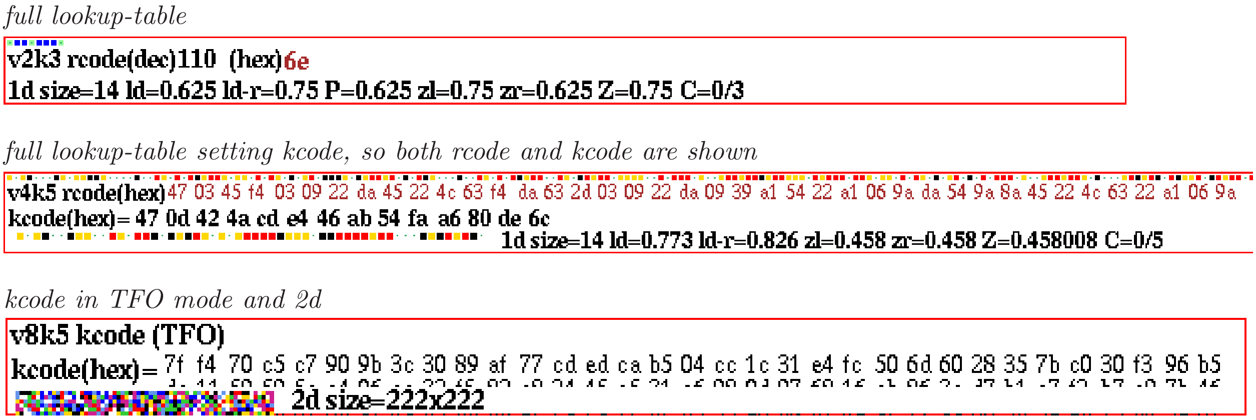
```
v3k5 kcodeSize=21           2:544333222211111000000    2 0 2 1 2 0
(hex) 01 a0 02 86 62 62     1:010210321043210543210    2 1 2 1 0
(kcode-table:2-0)           0:001012012301234012345    2 2 0 0
122000002201212021202         |||||||||||||||||||||    0 0 0
vfreq=9+4+8=21                122000002201212021202    2 2
                                                       1
```

Table 16.5: Swapped values 1 and 2 from the $v3k5$ kcode in sections 16.18.1 – 16.18.4. The swapped kcode is shown in layouts from the terminal.

16.19 The rule window

When a rule is set or updated, or selected in the network-graphic (chapter 17), it is displayed in a lower rule window. The rcode, tcode or kcode are also displayed if applicable, and TFO-mode is indicated if current. Rule-tables are shown as as bit/value patterns and in hex (as much as will fit), and in decimal (if applicable).

The window also gives v, k, rule types, network size, and for rcode – details of the λ, P, and Z parameters, canalizing inputs, and Post function data (if applicable). This window is updated as rules are transformed or mutated. Examples of rule windows and their data are shown in figure 16.16, and decoded in section 16.19.1.



full lookup-table

v2k3 rcode(dec)110 (hex)6e
1d size=14 ld=0.625 ld-r=0.75 P=0.625 zl=0.75 zr=0.625 Z=0.75 C=0/3

full lookup-table setting kcode, so both rcode and kcode are shown

v4k5 rcode(hex)47 03 45 f4 03 09 22 da 45 22 4c 63 f4 da 63 2d 03 09 22 da 09 39 a1 54 22 a1 06 9a da 54 9a 8a 45 22 4c 63 22 a1 06 9a
kcode(hex)= 47 0d 42 4a cd e4 46 ab 54 fa a6 80 de 6c
1d size=14 ld=0.773 ld-r=0.826 zl=0.458 zr=0.458 Z=0.458008 C=0/5

kcode in TFO mode and 2d

v8k5 kcode (TFO)
kcode(hex)= 7f f4 70 c5 c7 90 9b 3c 30 89 af 77 cd ed ca b5 04 cc 1c 31 e4 fc 50 6d 60 28 35 7b c0 30 f3 96 b5
2d size=222x222

Figure 16.16: Rule window examples.

16.19.1 decoding the rule window

The abbreviated headings in this example of the rule window are explained below,

(the rcode – look-up table shown as a bit pattern)
$v2k3$ **rcode(dec)186 (hex)ba** *(values shown are examples)*
1d size=14 ld=0.625 ld-r=0.75 P=0.625 zl=0.75 zr=0.25 Z=0.75 C=1/3 **0 Post=A[0]i

	key to data in the rule window
v2k3 . . .	the value-range v and neighborhood size k.
rcode . . .	the rule type, kcode or tcode is also shown if applicable.

1d size=14 . . . the network dimension and size for 1d, 2d or 3d would be shown thus (for example) **2d size=40x40** or **3d size=20x20x20**.

for a full lookup-table only

ld . . . the λ parameter.

ld-r . . . the λ ratio.

P . . . the P parameter.

zl . . . Z_{left}.

zr . . . Z_{right}.

Z . . . the Z parameter.

C=1/3 . . . the number of canalizing inputs, in this case 1 out a possible 3 (k=3).

****0** . . . shows exactly which of the k inputs are canalizing (if none this is not shown),
in this example there is 1 canalizing input, at neighborhood index 0.

Post=A[0]i . . . Post-function data, shown only if the rule qualifies, see section 14.12.

16.20 Complimentary values

Various rule and seed setting methods include an option to compliment values, for example **comp-m** in sections 16.3, 16.4, 18.4.1 and 21.4.

Transforming a value into its compliment is simple for binary (v=2) – change 1s to 0s and vice-versa. For *any* value-range v the "compliment" of a value in DDLab is defined as follows: each value a is changed to its compliment a_c by subtracting a from the maximum value $v-1$, so $a_c = (v-1)-a$. This satisfies binary, and for any v the maximum value is the compliment of zero. For odd v this means that the "central" value does not change.

The values are therefore swapped as follows,

```
v=2    v=3      v=4        v=5          v=6            v=7              v=8
1 0   2 1 0   3 2 1 0   4 3 2 1 0   5 4 3 2 1 0   6 5 4 3 2 1 0   7 6 5 4 3 2 1 0
\_/   \___/   \ \_/ /   \ \___/ /   \ \ \_/ / /   \ \ \___/ / /   \ \ \ \_/ / / /
               \___/     \_____/     \ \___/ /     \ \_____/ /     \ \ \___/ / /
                                      \_____/       \_______/       \ \_____/ /
                                                                     \_______/
```

Complimentary values as defined apply to seed options (chapter 21) as well as rule-table options.

16.21 Transforming the single rule

full lookup tables only – not TFO-mode

For full lookup tables only including kcode and tcode, but not in TFO mode, single rules that have been set can be transformed in various ways. Here is an example of the top-right prompt, though it will vary according to context, presented once a single rule has been selected,

transform rcode: solid-o invert-v comp-m neg-n ref-r, canal-C:
equiv greater k(3): (4-13), eff k-k: *(if k=3)*
save/prtx: rcode/kcode-s/S/x/X

The various methods for transforming rules are described in chapter 18, and the prompt may also be accessed from the wiring graphic – chapter 17, space-time pattern interupt options – section 32.14.2 and attractor basin complete options – section 30.5.

Chapter 17

Reviewing network architecture

This chapter describes ways of reviewing the network architecture by the following methods,

wiring matrix . . . shows the wiring (and rules if set) in a table or spread-sheet format, where the wiring can be changed.

1d, 2d or 3d wiring graphic . . . the most powerful method for tailoring the wiring and rules, and also seeing the details of the network.

Alternativly the network can be reviewed as a graph, described in chapter 20. The graph option does not allow changes to the underlying network, but the graph can be unraveled by dragging nodes and components, and rotated, rescaled, and many other manipulations performed. Default presentations include: circle, spiral, 1d, 2d and 3d.

The methods described in this chapter, the wiring matrix and wiring graphic, allow wiring and rules, set in chapters 10 – 16, to be examined, changed, and tailored to requirements, including biased random settings to pre-defined parts of the network. These are very flexible methods, and for RBN/DDN its usualy easier to set up a suitable dummy network initially, then tailor it here.

Although there are some differences in the methods for amending the network between the 1d, 2d and 3d wiring graphic formats, in most cases they are the same or similar. The 1d graphic has two alternatives. Wiring can be shown between successive time-steps, or between cells arranged in a circle. Both 1d methods apply whatever the native dimensions of the network. The 2d graphic can be applied to both 1d and 2d networks. For a 3d network, the wiring graphic shows a simultaneous display in both 2d and 3d (2d+3d), where the 2d graphic shows horizontal slices (levels) through the 3d network stacked above each other.

17.1 The network architecture prompt

see also "Reviewing wiring" section 12.7

Once special wiring (chapter 12), or both the wiring and rules, have been set, a top-right network architecture prompt is presented. The exact wording and options offered depends on other settings. The prompt can also be activated at later stages in DDLab, while drawing space-time patterns (section 32.14) or attractor basins (sections 30.4, 30.5, 30.5.1).

The prompt starts as follows, where the network sizes shown are examples,

1d network (n=150), ... *(for a 1d network)*
2d network (40x40), ... *(for a 2d network)*
3d network (15x15x15), ... *(for a 3d network)*

The first line of the prompt continues,
... **wiring only - rules not set** (*if rules have not been set*)
... **review/revise/learn, wiring and rules** (*if rules were set*)

The prompt continues as follows,

graph-g, matrix: revise-m view-M prtx-Mp
graphic: 1d+timesteps-1 circle-c, 2d-2: (*for a 1d or 2d network*)
or
graphic: 1d+timesteps-1 circle-c, 2d+3d-3: (*for a 3d network*)

For example, for a 1d network with rcode set, the prompt appears as follows,

1d network (n=150), review/revise/learn, wiring and rcode
graph-g, matrix: revise-m view-M prtx-Mp
graphic: 1d:timesteps-1 circle-c, 2d-2:

options ... *what they mean*

graph-g ... enter **g** for the network-graph (chapter 20), where network nodes and connections can be rearranged and unravelled.

matrix: ... *the “wiring matrix” section 17.2*

revise-m ... enter **m** for the network architecture in a spread-sheet type format, the wiring matrix, where the wiring can be changed.

view-M ... enter **M** for to view the wiring matrix and save matrix as a vector PostScript file 17.2.1.

prtx-Mp ... enter **Mp** to print the wiring matrix immediately in the terminal.

graphic: ... *the “‘wiring graphic” section 17.3*

1d:timesteps-1 ... enter **1** for a 1d wiring graphic where wiring is shown between successive time-steps, section 17.5, applies also to 2d and 3d.

circle-c ... enter **c** for a 1d wiring graphic arranged in a circle, section 17.5, applies also to 2d and 3d.

2d-2 ... enter **2** for a 2d wiring graphic, section 17.6, applies also to 1d.

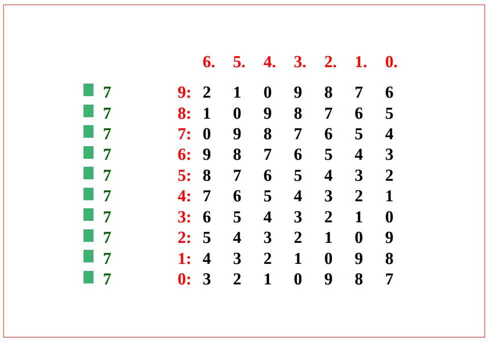
2d+3d-3: ... enter **3** for the wiring graphic in 3d, and simultaneously in 2d in successive horizontal slices, section 17.7.

17.2 The wiring matrix

Enter **m** or **M** in section 17.1 to show the network architecture in a spread-sheet type format, the wiring matrix; **M** to see the matrix or save it as a vector PostScipt file; **m** to to changethe wiring. The rules in a rule-mix (if set) are also shown, as much as will fit, but the wiring matrix window, and fonts, can be re-sized. Examples are shown in figure 17.1. Rules cannot be changed in the wiring matrix, this can be done in the wiring graphic (section 17.3).

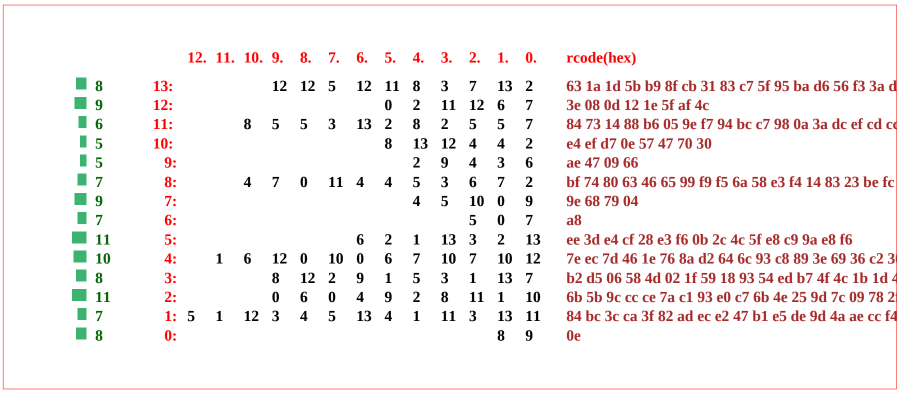
As described in section 12.6, columns give the cell's pseudo-neighborhood index, K (k-1...0). Rows give the cell network position, N (n-1...0). The 0-0 grid (or the 0-minimum n grid if the whole matrix does not fit within one window) is in the lower right hand corner. Each grid records the position in the network x, (n-1...0), to which the K'th wire of the N'th cell is connected.

Note that positions N and x are 1d indexes, even if the network is 2d or 3d. For 2d $I \times J$, to convert ij coordinates, $x = Ij+i$. For 3d $I \times J \times H$, to convert ijh coordinates, $x = IJh+Ij+i$. A column to the left of the cell index (colored green) shows the out-degree and out degree histogram of each cell.



			6.	5.	4.	3.	2.	1.	0.
🟩	7	9:	2	1	0	9	8	7	6
🟩	7	8:	1	0	9	8	7	6	5
🟩	7	7:	0	9	8	7	6	5	4
🟩	7	6:	9	8	7	6	5	4	3
🟩	7	5:	8	7	6	5	4	3	2
🟩	7	4:	7	6	5	4	3	2	1
🟩	7	3:	6	5	4	3	2	1	0
🟩	7	2:	5	4	3	2	1	0	9
🟩	7	1:	4	3	2	1	0	9	8
🟩	7	0:	3	2	1	0	9	8	7

The wiring matrix of a 1d CA, $v2k7$, n=14.



			12.	11.	10.	9.	8.	7.	6.	5.	4.	3.	2.	1.	0.	rcode(hex)
🟩	8	13:				12	12	5	12	11	8	3	7	13	2	63 1a 1d 5b b9 8f cb 31 83 c7 5f 95 ba d6 56 f3 3a d
🟩	9	12:								0	2	11	12	6	7	3e 08 0d 12 1e 5f af 4c
🟩	6	11:			8	5	5	3	13	2	8	2	5	5	7	84 73 14 88 b6 05 9e f7 94 bc c7 98 0a 3a dc ef cd cc
🟩	5	10:								8	13	12	4	4	2	e4 ef d7 0e 57 47 70 30
🟩	5	9:									2	9	4	3	6	ae 47 09 66
🟩	7	8:			4	7	0	11	4	4	5	3	6	7	2	bf 74 80 63 46 65 99 f9 f5 6a 58 e3 f4 14 83 23 be fc
🟩	9	7:									4	5	10	0	9	9e 68 79 04
🟩	7	6:											5	0	7	a8
🟩	11	5:							6	2	1	13	3	2	13	ee 3d e4 cf 28 e3 f6 0b 2c 4c 5f e8 c9 9a e8 f6
🟩	10	4:		1	6	12	0	10	0	6	7	10	7	10	12	7e ec 7d 46 1e 76 8a d2 64 6c 93 c8 89 3e 69 36 c2 3
🟩	8	3:				8	12	2	9	1	5	3	1	13	7	b2 d5 06 58 4d 02 1f 59 18 93 54 ed b7 4f 4c 1b 1d 4
🟩	11	2:				0	6	0	4	9	2	8	11	1	10	6b 5b 9c cc ce 7a c1 93 e0 c7 6b 4e 25 9d 7c 09 78 2
🟩	7	1:	5	1	12	3	4	5	13	4	1	11	3	13	11	84 bc 3c ca 3f 82 ad ec e2 47 b1 e5 de 9d 4a ae cc f4
🟩	8	0:												8	9	0e

The wiring matrix for a v=2 k-mix (k=2 to 13) with random wiring, showing the rcode in hex for each cell, as much as will fit.

Figure 17.1: Wiring matrix examples. k-1...0 indexes columns, n-1...0 indexes rows. The column on the left shows the "out-degree" of each cell, the number of output wires that link to it, also shown as a histogram. If a rule-mix was set, the rules are shown in hex, as much as will fit, in the right column **rcode(hex)** (or **kcode/tcode**). For $k \leq 3$ rcode, the decimal rule is added: **rcode(hex/dec)**, otherwise **rules not set** is shown. If the matrix was set with **m** at prompt 17.1 the wiring can be set by hand as in a spread-sheet (section12.6) and a rule in the active cell also appears in the rule window (section 16.19). If the matrix was set with **M** at prompt 17.1 there is an option to create a vector PostScript file, as in these figures.

17.2.1 Viewing the wiring matrix and creating vector Postscript

If **M** was selected in section 17.1, the wiring matrix is just displayed (enter **m** in section 17.2.2 to amend), together with the following top-right prompt,

> *(if the matrix fits one window)*
> **postScipt-P layout-l font-f jump-(j=13-0) quit-q:**
> *(for a large matrix requiring a succession of windows, this example for a 2d network, 40x40)*
> **more-ret postScipt-P layout-l font-f jump-(j=1599-0) quit-q:**

options ... *what they means*

more-ret ... for a large network that requires more than one window, enter **return** for the next section of the list, otherwise enter **return** to continue.

postScipt-P ... enter **P** to save the matrix as seen in the matrix window to a vector PostScript file. The default filename is `my_mxPS.ps` (see chapter 36).

layout-l ... enter **l** to change the matrix window width. The following top-right prompt is presented, for example,

change window width (922-231 now=462):

Enter the new width in pixels within the limits indicated.

font-f ... enter **f** to change the font size, a 3-way toggle between normal, medium and small.

jump-j(1599-0) ... enter a number within the limits indicated to jump directly to a given cell index, which will become the first entry in the top row.

quit-q ... enter **q** ro revert to the network architecture prompt in section 17.1.

For DOS there is a further option: enter **p** to print the current matrix window, with printer limitations as noted in section 5.6.3.

17.2.2 Amending the matrix

see also "Wiring by hand" section 12.6

Enter **m** in section 17.1, or **h** in in section 12.3, to display the wiring as a matrix or "spread sheet", which allows filling in the wiring positions for each cell's pseudo-neighborhood index. If **m**, the wiring will have already been set in chapters 11 and 12, so all positions will be filled but can be amended. If **h**, the matrix will start as a blank grid, to be set "by hand" (section 12.6) as in figure 12.5. In both cases, the following top-right reminder is presented,

> **hand wire/revise:jump-j** *(for a 2d network, 40×40)*
> **enter wiring positions 0-1599 (return on blank/0=random**
> **move-arrows more/complete-m layout-l font-f quit-q**

These options are also described in section 12.6 "Wiring by hand", but this section assumes a wiring matrix that has already been set – selected with **m** in section 17.1. Move around the matrix with the left/right/up/down arrow keys. Enter the new position at the flashing green cursor and complete the entry by moving to another grid with an arrow key or **return**. On zero, just **return** gives a random position. A rectangle is drawn around any position covered. An entry outside the network limits will be ignored.

Large networks may require several successive windows to display the matrix. Enter **m** to see the next window. Moving beyond the top or bottom row in the current window with the arrow keys or **return** also brings up the preceding or next window. **return** on the very last (bottom right) entry makes the program continue. Enter **j** to jump to a new cell index, the following prompt is presented,

jump to index (1599-0): *(for a 2d network, 40×40)*

Enter the new cell index, which will become the first entry in the top row. Options **l** or **f** to alter the presentation of data and the amount visible in the matrix window are described in section 17.2.1 above. Enter **q** to accept the wiring and revert to the prompt in section 17.1.

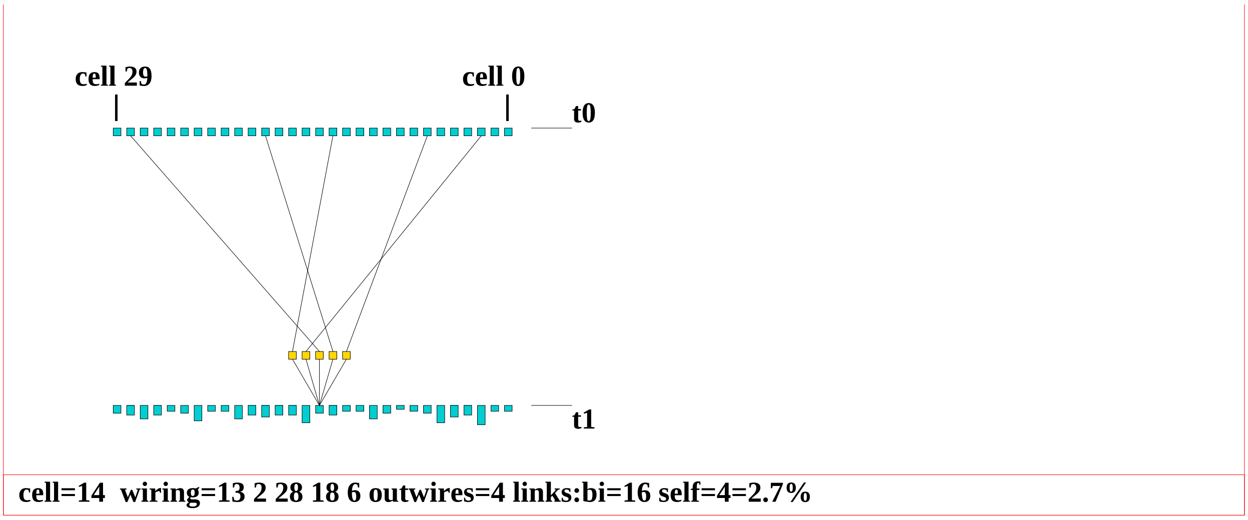
17.3 The wiring graphic

The wiring graphic is a diagram where the wiring and rules can be viewed, amended and reset by flexible methods described in the rest of this chapter.

Enter **1**, **c**, **2** or **3** (for 1d time-steps, 1d circle, 2d or 3d) in section 17.1.

options ...	*what they mean*
1d:timesteps-1 ...	for a 2d wiring graphic shown between successive time-steps.
circle-c ...	for a 2d wiring graphic shown between cells arranged in a circle.
2d-2 ...	for a 2d wiring graphic, which may also be selected for a 1d network.
2d+3d-3 ...	for a 3d wiring graphic which only applies to a 3d network.

Note the native dimensions of the network can differ from the dimensions chosen for the wiring graphic. Both 1d methods apply whatever the native dimensions of the network. The 2d graphic



cell=14 wiring=13 2 28 18 6 outwires=4 links:bi=16 self=4=2.7%

Figure 17.2: The 1d wiring graphic showing the wiring between successive time-steps, k=5, n=30. Each cell's "outwires" are represented by the height of the lower cells, at time-step t_1.

can be applied to both 1d and 2d networks. However the 3d graphic only applies to a 3d network. The display and amendments apply to the “active cell” or to a pre-defined block of cells.

17.4 The wiring graphic reminder

A wiring graphic reminder of the options available appears in a top-right window at the same time as the wiring graphic. The options vary between the 1d, 2d, and 3d wiring graphic but there are also many similarities. Various context dependent options are presented for analyzing the network, and amending the wiring or rule/s for a single cell, a group of cells – a ”block”, or for the network as a whole[1].

The first line of the reminder shows the network dimensions and size. If the rules have not been set, the first line of the prompt reads **wiring only** instead of **wiring/rules**. If a block is active, its botton right and top left coordinates are shown instead of the coordinates of the active cell, i.e. **rewire 2d block 3,3-8,8:** instead of **rewire 2d cell 26,24:**.

Examples of the 1d, 2d, and 3d wiring graphic reminders are shown below.

1d time-step and circle wiring graphic

```
1d (n=150) wiring/rules: move-arrows, jump-j
one cell: right/left arrows, 15 cells: up/down arrows, redraw-T Postscript-P
block-b tog:all-g pseudo-p, in/out-i/o avZ-z learn-l
rewire 1d cell 74: untangle-u hand-h rnd-r/w/R special-s local:1d-1
change k=8(max 13)-k, kill-K del-d, rule:Save/rev/trans-S/v/t, Derrida plot-D
file/data-f hist:In(k)/Out/Both-I/O/B, reset-q cont-ret:
```

2d wiring graphic

```
2d (40x40) wiring/rules: move-arrows, jump-j
exp/contr-e/c up/down-u/d redraw-T Postscript-P toghex-x
block-b tog:block-g links-n pseudo-p index-i avZ-z
rewire 2d cell 26,24: hand-h rnd-r/w/R special-s local:1d-1 2d-2
change k=8(max 13)-k, kill-K, rule:Save/rev/trans-S/v/t, Derrida-D
file/data-f hist:In(k)/O/B-I/O/B, reset-q cont-ret:
```

3d wiring graphic

```
3d (20x20x20) wiring/rules: move-arrows, levels up/down-[/], jump-j
exp/contr-e/c/E/C up/down-u/d redraw-T Postscript-P
block-b tog:block-g links-n pseudo-p index-i avZ-z
rewire 3d cell 5,5,5: hand-h rnd-r/w/R special-s local:1d-1 2d-2 3d-3
change k=6(max 13)-k, kill-K, rule:Sace/rev/trans-S/v/t, Derrida-D
file/data-f hist:In(k)/Out/Both-I/O/B, reset-q cont-ret:
```

[1] Note that while interrupting incomplete attractor basins, though the wiring graphic can be accessed, any options for changing wiring or rule/s are deactivated and do not appear in the prompt.

17.4.1 wiring graphic options summary

The wiring graphic options options activate as soon as the key is pressed, without pressing return – some have secondary prompts. Below is just a brief summary following the order in which the options are listed in the reminders. More details are given in further sections in this chapter.

presentation options . . . *what they mean*

move-arrows, jump-j . . . use the arrow keys to move around the network or enter **j** to jump to a particular cell index.
For 1d left/right arrows (section 17.5.2).
For 2d left/right/up/down arrows (section 17.6.2).

levels up/down-[/] . . . *(3d graphic only)* move with left/right/up/down arrows within each level (or jump) as for 2d above, and use the square braces, [for up,] for down, to move between levels (section 17.7.2).

exp/contr-e/c . . . *(2d and 3d graphic only)* to expand or contract a 2d wiring graphic (section 17.6.7), or the 2d version of the 3d wiring graphic (section 17.7.7).

exp/contr-E/C . . . *(3d graphic only)* to expand or contract the 3d version of the 3d wiring graphic (section 17.7.7).

up/down-u/d . . . *(2d and 3d graphic only)* to shift the 2d version of the 3d wiring graphic to see the relevant part (section 17.7.8). Also works for the 2d wiring graphic (17.6.8).

redraw-T . . . to redraw the current wiring graphic.

***Postscript-P** . . . to redraw and create a vector PostScript file of the current wiring graphic (section ??).

toghex-x . . . *(2d graphic only)* to toggle square and hex layout.

block-b . . . define a block of cells in 1d, 2d or 3d, for resetting the wiring or rules (sections 17.5.3, 17.6.5, 17.7.5).

tog: *toggle options* . . .

all-g . . . toggle for the active cell, showing the active cell on its own, or the the active cell agains a background of the block (if set) or all network connections (if a block is not set).

links-n . . . *(2d and 3d graphic only)* a 5-way toggle for alternative presentations of wiring relative to either the active cell block. For the active cell these options may be used with **pseudo-p** below. (sections 17.6.3 and 17.7.3).

pseudo-p . . . toggle showing/omitting the pseudo-neighbourhood.

index-i . . . *(2d and 3d graphic only*) Toggle the index numbers in the pseudo-neighbourhood. The numbers shown up if the scale is large enough.

miscellaneous options . . .

in/out-i/o . . . *(1d time-steps and circle only)* show the indirect inputs and outputs past time-steps, the “degrees of separation” between cells (sections 17.5.6 and 17.5.7).

***avZ-z** . . . *(rcode-mix and/or k-mix)* to calculate the average and weighted average Z and λ parameters (section 17.8.2).

***learn-l** . . . implement the learning/forgetting routine (chapter 34).

rewire: *options* . . . *shows the coordinates of the cell, or block if active.*

untangle-u . . . *(1d time-steps and circle only)* "untangle" the wiring of the cell, block, or the whole network, and reset the rules for equivalent dynamics (section 17.5.8).

***hand-h** . . . rewire the active cell by hand (section 17.8.4).

***rnd-r** . . . to randomly rewire the active cell or block. **r** respects any biases set in **special-s** below, with a local zone equal to the shortest axis for 2d and 3d).

***rnd-w** . . . *(if a block is active)* to randomly rewire cells *outside* the active block, with biases as **rnd-r** above.

***rnd-R** . . . to bias the wiring according to the k distribution in a k-mix, allowing a power-law distribution of outputs as well as inputs (section 9.8.2). For homogeneous k, **R** gives totaly unbiased random wiring.

special-s . . . set "special" wiring biases, which can then be applied to the active cell or block with **r** above (section 17.8.6).

***local:1d-1 2d-2 3d-3** . . . set local CA wiring for the cell or block (section 17.8.7).

***change k=8(max 13)-k** . . . *(rcode-mix set)* change k up or down for the cell or block, up to max-k (section 17.8.8).

***kill-K** . . . *(rcode-mix set)* kill or neutralize a cell by cutting all its links, both inputs and outputs, except for one input to itself. The rcode is also set for effective k=0 (section 17.8.9).

del-d . . . *(1d networks and wiring graphics, k-mix, rcode set, main prompt sequence, not in TFO mode)* delete a cell from the network, and shorten the network by one (section 17.5.9).

rule: *options* . . .

***Save-S** . . . save the rule for a cell,

***rev-v** . . . revise or load the rule for a cell, and copy to a block section 17.8.10.

***trans-t** . . . transform the rule or rules, set Canalizing inputs.

more miscellaneous options . . .

***Derrida plot-D** . . . draw the Derrida plot for the network, described in chapter 22.

***file/data-f** . . . filing, save and load the network architecture.

***hist:In(k)/Out/Both-I/O/B** . . . show a histogram of the frequency distribution of inputs (i.e. k), outputs, or both (i.e all connections) in the network.

***reset-q** . . . backtrack to redisplay the network (section 17.1).

Options marked with an asterisk such as ***rnd-r** apply jointly for 1d, 2d and 3d networks – most of these options are described further in section 17.8. Other options apply more specifically, or uniquely, to a particular dimension – these options are described further in sections 17.5 for 1d, 17.6 for 2d, and 17.7 for 3d.

17.5 Wiring graphic, 1d

Enter **1** or **c** in section 17.1 for a 1d graphic, which shows how a particular cell (the "active cell"), or a pre-defined 1d block of cells, is wired in a 1d, 2d or 3d network.

If **1** is entered, the wiring is shown between two time-steps, from time-step t_0 to t_1, as in figure 17.2 and elsewhere. Network cells at t_0 are shown in a row above cells at t_1, and the out-degree of each cell is indicated by the height of the cell's representation at t_0 (the cell wiring out-degree histogram).

If **c** is entered, the wiring is shown between cells, represented as nodes or radial lines, arranged in a circle, as in 17.3 and elsewhere. The zero index is due east and increases clockwise. The cell index is shown within each disk for networks smaller than about $n = 40$. For networks greater than about $n = 162$, cells are shown as short radial lines. Each cell's "outwires" are represented by the length of radial lines outside each cell (the cell wiring out-degree histogram).

In both presentations, an option allows the pseudo-neighborhood to be omitted, showing just direct links. The active cell is moved around the network with the arrow keys, and its possible to

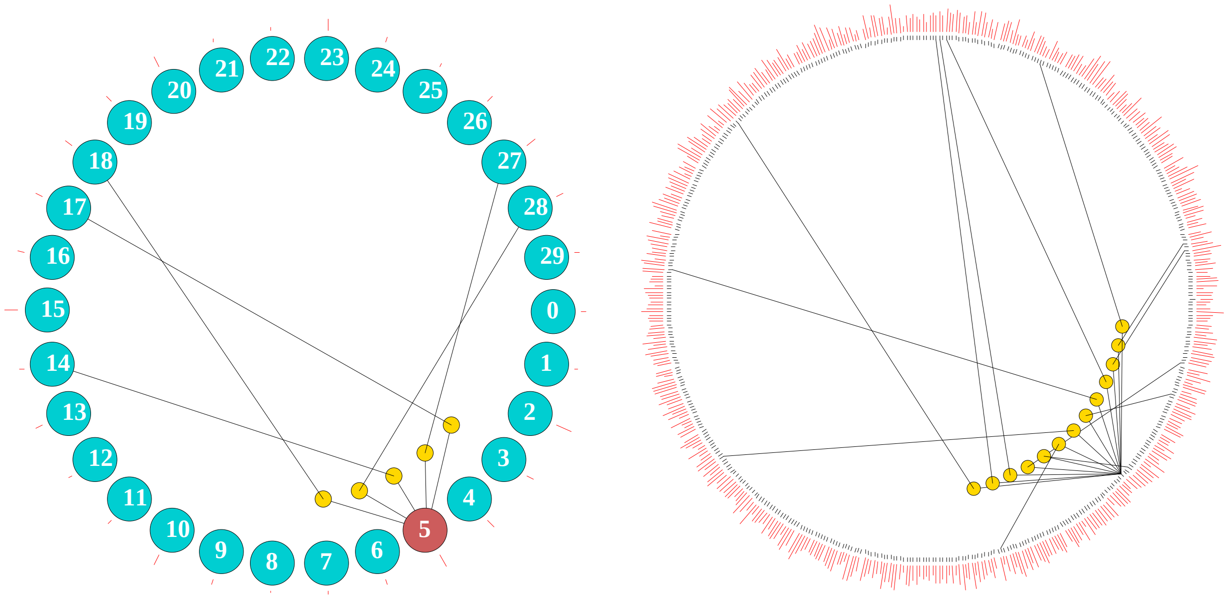
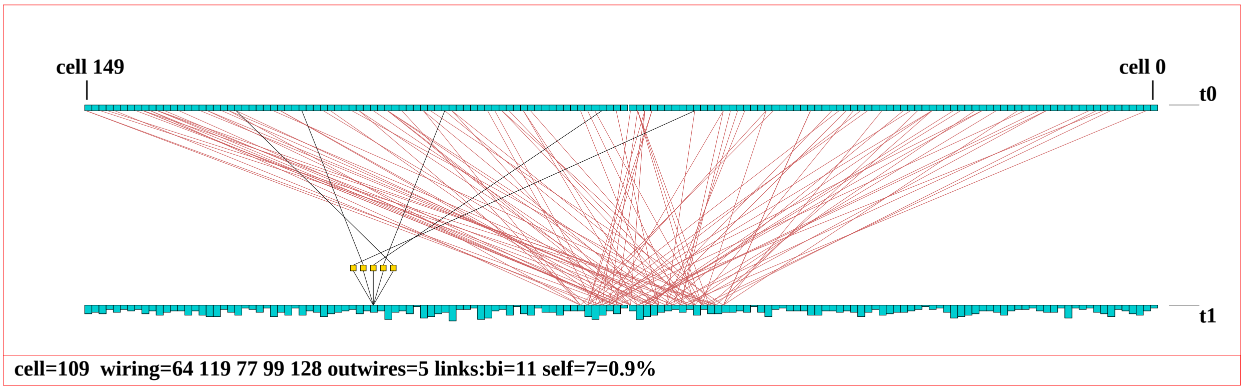
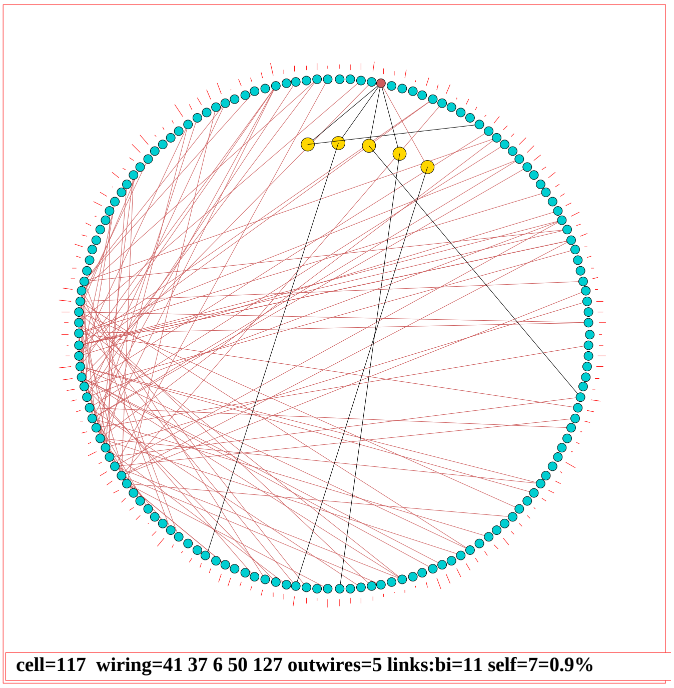


Figure 17.3: Examples of the 1d circle wiring graphic. *left*: the same network as in figure 17.2, k=5, n=30. *right*: k=13, n=500. The "active" cell is moved by the arrow keys. Each cell's "outwires" are represented by the length of the red radial lines outside the circle. The cell index is shown for networks smaller than about $n = 40$. For networks greater than about $n = 162$, the nodes themselves are not shown, but just the radial lines.



cell=109 wiring=64 119 77 99 128 outwires=5 links:bi=11 self=7=0.9%



cell=117 wiring=41 37 6 50 127 outwires=5 links:bi=11 self=7=0.9%

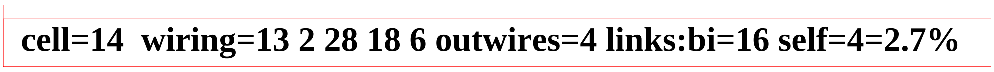
Figure 17.4: The 1d wiring graphic, showing wiring to a block of 11 cells, k=5, $n = 150$. *Above*: As time-steps. *Left*: As a circle. The block was defined from cell 60-80. The "active cell" (109) is still visible, and can be moved with the arrow keys usual.

"jump" to a new location. The rule for the active cell appears in the rule window described in section 16.19.

Sections 17.5.1-17.5.9 below explain some of the options specifically for 1d in more detail that were summarised in section 17.4.1.

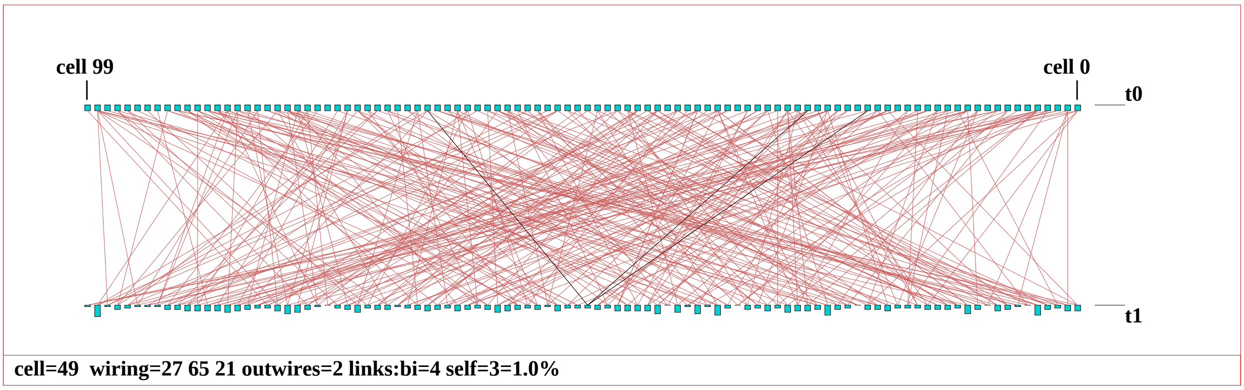
17.5.1 Data - 1d wiring graphic

Various data about the active cell's wiring are shown below the 1d graphic. For example, in figure 17.2, a $k = 5$ network, with mixed rules and non-local wiring, the data is as follows,

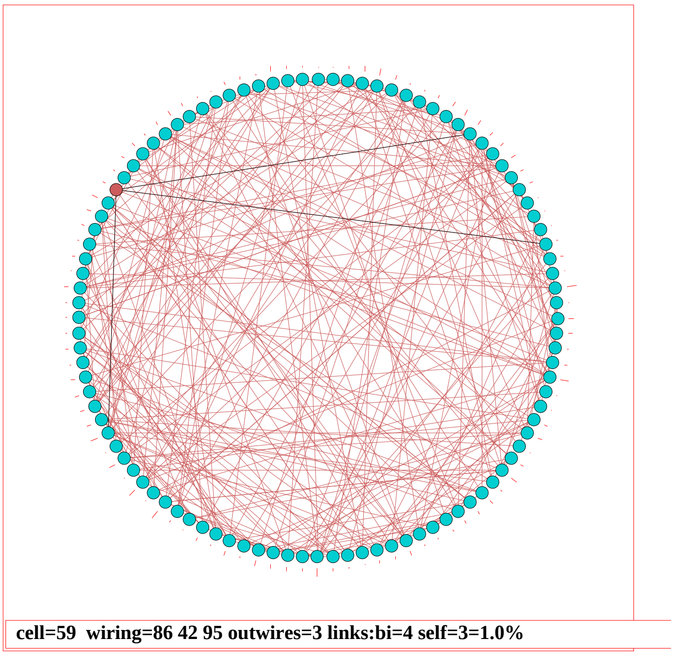


cell=14 wiring=13 2 28 18 6 outwires=4 links:bi=16 self=4=2.7%

See section 17.8.1 to decode this data. The rule and rule data for the active cell are also shown (if the rule/s have been set) in the rule window described in section 16.19.



cell=49 wiring=27 65 21 outwires=2 links:bi=4 self=3=1.0%



cell=59 wiring=86 42 95 outwires=3 links:bi=4 self=3=1.0%

Figure 17.5: The 1d wiring graphic, showing the wiring of the whole network, k=3, n = 100. If a block is not set, **g** toggles the full wiring on and off. In these examples, the pseudo-neighborhood was toggled off with **p** to show just the (black) direct links to the active cell. *Above*: As time-steps. *Left*: As a circle.

17.5.2 Moving or jumping between cells, 1d

To move the active cell, use the arrow keys. The left and right arrow keys move by one cell, the up and down arrow keys move by by $n/10$ cells for $n > 20$ or by 2 cells otherwise. Alternatively, enter **j** to jump to a specific cell index. The following prompt is displayed,

> **jump to index (1295-0):** *(for example)*

17.5.3 Defining a block, 1d

Enter **b** in section 17.4 to define a block of cells. The following top-right prompt is presented,

> **1d block-1, all-a single cell-def:**
> *(then, if* **1** *is selected...)*
> **1d block (0-49, def 102), low:** **high:**

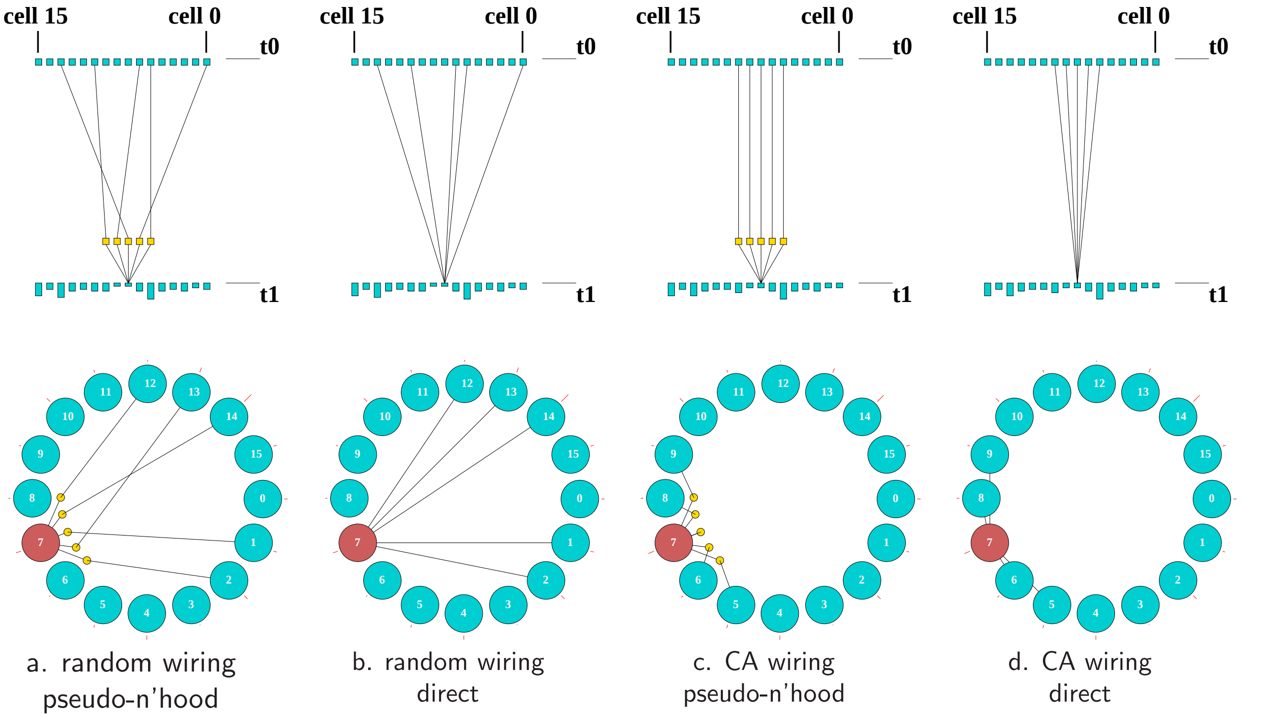


Figure 17.6: Examples of the 1d wiring graphic, k=5, $n = 20$. (a) and (c) include the pseudo-neighborhood, (b) and (d) show just direct wiring, goggle between with **p**. (a) and (b) have have random wiring, (c) and (d) have their wiring changed to 1d CA by entering **1**. *top row*: As time-steps. *bottom row*: As a circle.

Enter **1** to define the block, **a** to define the whole network as a block. If **1** is entered, a subsequent prompt is presented for the low, then high, cell indexes delimiting the block.

The wiring of all the cells in the block will be shown in the graphic as direct links, colored light red. The active cell will still be visible and can be moved as usual. For a 1d network shown in 2d, the prompt is as described in section 17.6.5

An active block is indicated in the wiring graphic reminder (17.4) for example

rewire 1d block 22-33: ...

To deactivate the block enter **b** in section 17.4, then **return**. If the block is active and visible the following **rewire** options ...

... **untangle-u** ... **rnd-r/w/R special-s local:1d-1**
change k=8 (max 13)-k ... *(for example)*

... will apply to just the block, not to the active cell if it is outside the block.

The following **rule** options apply to the active cell, but if the block is active and visible the new rule can then be copied to the block.

... **rule:Save/rev/trans-S/v/t**

17.5.4 Toggling the block, 1d

Enter **g** in section 17.4 to toggle the active block, making it either visible or invisible, without deactivating it. Note that options in section 17.5.3 affect a block only if it is both active and visible. If no block is active, toggling with **g** just shows the the wiring in the whole network.

17.5.5 Include the pseudo-neighborhood, or direct wiring only, 1d

The default wiring representation for the active cell includes the pseudo-neighborhood. Alternatively just the direct connections may be shown, always the case for a block. Enter **p** to toggle between the two (figure 17.6).

17.5.6 Recursive inputs to a cell, 1d

Enter **i** in section 17.3 to show all recursive inputs to a cell, whether direct or indirect, showing the *input* degrees of separation" between cells. This will include inputs to inputs, and so on, relative to past time-steps, showing the potential *upstream* influence on the cell from past network dynamics. This is shown in the direct wiring format (section 17.5.5).

Initially, just the wiring from the previous time-step appears, with following prompt displayed in the top left hand corner of the wiring window,

step 1, inputs=2/100, step-s all-def: *(for a k=2, n=100 network)*

Enter **return** to generate all inputs in one go. Enter **s** to show the inputs in steps, by entering **return**, or enter **q** to quit early. In both cases, inputs that relate to different past time-steps are shown in different colors (cycling through about 7 colors). A small insert in the top right hand corner of the wiring window graphically indicates the fraction of cells connected by inputs so far.

Intermediate prompts show the number of inputs so far, for example,

step 4, inputs 27/100=27% cont-ret: *(values shown are examples)*

Once all possible input cells have been found, the following prompt appears in the top left hand corner of the wiring window,

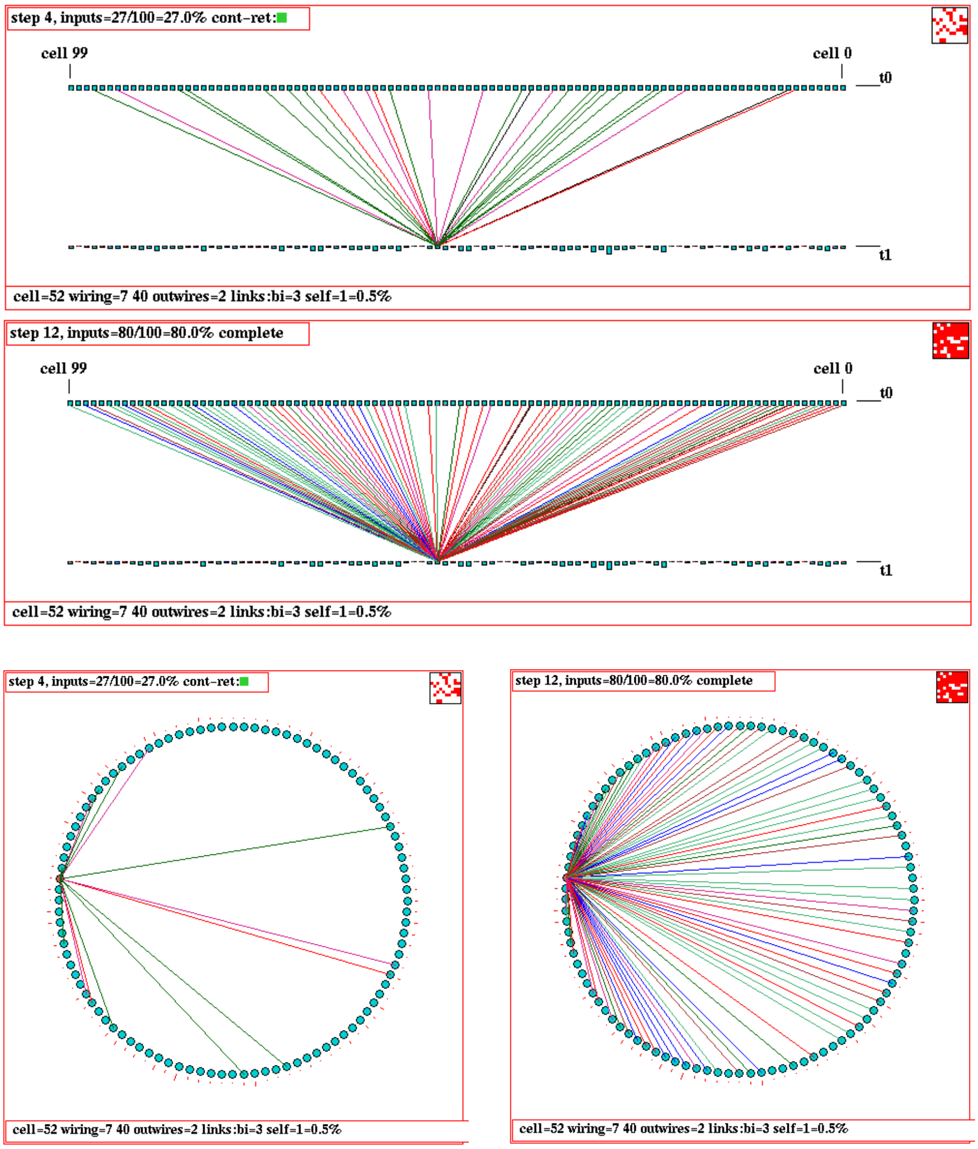
step 12, inputs 80/100=80% complete *(values shown are examples)*

Enter **return** to reactivate the main wiring reminder (section 17.4) and revert to the normal wiring graphic functions.

17.5.7 Recursive outputs from a cell, 1d

Enter **o** in section 17.4 to show all recursive outputs from a cell, whether direct or indirect, showing the "*output* degrees of separation" between cells, the converse of of recursive inputs in section 17.5.6. This will include outputs from outputs, and so on, relative to future time-steps, showing the potential influence from the cell on future network dynamics *downstream*.

Initially just the immediate wiring outputs (if any) will be shown from t_0 to t_1. The number of these outputs may be less than that shown as "outwires" in the 1d wiring window data (see section 17.5.1) because there may be duplicate output wires which are only counted once. The following prompt is displayed in the top left hand corner of the wiring window,



step 4, inputs=27/100=27.0% cont-ret:🟩

cell=52 wiring=7 40 outwires=2 links:bi=3 self=1=0.5%

step 12, inputs=80/100=80.0% complete

cell=52 wiring=7 40 outwires=2 links:bi=3 self=1=0.5%

step 4, inputs=27/100=27.0% cont-ret:🟩

cell=52 wiring=7 40 outwires=2 links:bi=3 self=1=0.5%

step 12, inputs=80/100=80.0% complete

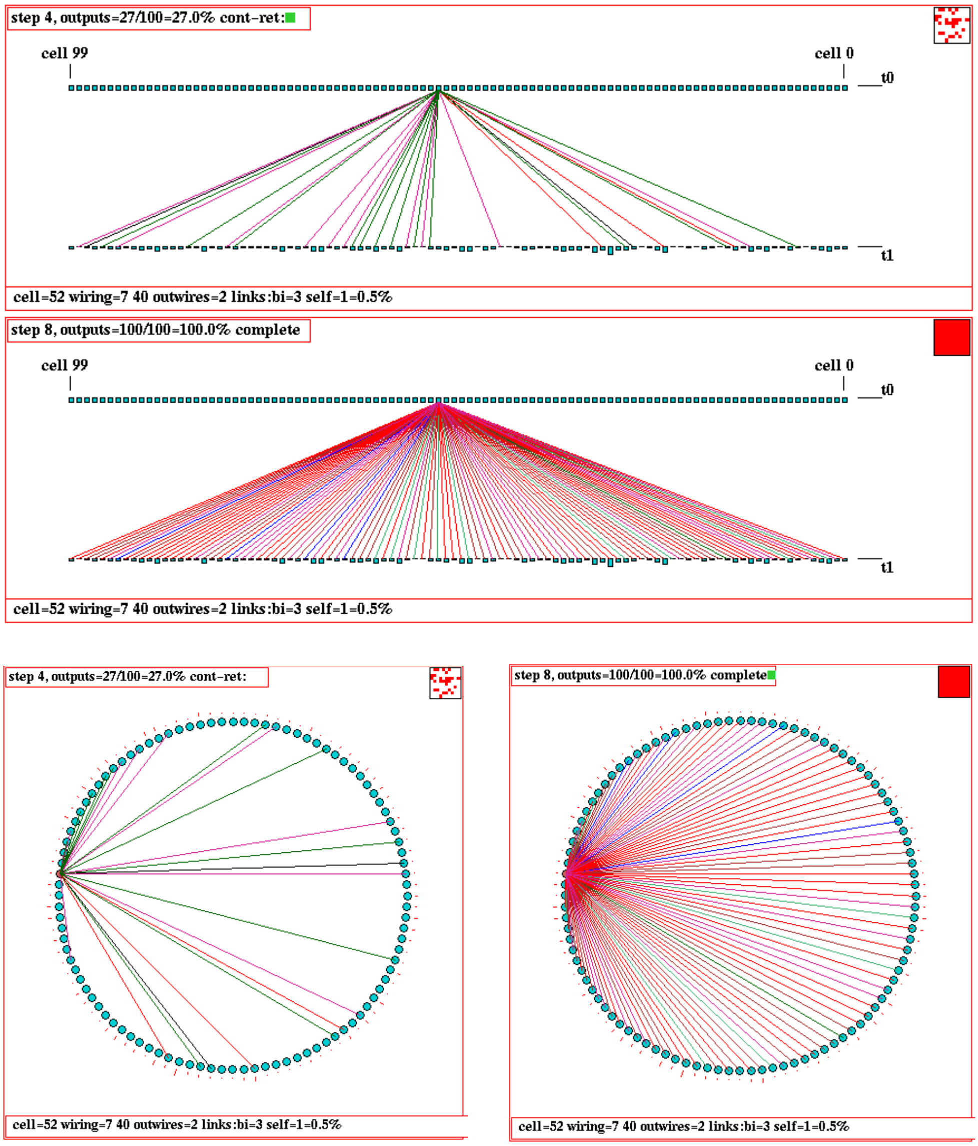
cell=52 wiring=7 40 outwires=2 links:bi=3 self=1=0.5%

Figure 17.7: Recursive inputs (direct and indirect) to a given cell for a $k = 2$, $n = 100$ RBN. *Above*: As time-steps. *Below*: As a circle. Firstly, inputs are shown after 4 backward steps with 27 cells reached. Secondly, the complete inputs are shown after 12 backward steps with 80 cells reached. A small 2d insert in the top right hand corner of the wiring window graphically indicates the fraction of cells connected so far.

step 1, outputs=2/100, step-s all-def: *(for a k=2, n=100 network)*

Enter **return** to generate all outputs in one go. Enter **s** to show the outputs in steps, by entering **return**, or enter **q** to quit early. In both cases, outputs that relate to different future time-steps are shown in different colors (cycling through about 7 colors). A small insert in the top right hand corner of the wiring window graphically indicates the fraction of cells connected by outputs so far.

Intermediate prompts show the number of outputs so far, for example,



step 4, outputs=27/100=27.0% cont-ret:🟩

cell=52 wiring=7 40 outwires=2 links:bi=3 self=1=0.5%

step 8, outputs=100/100=100.0% complete

cell=52 wiring=7 40 outwires=2 links:bi=3 self=1=0.5%

step 4, outputs=27/100=27.0% cont-ret:

cell=52 wiring=7 40 outwires=2 links:bi=3 self=1=0.5%

step 8, outputs=100/100=100.0% complete🟩

cell=52 wiring=7 40 outwires=2 links:bi=3 self=1=0.5%

Figure 17.8: Recursive outputs (direct and indirect) from a given cell for a k=2, n=100 RBN. *Above*: As time-steps. *Below*: As a circle. Firstly, outputs are shown after 4 forward steps with 27 cells reached. Secondly, the complete outputs are shown after 8 forward steps with all 100 cells reached. A small 2d insert in the top right hand corner of the wiring window graphically indicates the fraction of cells connected so far.

step 4, outputs=27/100=27% cont-ret: *(values shown are examples)*

Once all possible output cells have been found, the following prompt appears in the top left hand corner of the wiring window,

step 8, outputs=100/100=100% complete *(values shown are examples)*

Enter **return** to reactivate the wiring graphic reminder (section 17.4) and revert to the normal wiring graphic functions.

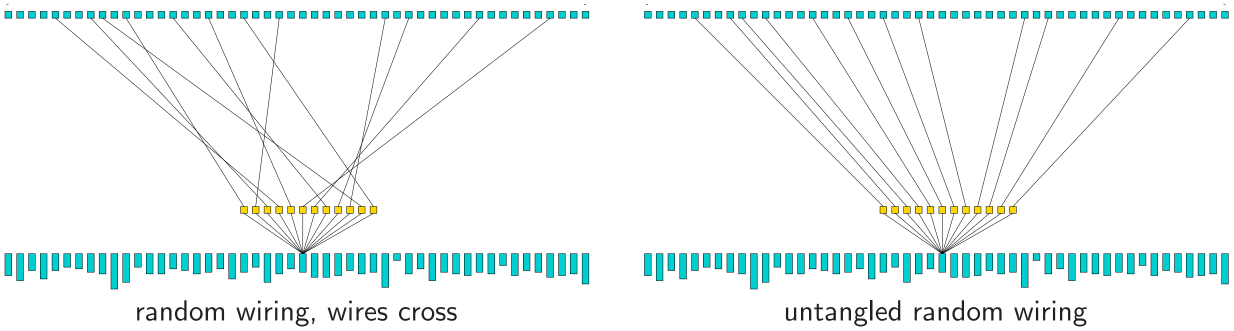


Figure 17.9: Untangling the wiring: wire connection points to the 1d pseudo-neighborhood a cell are rearranged so that the wires do not cross, and the rcode (if set as part of a rule-mix) is automatically transformed to give equivalent behavior. In TFO mode there is no need to transform the rule. In this example k=12, n=50.

17.5.8 Untangling the wiring

Enter **u** in section 17.4 to "untangle" the wiring for the active cell, the defined block if visible, or all cells in the network so that wire connection points to the pseudo-neighborhood (when shown in 1d) do not cross each other (figure 17.9) – the wiring positions in the network itself are unchanged. This option applies in the 1d wiring graphic (to 2d and 3d networks as well as 1d) but the effect is best seen in 1d time-steps rather than the alternative circle presentation.

The following top-right prompt is presented,

untange 1d wiring: all-a, single cell-(def): *(for the active cell)*
or if a block is active and visible
untange 1d wiring: block 11-22, all-a, single cell-(def): *(for example)*

If an rcode-mix has been set, rules are transformed for equivalent dynamics[2]. For a network with a single rcode network (section 14.1) the wiring will be untangled, but the rcode will remain unchanged – to get arounf this, set the rcode-mix where all the rules are the same (section 14.4.3).

Note that in TFO-mode there is no need to transform rules when untangling because changing the connection points of wires to the pseudo-neighborhood has no effect on dynamics.

17.5.9 Deleting a cell

(1d networks and wiring graphics, k-mix, rcode set, main prompt sequence, not in TFO mode)

Enter **d** in section 17.4 to delete a cell from the network and shorten the network by one cell. This option is only available in the main prompt sequence, not while interrupting space-time patterns or attractor basins. Backtrack to the main prompt sequence if necessary. In addition, the base network itself must be 1d and have mixed-k, with the rcodes set (so not in TFO mode). When a cell is deleted, all links, both inputs and outputs, between the cell and other cells will be cut, and cell indexes greater than the deleted cell will be reduced by one.

[2] If the order of the k inputs in the pseudo-neighborhood is changed, the rcode is transformed to achieve identical behavior. Thus there are k! equivalent pseudo-neighborhood orders and corresponding rcode.

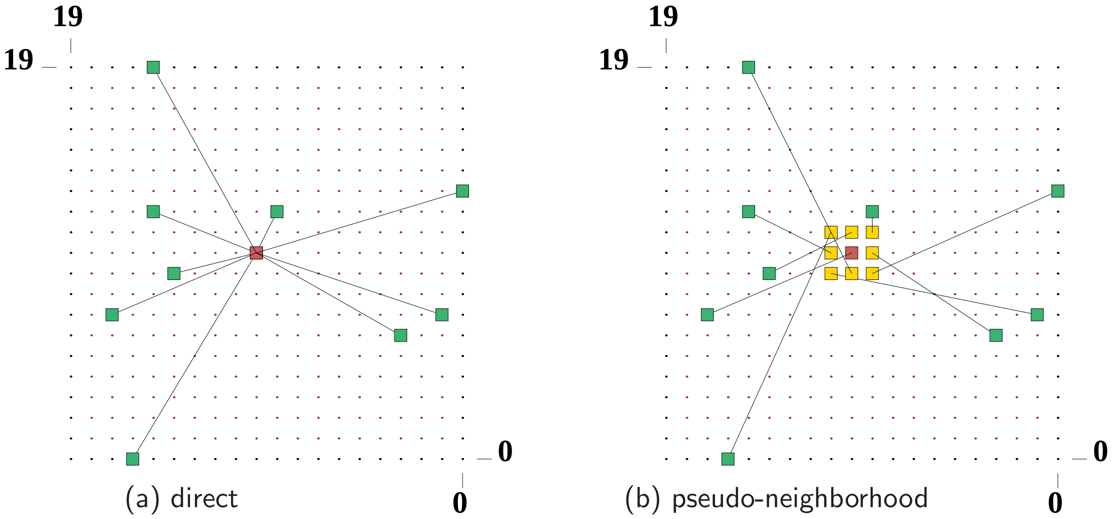


Figure 17.10: The 2d wiring graphic, n=20×20 RBN, $k = 9$, which can be toggled between (a) direct wiring – the default for random wiring, and (b) wiring to the pseudo-neighborhood – the default for CA.

17.6 Wiring graphic, 2d

Enter **2** in section 17.1, for the 2d graphic. This shows how a particular cell (the "active cell"), or a pre-defined 2d block of cells, is wired in the 2d network (or in a 1d network presented in 2d). The 2d graphic can be expanded and contracted, and shifted up and down if necessary to see the relevant part. The active cell is moved around the network with the arrow keys, and its possible to "jump" to a new location. The rule for the active cell appears in the rule window described in section 16.19.

For 1d networks shown in 2d, the most reasonable i, j dimensions are automatically computed, with $i \geq j$ (for n prime $i = n, j = 1$).

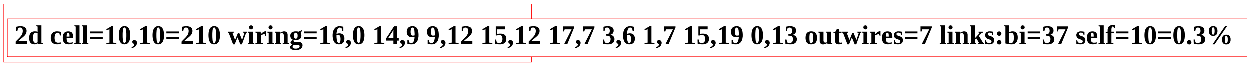
The display can be toggled between 'direct wiring" and wiring to the pseudo-neighborhood (section 17.6.4) as in figure 17.10. By default, "direct wiring" is displayed if the wiring is non-local, and the pseudo-neighborhood for local CA wiring. Enter **p** to toggle between the two. There is also a 5-way toggle (enter **n**) to alter the presentation of connections, and the cells connected, which is especially useful in visualizing blocks.

The current cell is colored red, its pseudo-neighborhood yellow, the "actual neighborhood" cells are colored green (see examples in figure 12.3).

Sections 17.6.1-17.6.8 below explain some of the options specifically for 2d in more detail that were summarised in section 17.4.1.

17.6.1 Data - 2d wiring graphic

As for the 1d wiring graphic, various data about the active cell's wiring are shown at the foot of the 2d wiring graphic. For example, for figure 17.10 the data is as follows,



```
2d cell=10,10=210 wiring=16,0 14,9 9,12 15,12 17,7 3,6 1,7 15,19 0,13 outwires=7 links:bi=37 self=10=0.3%
```

See section 17.8.1 to decode this data. The current cell, and cell input positions, are hown as I, J coordinates. The rule and rule data are shown in the rule window (section 16.19).

17.6.2 Moving or jumping between cells, 2d

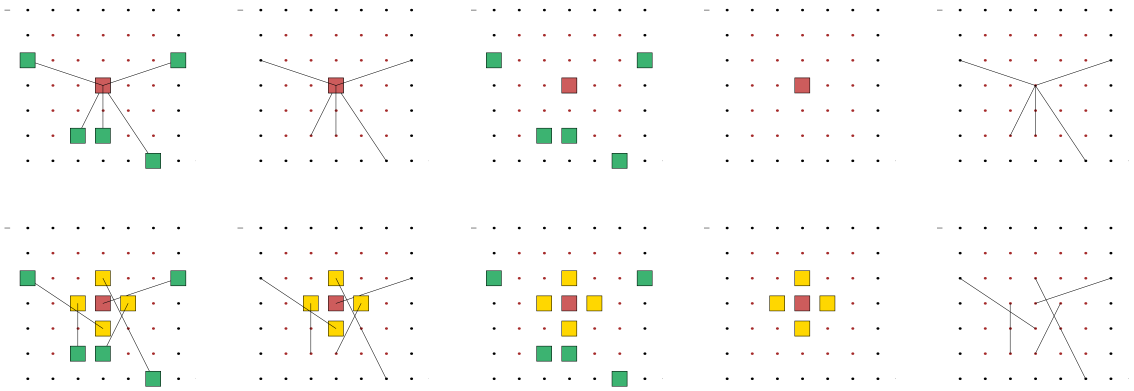
To move the active cell use the arrow keys. Alternatively, enter **j** to jump to a specific cell. The 2d cell position is specified by its I, J coordinates. The following prompts are presented,

> **jump to coord I,J**
> **enter I(19-0): enter J(19-0):** *(for a 2d network size* 20×20*)*

17.6.3 Alternative wiring presentation, 2d

A 5-way toggle allows alternative presentations of the the active cell, or an active block. Enter **n** to toggle between the five alternatives, as shown in figure 17.11. For the active cell, the 5-way toggle works both with direct wiring and with the pseudo-neighborhood (**pseudo-p**) in section 17.7.4.

For a 2d block, the alternatives are shown in in figure 17.12.



1. cell+links+inputs 2. cell+links 3. cell+inputs only 4. cell only 5. links only

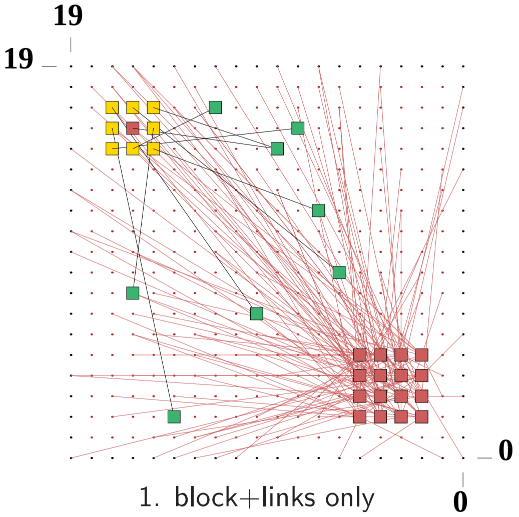
Figure 17.11: Toggling between 5 alternative ways of showing a cell and its connections in 2d with **n**. *Top row:* with direct wiring. *Bottom row:* with the pseudo-neighborhood set (toggle with **p**, section 17.6.4). n=7×7 RBN, k=5.

17.6.4 Include the pseudo-neighborhood, or direct wiring only, 2d

By default, "direct wiring" is displayed if the wiring is non-local, and the pseudo-neighborhood for local CA wiring. Enter **p** to toggle between the two, as between the top and bottom rows of figure 17.11.

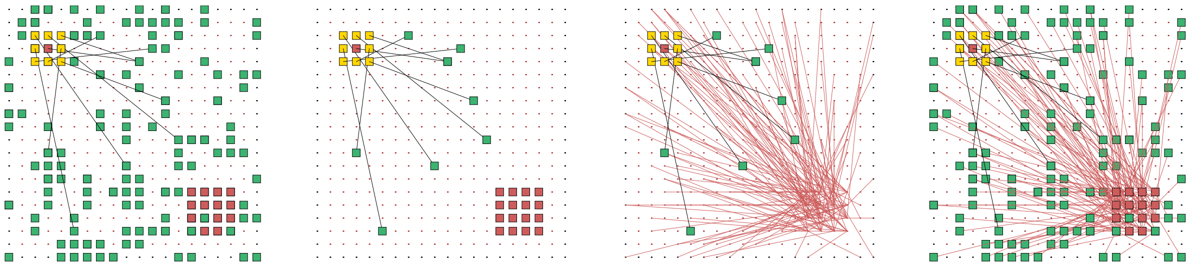
17.6.5 Defining a block, 2d

Enter **b** to define a 2d block of cells. The following top-right prompt is presented,



1. block+links only

Left: The 2d wiring graphic, just after a block was set – defined by the lower right (2,2) and upper left (5,5) coordinates. While visible, the presentation of the block is subject to the 5-way toggle (with **n**). The active cell behaves as usual, apart from not responding to **n**, but if the block is made invisible by toggling (with **g**), the 5-way toggle applies to the active cell (as in figure 17.11) instead of the block.



2. block+inputs only

3. block only

4. links only

5. block+input+links

Figure 17.12: Toggling between 5 alternative presentations of a 2d block (with **n**), starting with the initial block presentation (1. block+links only). n=20×20 RBN, k=9.

2d block-2, all-a single cell-def: *(then, if* **2** *is selected...)*
2d block (this=16,16 max 19,99) *(for a* 20 × 20*)*
low corner (def 0,16) I: J:
high corner (def 0,16) I: J:

Enter **2** to define a block, **a** to define the whole network as a block. If **2** was entered, enter the low and high cell coordinates defining the block. The wiring of all the cells in the block will be shown in the graphic as direct links colored light red, as in figure 17.12. The active cell will still be visible and can be moved, and its pseudo-neighborhood toggled with **p**, as usual. As long as a block is active and visible (toggle with **g**), the 5-way toggle **n** applies to the the block.

For a 2d network shown in 1d, the prompt is as shown in section 17.5.3

An active block is indicated in the wiring graphic reminder (section 17.4),

rewire 2d block 2,2-5,5: . . .

To deactivate the block enter **b** in section 17.4, then **return**. If the block is active and visible the following **rewire** options . . .

. . . **rnd-r/w/R special-s local:1d-1 2d-2**
change k=9(max 13)-k . . . *(for example)*

. . . will apply to just the block, not to the active cell if it is outside the block.

The following **rule** options apply to the active cell, but if the block is active and visible the new rule can then be copied to the block.

. . . **rule:Save/rev/trans-S/v/t**

17.6.6 Toggling the block, 2d

If a block is active (see 17.6.5 above), enter **g** to toggle the block, making it either visible or invisible, without deactivating it.

17.6.7 Expand/Contract the scale, 2d

The scale of the 2d wiring graphic is set automatically to fit within its window. Enter **e** to expand, **c** to contract this default scale. The minimum contraction is one screen pixel for each network cell.

17.6.8 Shifting the 2d graphic up and down

If only part of the graphic fits within the display area, it can be shifted up and down to see the relevant part. This may be necessary for larger 2d networks, especially if they have been expanded in section 17.6.7. Enter **u** or **d** to move the graphic up or down by 1/5 of its vertical dimension.

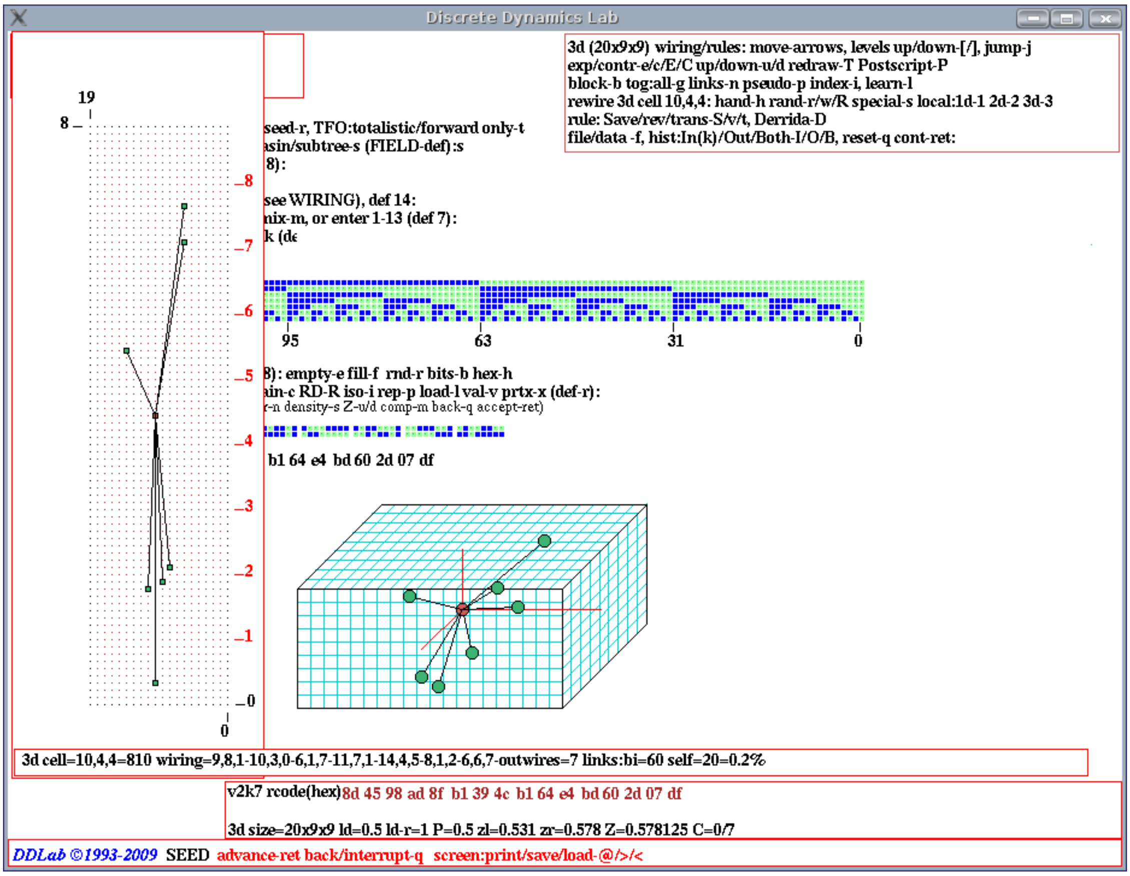
17.7 Wiring graphic, 3d

Enter **3** in section 17.1, for the 2d+3d graphic, where the network is shown simultaneously in 2d and 3d (see figure 17.13). As in 2d, this shows how a particular cell (the "active cell"), or a predefined 3d block of cells, is wired in the 3d network. The 2d and 3d versions of the 2d+3d graphic can be independently expanded and contracted.

The 2d version shows successive horizontal slices (levels), stacked above each other, with the levels indicated. For larger networks the 2d version can be shifted up and down to see the relevant part. Otherwise the 2d version behaves very much as the 2d graphic described in section 17.6.

The 3d version shows an isometric projection seen from below, as if looking down into a box. The active cell (for 2d+3d together) is moved around one level with the arrow keys, the square bracket keys [and] move up and down between levels, and its possible to "jump" to a new location. The rule for the active cell appears in the rule window described in section 16.19. As in the 2d wiring graphic (section 17.6), the display can be toggled between 'direct wiring" and wiring to the pseudo-neighborhood (as in figure 17.13). By default direct wiring is displayed if the wiring is non-local, and to the pseudo-neighborhood for local CA type wiring. There is also a 5-way toggle to alter the presentation of connections, and the cells connected, which is especially useful in visualizing blocks. The current cell is colored red, its pseudo-neighborhood yellow, the "actual neighborhood" cells are colored green (see examples in figure 12.4.

Sections 17.7.1 - 17.7.8 below explain some of the options specifically for 3d in more detail that were summarised in section 17.4.1.



```
3d (20x9x9) wiring/rules: move-arrows, levels up/down-[/], jump-j
exp/contr-e/c/E/C up/down-u/d redraw-T Postscript-P
block-b tog:all-g links-n pseudo-p index-i, learn-l
rewire 3d cell 10,4,4: hand-h rand-r/w/R special-s local:1d-1 2d-2 3d-3
rule: Save/rev/trans-S/v/t, Derrida-D
file/data -f, hist:In(k)/Out/Both-I/O/B, reset-q cont-ret:
```

```
seed-r, TFO:totalistic/forward only-t
asin/subtree-s (FIELD-def):s
8):

see WIRING), def 14:
nix-m, or enter 1-13 (def 7):
k (de

8): empty-e fill-f  rnd-r bits-b hex-h
ain-c RD-R iso-i rep-p load-l val-v prtx-x (def-r):
r-n density-s Z-u/d comp-m back-q accept-ret)

 b1 64 e4  bd 60 2d 07 df
```

```
3d cell=10,4,4=810 wiring=9,8,1-10,3,0-6,1,7-11,7,1-14,4,5-8,1,2-6,6,7-outwires=7 links:bi=60 self=20=0.2%
v2k7 rcode(hex)8d 45 98 ad 8f  b1 39 4c  b1 64 e4  bd 60 2d 07 df

3d size=20x9x9 ld=0.5 ld-r=1 P=0.5 zl=0.531 zr=0.578 Z=0.578125 C=0/7
DDLab ©1993-2009  SEED  advance-ret back/interrupt-q  screen:print/save/load-@/>/<
```

Figure 17.13: A 3d wiring graphic as it appears on the DDLab screen after the rules have been set. $[i, j, h]$=[20×9×9] RBN, k=7. *Left*: the network in 2d showing successive levels 0 to 9 stacked above each other. *Right*: the network in 3d, as if looking down into a box.

17.7.1 Data - 3d wiring graphic

As for the 1d and 2d wiring graphics, various data about the current cell's wiring are shown at the foot of the 3d wiring graphic as in figure 17.14. See section 17.8.1 to decode this data. The current cell position, and the actual neighborhood positions, are shown as i, j, h coordinates. The rule and rule data are shown in the rule window (section 16.19).

17.7.2 Moving or jumping between cells, 3d

To move the active cell use the arrow keys to move around one level. The square bracket keys ([,]) move up and down between levels, Alternatively, enter **j** to jump to a specific cell. The 3d cell position is specified according to the i, j, h coordinates. The following prompts are presented,

jump to index I,J,H *(this example for a 3d network size* $20 \times 9 \times 9$*)*
enter I(19-0): **enter J(8-0):** **enter H(8-0):**

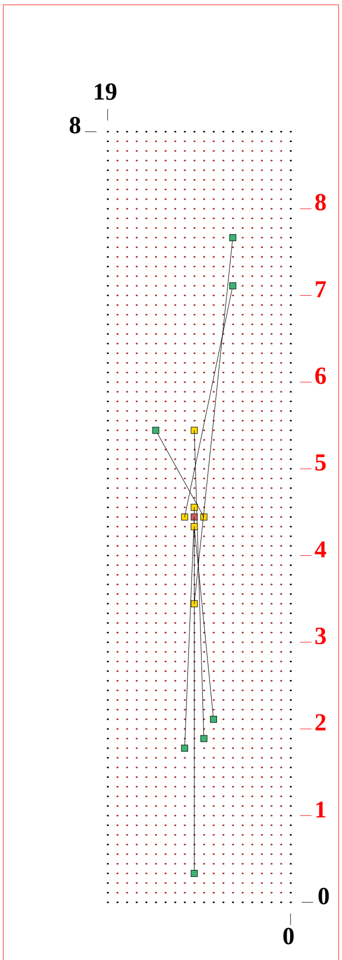
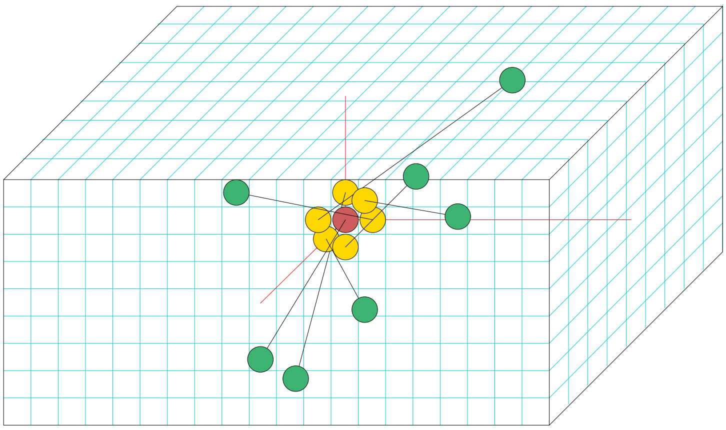


Figure 17.14: 3d wiring graphic: as figure 17.13 but toggled (with **p**) to show the pseudo-neighborhood. $[i, j, h]$=[20×9×9] RBN, k=7. *Left*: the network in 2d showing successive levels 0 to 9 stacked above each other. *Right*: the network in 3d, as if looking down into a box (expanded with **E**), where you see the floor, back wall and right side wall. *Bottom*: wiring data.



3d cell=10,4,4=810 wiring=9,8,1-10,3,0-6,1,7-11,7,1-14,4,5-8,1,2-6,6,7-outwires=7 links:bi=60 self=20=0.2%

17.7.3 Alternative wiring presentation, 3d

A 5-way toggle allows alternative presentations of the the active cell, or an active block. Enter **n** to toggle between the five alternatives, as shown in 3d in figure 17.15 – alternatives also appear in the 2d version of the 3d wiring graphic in a similay way as in figure 17.12. For the active cell, the 5-way toggle works both with direct wiring and with the pseudo-neighborhood (**pseudo-p**) in section 17.7.4.

For a 3d block, the alternatives are shown in in figure 17.16.

17.7.4 Include the pseudo-neighborhood, or direct wiring only, 3d

As in 2d, by default, "direct wiring" is displayed if the wiring is non-local, and the pseudo-neighborhood for local CA wiring. Enter **p** to toggle between the two, as between the top and bottom rows of figure 17.15.

17.7.5 Defining a block, 3d

Enter **b** to define a 3d block of cells. The following top-right prompt is presented,

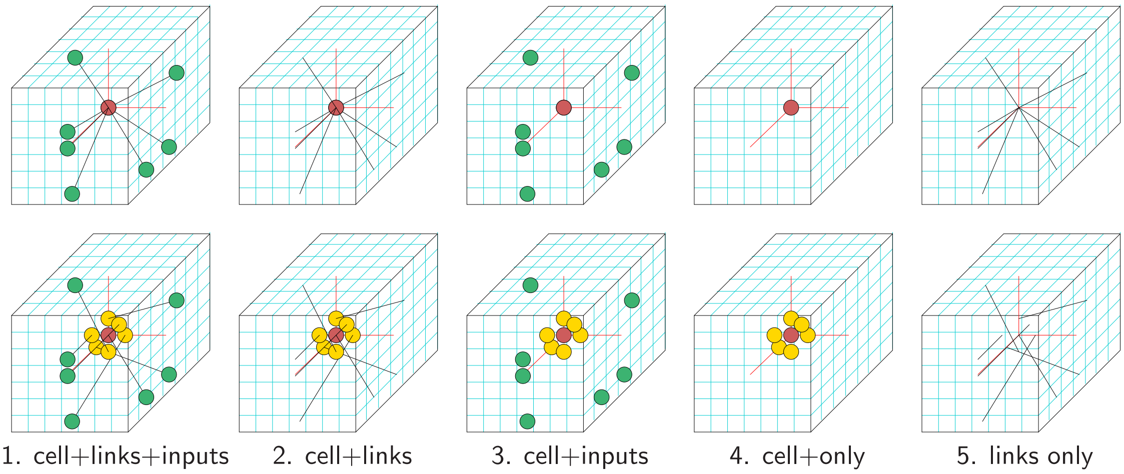


Figure 17.15: Toggling between 5 alternative ways of showing a cell and its connections in 3d with **n**. *Top row:* with direct wiring. *Bottom row:* with the pseudo-neighborhood set (toggle with **p**, section 17.7.4). n=7×7×7 RBN, k=7. The 2d version of the 3d wiring graphic (as in figure 17.14) presents the same information.

> **3d range-3, all-a single cell-def:** *(then, if* **3** *is selected...)*
> **3d block (this=7,7,7 max 14,14,14)** *(for a* 40 × 40*)*
> **low corner (def 0,0,7) I: J: H:**
> **high corner (def 0,24) I: J: H:**

Enter **3** to define the block, **a** to define the whole network as a block. If **3** was entered, enter the low and high cell indices defining the block. The wiring of all the cells in the block will be shown in the graphic as direct links, colored light red. The active cell will still be visible and can be moved as usual.

For a 3d network shown in 1d, the prompt is as shown in section 17.5.3

An active block is indicated in the wiring graphic reminder (section 17.4),

> **rewire 3d block 9,9,9-15,15,15:** . . .

To deactivate the block enter **b** in section 17.4, then **return**. If the block is active and visible the following **rewire** options . . .

> . . . **hand-h rnd–r/w/R special-s local:1d-1 2d-2 3d-3**
> **change k=7(max 13)-k** . . . *(for example)*

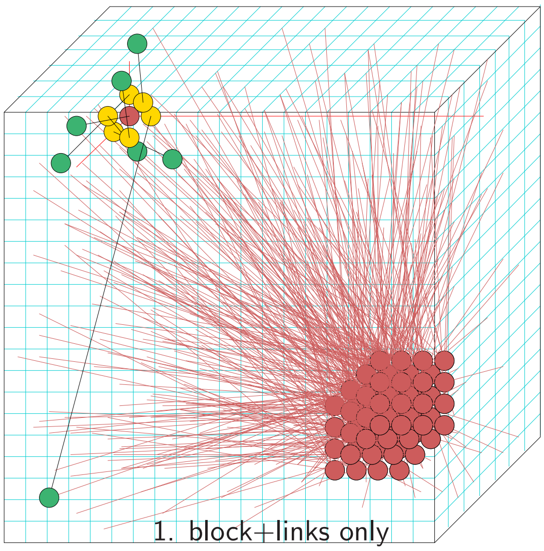
. . . will apply to just the block, not to the active cell if it is outside the block.

The following **rule** options apply to the active cell, but if the block is active and visible the new rule can then be copied to the block.

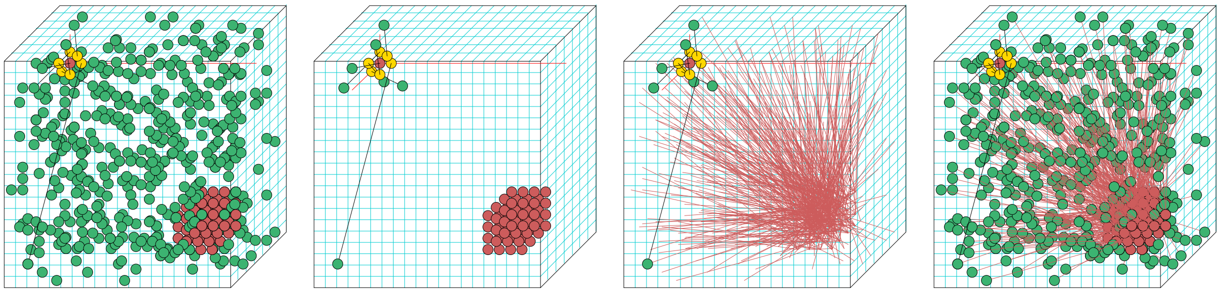
> . . . **rule:Save/rev/trans-S/v/t**

17.7.6 Toggling the block, 3d

If a block is active (see 17.7.5 above), enter **g** in section to toggle the block, making it either visible or invisible, without deactivating it.



Left: The 3d wiring graphic, just after a block was set – defined by the lower right (2,2,2) and upper left (5,5,5) coordinates. While visible, the presentation of the block is subject to the 5-way toggle (with **n**). The active cell behaves as usual, apart from not responding to **n**, but if the block is made invisible by toggling (with **g**), the 5-way toggle applies to the active cell (as in figure 17.11) instead of the block.



2. block+inputs only 3. block only 4. links only 5. block+input+links

Figure 17.16: Toggling between 5 alternative presentations of a 3d block (with **n**), starting with the initial block presentation (1. block+links only). n=20×7×20 RBN, k=7. The 2d version of the 3d wiring graphic (as in figure 17.14) presents the same information.

17.7.7 Expand/Contract the scale, 3d

The scale of the 2d and 3d versions of the 3d wiring graphic can be independently expanded and contracted. Enter **e** to expand, **c** to contract the 2d version. Enter **E** to expand, **C** to contract the 3d version. The minimum contraction in both cases is one screen pixel for each network cell.

17.7.8 Shifting the 3d graphic up and down

If only part of the 2d version of the 3d wiring graphic fits within the display area, it can be shifted up and down to see the relevant part. This is necessary for larger 3d networks, especially if they have been expanded in section 17.7.7.

Enter **u** or **d** to move the graphic up or down by about one level. Enter **s** to restore the default start position.

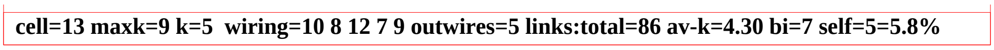
17.8 Further options for the 1d, 2d and 3d wiring graphics

The following options which were marked with a asterisk in section 17.4 apply jointly to 1d, 2d and 3d networks, and are described in the remainder of this chapter.sections 17.8.2 - 17.8.12 below,

17.8.1 Decoding wiring graphic data - 1d, 2d and 3d

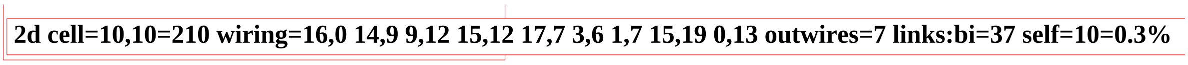
Various data about the active cell's wiring are shown at the foot of the 1d, 2d and 3d wiring graphic. For example,

for 1d



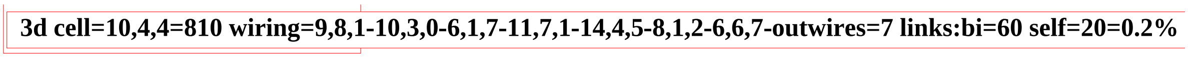
```
cell=13 maxk=9 k=5  wiring=10 8 12 7 9 outwires=5 links:total=86 av-k=4.30 bi=7 self=5=5.8%
```

for 2d



```
2d cell=10,10=210 wiring=16,0 14,9 9,12 15,12 17,7 3,6 1,7 15,19 0,13 outwires=7 links:bi=37 self=10=0.3%
```

for 3d



```
3d cell=10,4,4=810 wiring=9,8,1-10,3,0-6,1,7-11,7,1-14,4,5-8,1,2-6,6,7-outwires=7 links:bi=60 self=20=0.2%
```

decode of wiring data

cell= . . . the cell coordinates (1d, 2d or 3d) and the 1d cell index.

maxk . . . for mixed k networks, Shows the max-k setting.

k . . . for mixed k networks, shows k for the active cell.

wiring . . . network indexes wired to the pseudo-neighborhood, ordered $k-1\ldots0$, in 1d, 2d or 3d.

outwires . . . the number of output wires, how many wires from the network are "plugged into" to the cell in question.

self . . . the number of self-links, or inputs that are outputs from the same cell.

bi-links . . . The number of cell pairs that have both inputs and outputs to each other.

The active cell's rule with other rule details (updated as the active cell is changed) are displayed in the lower rule window at the foot the DDLab screen (section 16.19).

17.8.2 Computing the (weighted) average λ and Z parameters

rcode-mix and/or k-mix

This option applies once rcode-mix has been set in chapter 14 – rcode-mix is set automatically for a k-mix.

Enter **z** in section 17.4 to calculate and display the average, and weighted average, λ_r and Z parameters of the rules making up the rule-mix.

The data is displayed in a top-right window, for example, for a 1d network $n=150$, $k=3-7$, and randomly assigned rules,

```
av:ld-r=0.839063 Z=0.63653
wt.av:ld-r=0.826438 Z=0.626692 cont-ret:
```

For random wiring and/or mixed k networks, the weighted averages take account of the influence that each cell has on the network according to its proportion of the network's out-wires, represented graphically by the height of cells at t_0 in a 1d wiring graphic, for example in figure 17.4.1. In networks with regular wiring and no rule mix the weighted average equals the average.

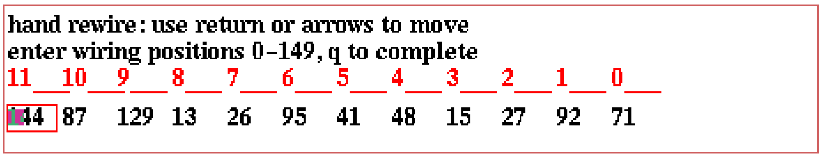
17.8.3 Learning pre-images without attractor basins

Requires random wiring or rcode-mix (set in section 14), preferably both

Enter **l** in section 17.4 to start the "learning/forgetting" routines for attaching/detaching sets of states as pre-images of a target state. The network's wiring and/or rule scheme is automatically amended to achieve the required transitions between states. The learning routine is described in detail in chapter 34, where learning is also invoked in the context of "sculpting attractor basins". In this case, the results and side affect of learning are displayed when attractor basins are generated, but this places a limit on the size of the network (see section 8.3). Invoking the routine at this point in the program, or when running networks "forwards only", allows much larger networks to "learn".

Note that the full scope of the learning routine requires a network with random wiring and mixed rules. If the network has regular 1d wiring, learning by rewiring is not applicable. If the network is set up with a single rule, learning by flipping bits in rule-tables is not applicable. Note that regular 1d wiring can be treated as random (see section 12.4.1), and a rule mix where all the rule are the same can be set up as described in section 14.4.3.

17.8.4 Hand rewiring



```
hand rewire: use return or arrows to move
enter wiring positions 0-149, q to complete
11__10__9___8___7___6___5___4___3___2___1___0___
144 87  129 13  26  95  41  48  15  27  92  71
```

Figure 17.17: Hand wiring a single cell from the 1d wiring graphic, n=150, k=13

Enter **h** in section 17.4 to rewire individual wires for just the active cell by hand, which is rewired in a way similar to that described in sections 12.6 and 17.2.2, but for the active cell only. A top-right window displays the pseudo-neighborhood wiring connections as a 1d "spread sheet" with k entries (figure 17.17), including the following reminder, for example,

hand rewire: use return or arrows to move
enter wiring position 0-149, q to complete *(this example for a 1d network, size 150)*

Note that the wiring position x is a 1d index, even if the network is 2d or 3d. To convert between the index and coordinates see sections 10.6.2 and 10.6.3.

17.8.5 Random rewiring

Enter **r** in section 17.4 to randomly reset the wiring of the active cell, or of the block if defined and visible.

w is similar to **r**, but randomly resets the wiring of the cells *outside* an active block, if the active block is visible.

Both **r** and **w** will respect the biases on random wiring (which include regular CA wiring) set in section 17.8.6 (see also section 12.5). Note that for non-square 2d networks and non-cubic 3d networks, the random wiring is limited to a local zone equal to the shortest axis. In this case unbiased random wiring (for homogeneous k) can be set with **R**.

For a k-mix network, if **R** is entered, the wiring will be randomly reset, but biased by the k distribution. That is, the probability of plugging wires into a cell with k inputs will be given by $P = k/T$, where T is the total number of links in the network. so that cells with most inputs also end up with the most outputs and vice-versa. If a power-law distribution of the k-mix was set for the network in section 9.8.2, entering **R** will rewire to give an approximate power-law distribution of outputs (figure 17.18), creating a "scale free" network , said to be characteristic of many natural and artificial networks, from metabolic networks to the world-wide-web[2]. The graph of such a network is shown in figure 20.2.

For homogeneous k, **R** gives totally unbiased random wiring.

17.8.6 Biased random rewiring

If **s** (for special) is selected in section 17.4 a series of options are presented in a top-right window that allow a variety of restrictions or biases to the random wiring before it is set in section 17.8.5 above. These biases apply to the active cell, or to the block if defined and visible.

The special wiring options are fully described in section 12.5 (*Special wiring, random*). They differ to some degree between the 1d, 2d or 3d wiring graphics.

17.8.7 Regular 1d, 2d or 3d wiring

If **1**, **2** or **3** is selected in section 17.4 the wiring of the active cell, or block if defined and visible, will be reset to regular 1d, 2d or 3d CA-like wiring to the local neighbourhood, with periodic boundary conditions.

For 3d networks, 1d, 2d or 3d regular CA wiring applies. For 2d networks, 1d, 2d regular wiring CA applies. For 1d networks, only 1d regular CA wiring applies.

Note that CA wiring can also be set as "random wiring" in section 17.8.6 above, allowing various biases to the CA wiring as described in section 12.5.

17.8.8 Changing the neighborhood size, k

rcode-mix set only – not in TFO mode

Whether or not the network has a k-mix, **k** may be selected in section 17.4 to change the neighborhood size k of the active cell, or the block if active and visible. The following additional prompt appears,

> **reset nhood size k (1-13):** *(if max-k=13)*

The maximum k allowed is *max-k*, which may be greater than the actual maximum k in the network (see section 9.11), with an upper limit of k=13 for binary rules. The new wiring will preserve as much as possible of the old neighborhood wiring. If k is increased, the wiring at the extra pseudo-neighborhood indexes is set as regular 1d, 2d or 3d (depending on the network dimensions) with periodic boundary conditions.

For a homogeneous k network, an individual cell may only have its k setting reduced. The network will from then on be treated as a k-mix with *max-k* equal to the original value of k.

Changing k effects the highest pseudo-neighborhood indexes, which are either removed or added. The original rcode (or part of it if k was reduced) are preserved. Any excess rule-table entries are set to 0.

Note that k may also be increased from the transformation options – sections 17.8.11 and 18.7, giving a neutral transformation with equivalent dynamics.

17.8.9 Kill a cell

rcode-mix set only – not in TFO mode

Enter **K** in section 17.4 to kill or neutralize the active cell, or the block if active and visible. by cutting all links, both inputs and outputs. The cell retains one input to itself, k=1, and the rcode is set so the cell's value stays constant, with effective k=0 (see section 9.3). For v=2 the decimal rcode is 2. The cell then has has no influence on the network, which can be applied to model gene knockout. Alternatively, the cell can serve as a constant unchanging input to other cells if they rewire into it.

For 1d networks, a cell can also be deleted entirely (see section 17.5.9).

17.8.10 Revising and copying the rule

if rules are set

Enter **v** in section 17.4 to revise the rule of the active cell. Once the rule is revised, it can be copied to a block. This option is intended for networks with mixed rules. For a single rule network (set in section 14.1), i.e. without a rule-mix, any revision applies to the whole network. Note that a rule-mix where all the rule are the same can be set up as described in section 14.4.3.

A secondary window is presented in the lower right hand corner of the screen, with rule selection prompts. The various methods for revising and re-selecting rules are the same as in *Setting a singe rule* described in chapter 16.

If a block is active and visible, the rule can be automatically copied to all cells in the block which have the same k as the cell in question. The following prompt is presented in a top-right window, showing the block coordinates and their 1d equivalent indexes in brackets (section 17.2),

> *for a 1d wiring graphic (values shown are examples)*
> **Copy rule to 33-55 -c:**
> *for a 2d wiring graphic,* 40 × 40
> **Copy rule to 2d range 0,22-39,33 (880-1359)-c:**
> *for a 3d wiring graphic,* 20 × 20 × 20
> **Copy rule to 2d range 0,0,8-19,19,12 (3200-5159)-c:**

Enter **c** to copy the rule within the block, to all cells with matching k. Once set (and copied), the new rule is displayed in the rule window (see section 16.19).

To change a rule to one with a different neighborhood size k, first change k in the rewiring window (see section 17.8.8), then change the rule.

17.8.11 Transforming the rule

(rcode set only – more options rcode-mix – more options k-mix – if not in TFO mode)

If **t** is selected in section 17.4 the rcode for just the active cell may be transformed in the various ways described in detail *Transforming network rules*, chapter 18. This option is intended for networks with rcode-mix or k-mix. For networks with a single rcode network any revision applies to the whole network.

A top-right window is presented with the transformation prompts. The rule may be transformed to equivalent or related rules. For example the rule may be complimented, or transformed to an equivalent rule by negative or reflection transformations. Canalizing inputs may be set or amended for the active cell or the whole network as described in section 15. A k-mix allows more options – neutral tranformations to rules with greater k, or k reduced to "effective k" for the particular cell or for the whole network.

The network may be "reverse engineered" by loading an exhaustive mapping of transitions (see section 18.10) and automatically generating the minimal mixed-k network that satisfies the mapping, (i.e. reduced to effective-k), one solution to the "inverse problem".

17.8.12 Filing, from the wiring graphic

Enter **f** in section 17.4 for the network filing options described in chapter 19, *Save/load/print network architecture*. To load a network file, first set up a similar "dummy" network. The whole network or just the wiring or rule-mix can be saved or loaded to a file. The options vary according to a k-mis, rules set, and other constraints.

If a network file smaller than the base network is loaded, it will be automatically be defined as a block in 1d, 2d or 3d. A prompt first locates the block, for example in a 2d network,

```
2d:i,j=40,40, file: i,j=11,11, enter start coords (def 14,14, max 29,29, rnd-r)
I:   J:
```

Enter the lower coordinates of the block, **r** for a random position, or **return** for a central position. Once loaded, the block is shown in the wiring graphic of the base network as in figures 17.4.1, 17.12 and 17.16. Any number of sub-networks can be loaded into the base network.

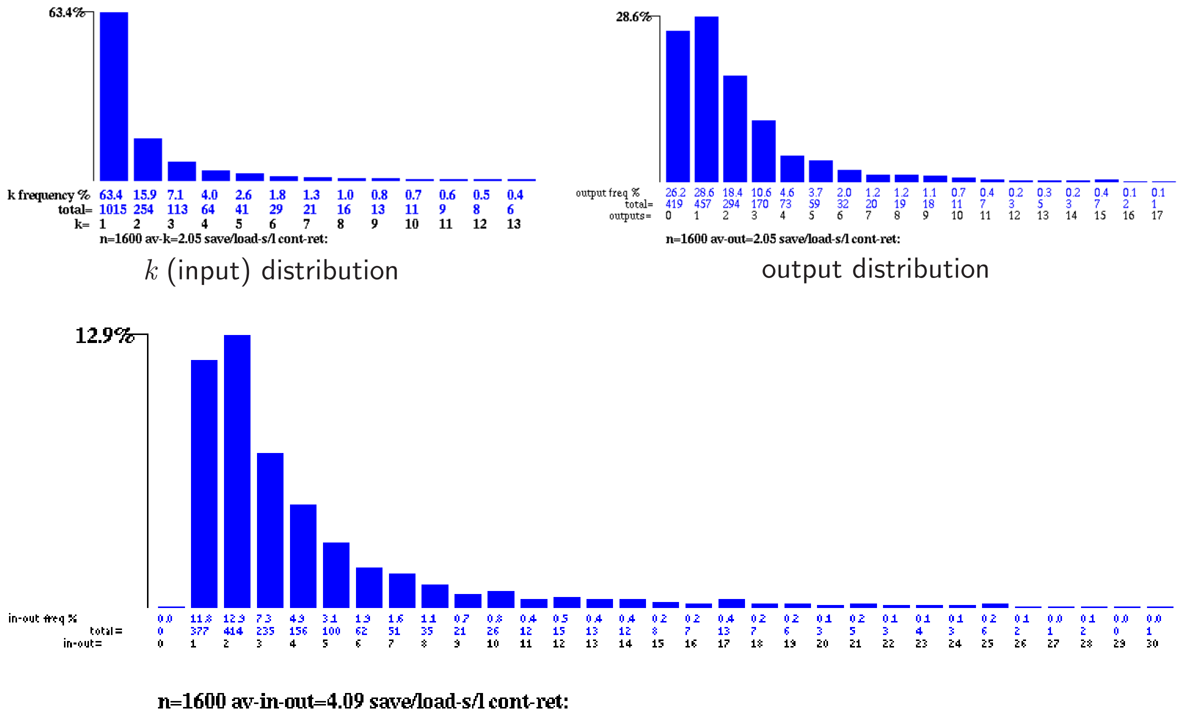
A network description can also be printed to a file, to the xterm window for Linux-like syatems, or to a printer for DOS. For further details see chapter 19, *Save/load/print network architecture.*

17.8.13 The histogram of the network's k and output distribution

Enter **I**, **O** or **B** in section 17.4 to show a histogram of link frequency distribution. **I** gives the k distribution – of inputs, **O** gives the output distribution, and **B** gives both the input and output, i.e of all connections in the network. The k distribution can also be displayed from section 9.12. Examples of power-law distributions are shown in figures 17.18. Random wiring without bias would give a Poisson distribution as in figure 9.1.

The histogram plots each k or output (x axis), against its frequency in the network as a percentage (y axis). The actual frequency, as a percentage and total, are shown under each histogram bar. The following information and prompt is also shown,

n=100 av-k=2.02 save/load-s/l cont-ret: *(for example)*



k (input) distribution

output distribution

both k and output distribution

Figure 17.18: Histograms of a power-law distribution of network links, (k) inputs, outputs, and both combined. The power-law exponent was set as 2.0 in section 9.8.2 for a mixed=k network 40×40. Random wiring was reset for the whole network with **R** (section 17.8.5), where outputs are preferentially allocated according to k.

n is the network size which should equal the sum of all the frequency totals. **av-k**, **av-out** or **av-in-out** is the average k, out-degree, or all links to a node.

Enter **s** or **l** save or load the histogram data as a `.his` file (see Filing, chapter 35). In both cases the data appear in the xterm window (*not for DOS*), for example for figure 17.18($left$),

```
histogram:  1+13 columns
0 1015 254 113 64 41 29 21 16 13 11 9 8 6
```

17.8.14 Creating a vector PostScript file of the wiring graphic

Enter **P** in section 17.4 to save the wiring graphic image (1d , 2d or 3d) as a vector postscript file, and will follow the presentation exactly as on the screen, including text data. The following top-right prompt appears,

save wiring graphic to PostScript: greyscale-P color-p:

Select greyscale or color, then the filename – default `my_wgPS.ps`. The complete wiring graphic will be redrawn as the file is being created, for example, figures 17.4.1, 17.10, and 17.14 in this chapter. Note that a vector postscript image can be cropped by method described in section 36.1.

Chapter 18

Transforming network rules

This chapter apples to rcode – not TFO-mode.

If not in TFO-mode, rule transformation options are automatically presented after a single rcode, kcode or tcode has been set (section 16.21), and can also be selected from the following options,

- the wiring graphic, section 17.8.11
- space-time pattern interupt options, section 32.14.2.
- attractor basin complete options, section 30.5

For totalistic rules the equivalent rcode expression of the kcode or tcode will be transformed. If there is not a rule-mix, transformations apply to the whole network. For a rule-mix from the wiring graphic, transformations apply to the active cell, otherwise the cell index is first selected with the following prompt,

enter cell index, 149-0: *for a 1d network*[1]*, n=150*

The rcode may be saved at any stage as a `*.rul` file (chapter 19), and the transformed rcode is displayed in the lower rule window. The transformations options include,

- various manipulations of the rcode table, including negative and reflection transformations within equivalence classes and rule clusters[19].
- inverting the rcode between two convensions for listing neighborhoods.
- setting or amending canalizing inputs (v=2 only).
- equivalent rcode with greater k, or with smaller "effective-k".
- creating a network to satisfy an exhaustive mapping of transitions (v=2 only).

[1] The cell positions shown, x, are 1d indexes, even if the network is 2d or 3d. As noted in section 10.6.2, for a 2d network i,j, to convert I,J coordinates to a 1d index, $x = iJ + I$. Converely, $I = x \mod I$, $J = \left|\frac{x}{I}\right|$. As noted in section 10.6.3, for a 3d network i,j,h, to convert I,J,H coordinates to a 1d index, $x = ijH + iJ + I$. Converely, $I = x \mod i$, $J = \left|\frac{x \mod ij}{i}\right|$, $H = \left|\frac{x}{ij}\right|$.

18.1 Options for transforming rules

Examples of the transformation options are show below. They vary according to the context[2] depending on factors such as the rule-mix, k-mix, and k_{max}. Any number of transformations may be made cumulatively.

18.1.1 Transform options, single rule

For a single rule network, transformations apply to all cells in the network.

> **transform rcode: solid-o invert-v comp-m neg-n ref-r canal-C**
> **equiv>k(3):(4-13), eff k***(if k=3)*
> **save/prtx: rcode-s/x**
> *or if totalistic, kcode (or tcode)*
> **save/prtx: rcode/kcode-s/S/x/X**

18.1.2 Transform options, mixed k

For a mixed-k network the transformation options are invoked from the wiring graphic in section 17.8.11. Then first two lines of the prompt are ...

> **transform rule: solid-o invert-v comp-m neg-n ref-r canal-C (all+a)**
> **equiv>k(3):(4-7), max k-M, eff k: all-K this-k** *(if k=3 and k_{max}=7)*

Note if k=k_{max} the prompt starts with **max k-M,** ...

For v=2 only, and $k_{max} = n$ and $n \leq 13$ and there is an additional option to load an exhaustive map (`.exh` file, see alsp section 29.8.1), which would have been saved previously in section 29.7.2, so the second line of the prompt would start with ...

> **exh map-e,** ...

18.1.3 Transform options, mixed rule, homogenious k

For a mixed rule network with homogenious k network, the "greater k" and "effective k" transformations do not apply.

> **transform rule: solid-o invert-v comp-m neg-n ref-r canal-C (all+a)**
> **save-s:**

Note that a mixed-k network may be set up where all rules and k's are the same, allowing the full scope of the transformations.

[2] Transformations which change the size of k are omitted while interrupting space-time patterns, and the transformation prompt is deactivated altogether while interrupting incomplete attractor basins.

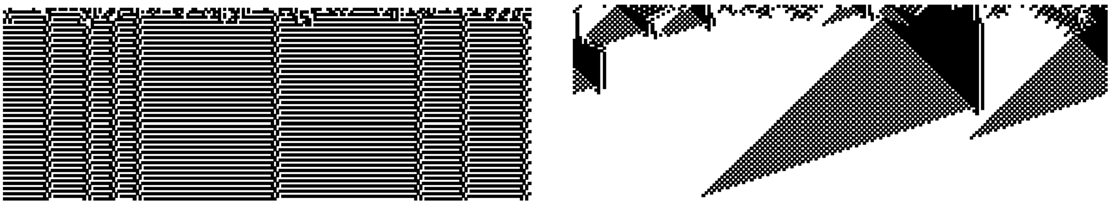
18.1.4 Transform all cells in a mixed rule network

To transform all cells in a mixed rule network, always the case for mixed-k, with options **o**, **v**, **m**, **n**, **r** or **C**, enter the option followed by **a**. For example to compliment all rules enter **ca**. Similarly, to revise canalizing inputs for the whole network (see chapter 15) enter **Ca**.

18.2 Solidifying the rule

If **o** (or **oa**) is entered at the prompt in section 18.1, the rcode/s will be changed to achieve the following behaviour. For binary (v=2) “on” (i.e. “1”) cells will remain on. For $v \geq 3$ a cell with the highest value will remain on.

18.3 Inverting the rcode



05040587 05000f77 03775583 7bffb77f

feedffde c1aaeec0 eef000a0 e1a020a0

Figure 18.1: Transforming rcode: *left*: the space-time patterns of a 1d “density” rule from[11] (section appear not as intended because the order convention is the opposite of DDLab’s. *right*: after the rule is inverted to DDLab’s convention the majority rule behaves correctly. n=199, $[v, k]$=[2,7]. The same random initial state is used in both cases. The rule-tables are shown in hexadecimal.

The rcode-table (or any rule-table) can be expressed according to two alternative conventions, where the output of the all-0s neighborhood is ether on the left or right. In DDLab the all-0s output is on the right (the least significant bit/value) and the all-max-value output is on the left (section 13.3 and 13.5), so that a rule-table string can be expressed as a decimal number following Wolfram convention[17] (if within the limits in section 16.6). Other conventions (e.g. [11]) follow the opposite order.

Enter **v** (or **va**) at the prompt in section 18.1 to invert the rcode/s between the two convensions. Effectively, the outputs of complimentary neighborhood pairs in the rule-table are swapped. For example (for k=5) the outputs of 00000 and 11111, or 00111 and 11000, etc. The example in figure 18.1 shows a 1d $[v, k]$=[2,7] “density” rule[11] inverted to the convention in DDLab.

18.4 Transformations for equivalence classes and rule clusters

As described in [19], starting with a given rcode, equivalent rcodes can be created by 3 transformations, negative, reflection, and negative/reflection (where the order is equivalent), described in sections 18.4.2 and 18.4.3 below.

For binary (v=2) rules the transformations making potentially 4 equivalent rcodes – there may be fewer because a transformation may lead to identity. By these transformations the rule-space of 256 k=3 binary rcodes collapses into 88 equivalence classes, so only 88 prototype rules are required to fully describe the rule-space behaviour. The rcodes are equivalent in the sense that a transformed initial state will make equivalent transformed space-time patterns as in figure ?, and attractor basins states are equivalent in the same way – their topology is identical.

Complimentary transformations (section 18.4.1) will further group the rcodes into 48 rule clusters[19] which share a number or properties, including in-degree, g-density and the Z-parameter.

For $v \geq 3$ there are potentially more equivalent rcodes in an equivalence class, because negative transformations can be partial.

18.4.1 Complimentary transformation

If **m** (or **ma**) is entered at the prompt in section 18.1, the rcode/s will transformed to its "compliment". For binary (v=2) this is done by flipping all entries in the rcode-table, 1 to 0 and 0 to 1. For $v \geq 2$ each entry a is changed to its compliment $a_c = (v-1) - a$. See section 16.20 for a further explanation. A complimentary rcode is usualy not equivalent to the original.

18.4.2 Equivalent rcode by the Negative transformation

If **n** (or **na**) is entered at the prompt in section 18.1, the rule will be transformed into its "negative" equivalent rule, by swapping the outputs of complimentary neighborhood pairs in the rule-table, then transforming the resultant rule-table into its compliment. A negative rule and the start rule have equivalent dynamics, but with all state configurations in the space-time pattern complimented.

Note that a negative transformation followed by a reflection transformation (see section 18.4.3), or vice-versa (the order is equivalent) produces another equivalent rule, making 4 in all. For certain rules the transformations result in identity.

18.4.3 Equivalent rcode by the Reflection transformation

If **r** (or **ra**) is entered at the prompt in section 18.1, the rule (or rules) will be transformed into its equivalent rule by "reflection" (by swapping the outputs of pairs of asymmetric reflected neighborhoods in the rule-table). A reflected rule and the start rule have equivalent, but reflected dynamics.

Note that a reflection transformation followed by a negative transformation in section 18.4.2, (or vice-versa – the order is equivalent) produces another equivalent rule, making 4 in all[19]. For certain rules the transformations result in identity.

18.5 Setting canalizing inputs, single rcode

If **Ca** is entered at the prompt in section 18.1 for a mixed rcode network, canalizing inputs can be randomly biased for the whole network as described in chapter 15

If just **C** is entered, canalising inputs may be amended to for a single rcode, or for the active cell in a mixed rcode network. A given number of the rcode's imputs may be set to be canalizing, either explicitly or at random. The following top-right prompt is presented.

canalizing inputs: explicitly-e random-n:

18.5.1 Canalizing inputs at random, single rcode

Enter **n** to set a given number of inputs as canalizing at random (which can be reduced as well as increased). The following prompt appears, enter the required number,

canalizing inputs=0, set new number (0-5): *(values shown are examples)*

18.5.2 Canalizing inputs explicitly, single rcode

Enter **e** to set the canalizing inputs explicitly, the following prompts appear in sequence,

wire (0-5): **input value (1,0):** **output value(1,0):** *(values shown are examples)*

Enter the neighborhood index (wire) to be set as canalizing, and the input and output canalizing values (0 or 1).

18.6 Transform all cells in the network

For a single rule network, transformations described in sections 18.2 to 18.5 apply to all cells, whereas for a mixed rule or mixed-k network, just the selected cell in the wiring graphic (see section 17.8.11) can be transformed, or all cells in the network. To transform all cells, add **a** to the selection, i.e. enter **oa** in section 18.2, **va** in section 18.3, **ca** in section 18.4.1, **na** in section 18.4.2, **ra** in section 18.4.3 and **Ca** in section18.5. Entering **Ca** allows canalizing inputs for the whole network to be reset as described in chapter 15.

18.7 Equivalent rules with greater k

A rule with a given neighborhood size k has equivalent rules with any greater value of k. If $k < k_{Lim}$ (section 7.2), or $k < k_{max}$ for a mixed k network, then the options in section 18.1 includes a prompt such as **equiv>k (4-13),**... *(for example)*

Enter a greater value of k, which will produce a new rcode – a neutral transformation for 1d networks[3]. Increasing k, for example from k=3 to k=5 or k=6, results in rcode giving identical dynamics to that of the original, but mutations of the larger rcode are finer grained, so the transformation is useful for looking at close mutations of a given rcode, as in figures 28.3 and 28.4. Filtering space-time patterns with very complicated background domains may also require neutrally increaseing k, for example the k=3 rcodes 110 and and 54 to k=5 (see section 32.10.5 and figures 32.10-32.12).

For networks with a homogeneous rule the transformation applies to the whole network. In mixed k networks, only the chosen cell in the network is transformed. For mixed rule, homogenious k networks, the option does not apply.

An extra wire added to an even-k pseudo-neighborhood is added on the left, at the new k-1 index. An extra wire added to odd-k is added at index 0 (on the right), the original connections are retained and their indexes increaced by one. Thus the original connections remain in the enlarged pseudo-neighborhood. If more than one wire is added these steps are repeated. Rcodes

[3] The algorithm is designed for 1d networks; for 2d and 3d the transformation is not neutral.

are transformed so that the added wires are effectively redundant and play no role in the dynamics[4]. However, if the rcode is mutated the extra wires would come into play. Extra wires are connected within the network as they would be in a 1d cellular automata.

18.8 Reducing k_{max} to the maximum k in the network

For mixed k networks, k_{max} may be larger than the maximum k in the network. This may have been deliberatly set in section 9.11, or a network may have been loaded with a smaller k_{max} than the base network (see section 19.4.4). Enter **m** to reduce k_{max} to the maximum k found in the network.

18.9 Effective k

The pattern of 0s and 1s in a rule-table may be such that some wires are redundant, playing no role in the dynamics. An example is a rule that has been neutrally transformed with greater k, in section 18.7 above. If **k** or **K** is entered at the prompt in section 18 and redundant wires exist, the rule will be neutrally transformed to a rule with decreased k (to a minimum of k=1) by pruning redundant connections and transforming the reduced rule-table appropriately.

For a single rule network, the effective k transformation applies to the whole network. For mixed k networks, if lower-case **k** is entered only the chosen cell in the network is transformed, whereas if upper-case **K** is entered all cells will have their rules reduced to each effective k (to a minimum of k=1). To make the effective k=0, see section 14.3.

If a mixed k network was set up where k_{max}=n, an option is presented to load an exhaustive mapping of transitions (see Filing, chapter 35) and "reverse engineer" the network. An algorithm automatically transforms the network to satisfy the mapping. The effective-k transformation may then be applied to reduce the network back to its minimal k-mix. A further option allows k_{max}, which may be greater than the maximum k found in the network, k_a (see section 9.11) to be reduced to k_a.

18.10 Reverse engineering - loading an exhaustive map

For a network were $k_{max} = n$ and $n \leq 13$, an exhaustive list of transitions, a random map, that was previously created (see section 29.8) may be loaded to "reverse engeener" the network by transforming it to satisfy the mapping. This is a rudimentary solution to the inverse problem. At present it is restricted to the case where all transitions are specifed, and where $k_{max} = n$, so network sizes $n \leq 13$ because for rcode k_{max}=13 in DDlab.

The resulting network's dynamics, and attractor basins, will correspond to the mapping. The method works in three stages, firstly the rules and wiring are transformed for a fully wired network where $k = n$ for all cells, then the effective k option (see section 18.9) is implimented, followed by the option, if required, to reduce k_{max} to the actual maximum k found in the network (see section 18.8).

[4] Added wires can be neutrally reduced to effective k in section 18.9 to revert to the original wiring and rcode

1. Enter **e** at the prompt in section 18 to load the mapping file and transform the network to a fully wired network. If you wish, you may see the fully wired transformed network by pressing "return" and reverting to "Reviewing network architecture, wiring and rules" (section 17).

2. Enter **K** to reduce the wiring to effective k (see section 18.9). There will be a pause while this is being computed.

3. Enter **m** to reduce k_{max} to the maximum k found in the network (see section 18.8).

The result may be seen by reverting to "Reviewing network architecture, wiring and rules" (section 17), and by generating attractor basins (section 30) to check that they are the same as the basins for the network that was used to generate the exaustive map.

18.11 Saving or printing the transformed rule

The transformed rule is displayed in the rule window. Enter **s** at prompt in section 18.1.1 to save the rule (see Filing, chapter 35), or **x** to print the rule in the terminal (xterm) for Linux-like systems. If a totalistic rule was selected, **S** will save, and **X** will print, the totalistic version of the rule.

Intentionally Blank

Chapter 19

Save/load/print network architecture

While the wiring graphic (section 17.4) is visible, the filing option **f** is available to save, load or print network architecture, described in this chapter. The k-mix, rule scheme and wiring scheme may be treated in isolation, or combined. Various compatibility issues apply for loading files into a "base" network depending on the base properties. Broadly, a file network must fit into the base network and $k_{max,file} \leq k_{max,base}$ – if so, networks with different dimensions can be loaded into each other. When successfully loaded, a file network is automatically defined as a block (sections 17.5.3, 17.6.5 or 17.7.5), with its wiring visible in the wiring graphic, where the block can be seperatly manipulated. Any number of sub-networks can be loaded into the base network.

The ascii data can also be saved, and printed in the terminal (xterm) for Linux-like systems.

19.1 Network filing options

While the wiring graphic is visible, Hit **f** (without **return**) in the "wiring graphic reminder" (section 17.4) for the top-right network filing prompt. The prompt differs between homogenious-k (k-**hom**) and mixed-k (k-**mix**), and also depends on the context, where **n/a** signifies non-applicable – two exampes are shown below,

for homogenious-k and mixed-rule networks
k-hom, rulemix, save/load/print:
wiring-only: print-pw, load-lw, save-sw
rulemix-only: print-pr, load-lr, save-sr
rulemix+wiring: print-pb, load-lb, save-sb:

for mixed-k networks – mixed-rules by default
k-mix, save/load/print: save kmix-k *(just k-mix can be saved here, but not loaded*[1]*)*
wiring-only: print-pw, save-sw
rulemix-only: n/a *(rulemix+wiring applies instead)*
rulemix+wiring: print-pb, load-lb, save-sb:

[1] The k-mix information is held as part of the wiring scheme and so is loaded with **rulemix+wiring**. Just the k-mix may be saved by entering **k**, but the k-mix *in isoloation* may only be loaded earlier in the program sequence (see sectin 9.5)

The following additional variations apply to the prompt,

- If rules were not yet set, **no rules** appears in the top line – **rules-only** and **rulemix+wiring** are non-applicable.
- For a single rule network, **no rulemix** appears in the top line – **rulemix-only** and **rulemix+wiring** are non-applicable. To avoid this, the single rule can be set as a rulemix (section 14.4.3).
- For a k-mix, **rulemix-only** is non-applicable – **rulemix+wiring** apples instead.

Subject to these constraints, to save, load or print the network architecture enter two key strokes as follows,

first option	*second option*
p - to print	**w** - for wiring
l - to load	**r** - for rules
s - to save	**b** - for both wiring and rules

For example, enter **sw** to save a **wiring-only** only file (`.wso`), **sr** for a **rulemix-only** file (`.rso`), or enter **lb** to load a **rulemix+wiring** file (`.wrs`) which combines both wiring and rule scheme. The k-mix is implicit in the wiring encoding (section 19.3.1).

19.2 Wiring/rulemix file names

The binary files defining the wiring scheme (**wiring-only**), rule scheme (**rulemix-only**), or wiring and rule scheme combined (**rulemix+wiring**) have the following filename extentions and default filenames, where x is the value-range v (1 to 8).

	extention	*default file name*
wiring-only . . .	`.w_s`	`my_wso.w_s`
rulemix-only . . .	`.r_s`	`my_rso_`x`.r_s` for rcode
	`.r_v`	`my_rso_`x`.v_s` for kcode
	`.r_t`	`my_rso_`x`.t_s` for tcode
rulemix+wiring . . .	`.wrs`	`my_wrs_`x`.wrs` for rcode
	`.wrv`	`my_wrs_`x`.wrv` for kcode
	`.wrt`	`my_wrs_`x`.wrt` for tcode

19.3 Wiring/rulemix encoding

The encoding for all three file types above (section 19.2) starts with 6 bytes as follows[2], where k_{max} = the maximum neighborhood size in a mixed-k network (for homogenious-k, $k_{max} = k$), n is the network size, (i, j) the axes of a 2d network, and (i, j, h) for 3d,

[2] In the binary version of DDLab, the old style encoding started with 5 bytes. In the new style encoding byte 0 is reserved for the value-range v, and subsequent bytes are displaced by one. Old style files are nevertheless compatible for loading in the present version of DDLab. For single rule encoding see section 16.16 and for seed encoding see section 21.9..

byte 0 . . . value-range v, if byte 0=0 then old style encoding is recognized.

byte 1 . . . the lower 7 bits=k or max-k for a k-mix. The highest bit=1 indicates a k-mix, the highest bit=0 indicates homogeneous-k.

byte 2,3 . . . network size, n.

byte 4,5 . . . i, j. If $i = 0$ and $j = 0$ the network is 1d. If both $i \geq 1$ and $j \geq 1$ the network can be either 2d or 3d. $h = \frac{n}{i \times j}$. If $h > 1$ the netork is 3d, otherwise 2d.

The encoding continues as follows, for each successive cell index from $0 \ldots (n-1)$, where the number of bytes encoding a rule-table, R, depends on the rule type, the value-range v and the neighborhood k as specified in "Single rule file encoding" section 16.16,

file type . . . *bytes required*

R**: rulemix-only** . . . B bytes, starting with the least significant bit/value in the rule-table.

W**: wiring-only** . . . W bytes: $W = k_{max}$ if $n < 255$, $W = 2 \times k_{max}$ if $n \geq 255$, indexed $0 \ldots k_{max} - 1$ starting with wiring index 0.

wiring+rules . . . The combined wiring-rule scheme, R followed by W.

19.3.1 Mixed-k encoding

For mixed-k networks, the k-mix is implicit in the wiring encoding. Where the actual k of a cell is smaller than k_{max}, there will be excess bytes for both the rules and wiring in the encoding in section 19.3 above. These excess bytes are filled as follows,

rules . . . excess bytes are initially filled with 0s, but may subsequently contain obsolete values which are irrelevant.

wiring . . . excess bytes are filled with 255 (hex ff) if $n < 255$, or pairs of bytes with 65535 (hex $ffff$) if $n \geq 255$.

Counting the number of bytes not containing ff or $ffff$ is how the file keeps track of the k-mix. Note that k=0 is illegal, but an effective k=0 can be set (section 9.3 and 14.3).

The k-mix can also be saved separately in a .mix file (section 19.5).

19.4 Loading networks and sub-networks

Once a base network is set up and visible as a wiring graphic (section 17.3), previously saved network files, types **wiring-only**, **rulemix-only** and **rulemix+wiring**, can be loaded, either to replace the base compleatly or to insert a sub-network positioned somewhere within the base network. If successfully loaded the file network is automatically defined as a block (sections 17.5.3, 17.6.5 or 17.7.5), with its wiring visible in the wiring graphic – the block can be seperatly manipulated. Initially the block's wiring, whether local or random, will be self-contained, but can be wired-in to the base network as in figure 19.4. Any number of sub-networks can be loaded into the base.

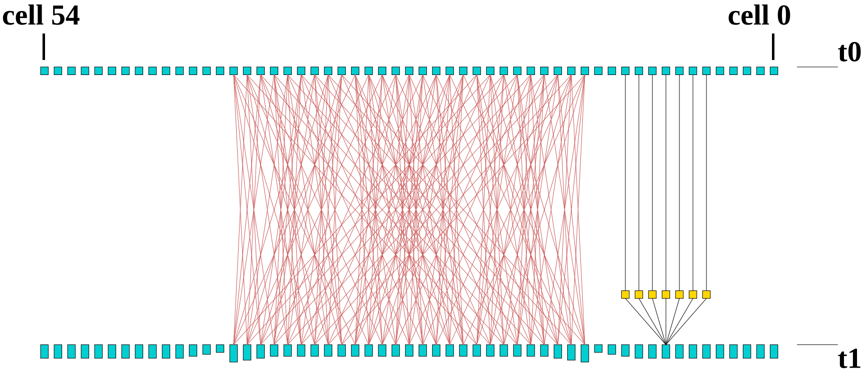


Figure 19.1: Loading a 3d wiring-only file ($3 \times 3 \times 3$, k=6) into 1d CA base network (n=55, k=7). The file appears in the 1d wiring graphic as a bock (section 17.5.3).

19.4.1 loading networks - compatibility

A valid file type in section 19.1 may be still turn out to be incompatible with the base network – which becomes clear once the file is selected. The rules for compatibility between the base and file network relate to value-range v, neighbourhood k, network size n, and with edge sizes i, j, h in 2d and 3d. The rules, exceptions, and other issues are described below (sections 19.4.1.1 to 19.4.1.4).

19.4.1.1 compatibility with v

For compatibility with value-range, v, $v_{file} = v_{base}$ is required. Otherwise a top-right error message appears,

> **file-v(2) != base-v(3), can't load, cont-ret:** *(for example)*

An exception applies if loading **wiring-only** from a wiring graphic before rules have been set, when v is irrelevant.

19.4.1.2 compatibility with k

For compatibility with neighbourhood k, $k_{max,file} \leq k_{max,base}$ is required. Otherwise a top-right error message appears,

> **file-kmax(7) > base-kmax(6), can't load, cont-ret:** *(for example)*

If a **wiring-only** or a **wiring+rules** file with mixed-k is loaded into a homogeneous-k base, the base will automatically be redefined as having mixed-k. Loading **wiring-only** into a homogeneous-k base with rules set will truncate rule-tables where $k_{file} < k_{base}$.

An exception to $k_{max,file} \leq k_{max,base}$ applies if loading **rules-only**; in this case $k_{file} = k_{base}$ (both with homogeneous k) is recommended. Loading a mixed-k file, or a file where $k_{file} < k_{base}$ is risky because k_{base} will remain unchanged – a smaller rule-table will simply overwrite the start of a larger base rule-table giving unexpected results. Attempting this results in a top-right warning message,

rulemix-only: loading kmix-file is risky, load anyway-l, abort-ret: *(or, for example)*
rulemix-only: loading file-k(3) < base-k(5) is crazy, load anyway-l, abort-ret:

19.4.1.3 compatibility with n

For compatibility with network size n, $n_{file} \leq n_{base}$ is required. Otherwise a top-right error message appears,

file-n(150) > base netsize(100), can't load, cont-ret: *(for example)*

An exception applies if loading a **rulemix-only** file, where both file and base have homogeneous-k and $k_{file} = k_{base}$. In this case $n_{file} > n_{base}$ will load as may rules as will fit.

19.4.1.4 compatibility with edge sizes and dimensions

For compatibility with edge sizes i, j, h in 2d or 3d, its recommended that file coordinates fit within the base coordinates – if so, as well as loading a file into a base with the same dimensions, lower dimensions can be loaded into higher – the file coordinates are first reallocated with the higher dimension, for example a 1d file network $n = 30$ is treated as 2d $i, j = 30\times1$ or 3d $i, j, h = 30\times1\times1$, a 2d file network $i, j = 30 \times 40$ is treated as 3D network $i, j, h = 30 \times 40 \times 1$. A top-right warning message will show incompatible dimensions,

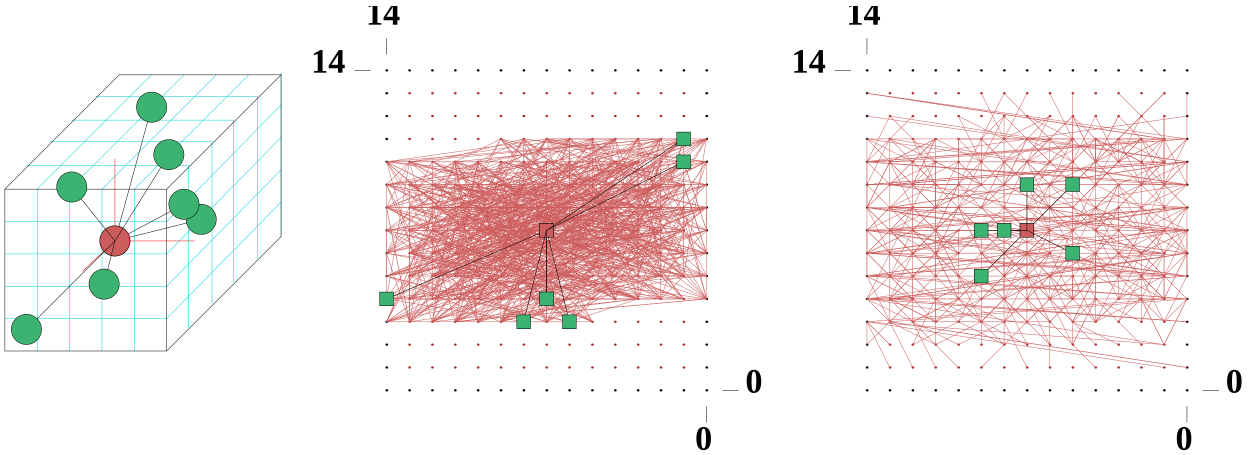


Figure 19.2: *left:* a small 3d DDN ($i, j, h = 5 \times 5 \times 5$, k=7) was saved as a **rulemix+wiring** file.
centre: the 3d DDN was loaded as a sub-network into the centre of a 2d DDN ($i, j = 15 \times 15$, k=9), with random wiring confined to a 5 cell local zone (section 12.5.2). The sub-network is automatically defined and visible as a block, but because the sub-network dimension is higher than the base, the block consists of consecutive cells. When first loaded the sub-network is self-contained with its original boundaries – the dynamics within the sub-network would be independent of the rest of the network, but not vise-versa.
right: The result of resetting the sub-network block with confined random wiring by pressing **r** in the wiring graphic reminder, section 17.4. This connects the sub-network to its surroundings. Note the active cell, which can be moved anywhere.

file-j(40) > base-j(30), load anyway-l, abort-ret: *(for example)*

If **l** is entered to load anyway, the start position will be set in section 19.4.3 as if the base were 1d. This is also the case if loading higher dimensions into lower, which can be done as long as $n_{file} \leq n_{base}$. Loading in these ways results in a consecutive block, not the usual 2d or 3d block in a 2d or 3d base, but the block can still be independently manipulated, as in figure 19.2.

19.4.2 Loading a complicated network into a CA and vice-vesa

In general, to load a file network with a rule-mix, the base also requires a rule-mix. but this can be confined to a limited number of rules, or just one rule. What if a complicated file network with a rule-mix, k-mix, and random wiring, (RBN or DDN) needs to be loaded as a sub-network into a simple CA base network? In such a case (figure 19.3) the CA base network can easily be set up where its single rule is treated as a rule-mix (section 14.4.3). Uniform k can also be treated as mixed-k (section 9.10), and local wiring as non-local (section 12.4.1), but neither is necessary.

Conversely, if a simple CA needs to be loaded as a sub-network into a complicated base network with mixed rules, the CA should be set up and saved as a rulemix with one rule. To load a CA into another CA, where their rules or k differ, both the base and the file should be have a rulemix with one rule.

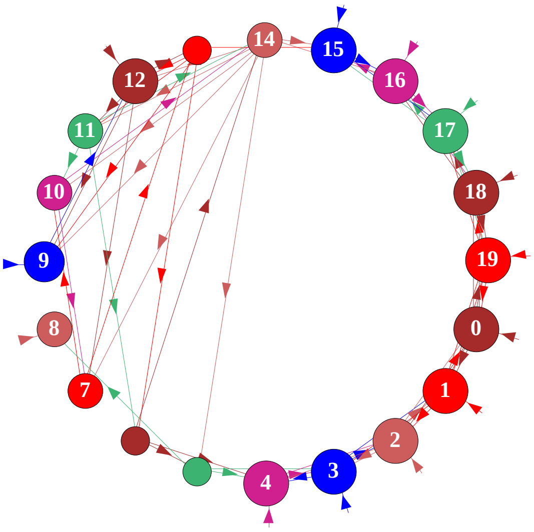


Figure 19.3: Loading a complicated sub-network (n=10) with a rule-mix, k-mix and non-local wiring into a simple 1d CA (n=20). To do this the CA is set up to have a rule-mix with just one rule. The figure shows the resulting network as a network-graph (chapter 20).

19.4.3 Loading sub-networks in a set position

A file sub-network that is smaller than the base network, which can fit into the base network as specified in section 19.4.1.4 can be loaded in a set position even if the dimensions of the file and base network are not the same, as long as $n_{file} \leq n_{base}$.

If the dimensions and edge sizes are equal, the file will be loaded without further ado, otherwise prompts allow the file sub-network to be positioned in the base network (the default is a central position).

For a 1d base, or if edge sizes do not fit, or higher dimensions are being loaded into lower and **load anyway-l** is selected in section 19.4.1.4, prompt is as follows,

for 1d networks
1d: array length=150, length file=14, enter start pos *(for example)*
(def 68, max 136, rnd-r)):

For a 2d or 3d base, if edge sizes fit, the prompt is as follows,

for 2d networks
2d:i,j=66,66, file: i,j=10,10, enter start coords *(for example)*
(def 28,28, max 56,56, rnd-r)) i: j:

for 3d networks
3d:i,j,h=40,40,40, file:i,j,h=9,9,9, enter start coords *(for example)*
(def 15,15,15 max 31,31,31, rnd-r)) i: j: h:

Enter the start coordinates for positioning index 0 of the file sub-network, **return** for the default central position, or **r** for a permissible random position.

Any number of independent sub-networks can be loaded consecutively. The the sub-network block will be visible and active in the base wiring graphic, where it can be be wired-in or otherwise maniputated, Deactivate/reactivate the block with **g**, cancel the block with **b** then **return**, in the wiring graphic reminder (section 17.4). Sub-network can be wired into other sub-networks or other parts of the network as described in chapter 17.

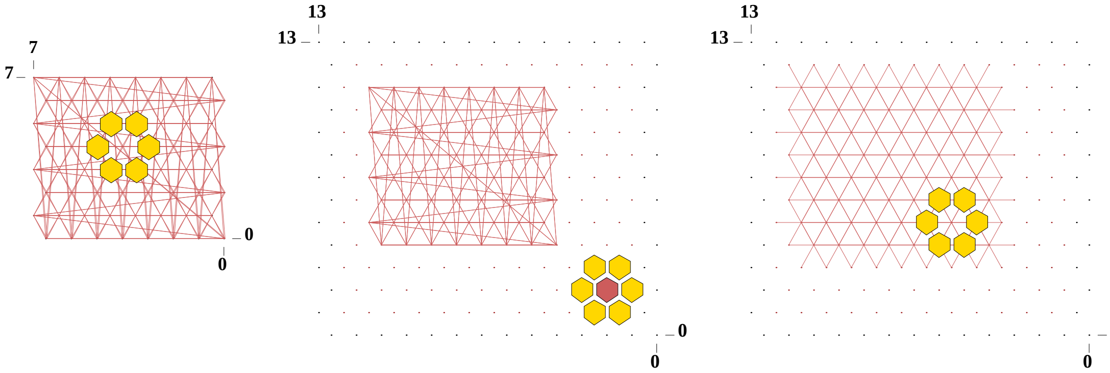


Figure 19.4: *left:* a small 2d hex CA ($i,j = 8 \times 8$, k=6) was saved as a **rulemix+wiring** file.
centre: the CA was loaded as a sub-network into a larger 2d hex CA k=7 ($i,j = 14 \times 14$, k=7), positioned at coordinates $I,J = 4,4$ and automatically defined and visible as a block in the wiring graphic. Because the CAs are both hexagonal, for correct wiring, j in both networks, and the position coordinate J should be even. When first loaded the sub-network CA is self-contained with its original periodic boundaries – the dynamics within the sub-network would be independent of the rest of the network, but not vise-versa.
right: The result of resetting the sub-network block as 2d by pressing **2** in the wiring graphic reminder section 17.4. This connects the sub-network (the block) to its surroundings – removing its periodic boundaries. Note the active cell, which can be moved anywhere, showing its neighborhood.

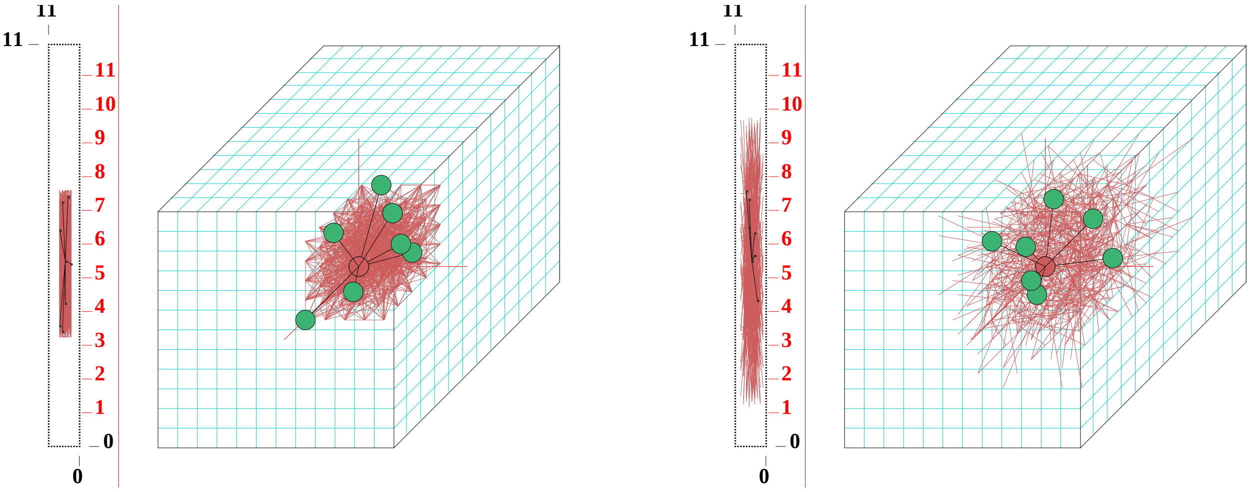


Figure 19.5: (showing the 2d+3d wiring graphic) *left:* A small 3d DDN ($i, j, h\ = 5 \times 5 \times 5$, k=7) (the same as in figure 19.2) was saved as a **rulemix+wiring** file, then loaded as a sub-network into the centre of a larger 3d DDN ($i, j, h\ = 12 \times 12 \times 12$, k=7). When first loaded the sub-network is automatically defined and visible as a self-contained block within its original boundaries – the dynamics within the sub-network would be independent of the rest of the network, but not vise-versa.
right: The randon wiring for the block was first confined to a 5 cell local zone (section 12.5.2), then its wiring was randomised **r** in the wiring graphic reminder, section 17.4. This connects the sub-network to its surroundings.

19.4.4 Loading mixed-k networks

A file network with mixed k can be loaded into a homogenious k or a mixed-k base network provided that the base k or max-k is at least as big as the file k or max-k (the precise base k-mix is not relevant). A base max-k smaller than the file max-k gives the following error message,

mixed k, but max net k3 smaller than max file k5 *(for example)*

The max-k in a mixed k file network may be smaller than max-k in the base network. Max-k can be reduced to the actual maximum k found in the network as described in section 18.8.

Note that only the combined network file, wiring and rules, can be loaded into a base network with mixed k. The combined file need not be a mixed k file. To load “just rules” or “just wiring” in section 19.1, the base network must have homogeneous-k.

19.5 Saving just the k-mix

Enter **k** in section 19.1 to save just the k-mix (see also section 9.12.3). A top-right filing prompt appears to save a `.mix` file (default filename `mymix.mix`), which is encoded in n bytes, recording the neighborhood size k for each cell (index 0 to n-1) in each byte. Information on the k-mix is also implicit in the **wiring-only** (`.w_s`) and **rulemix+wiring** (`.wrs`) files (section 19.3.1), so usually the k-mix does not need to be saved separately. Loading just the k-mix from a `.mix` file is only possible towards the start of DDLab (section 9.5).

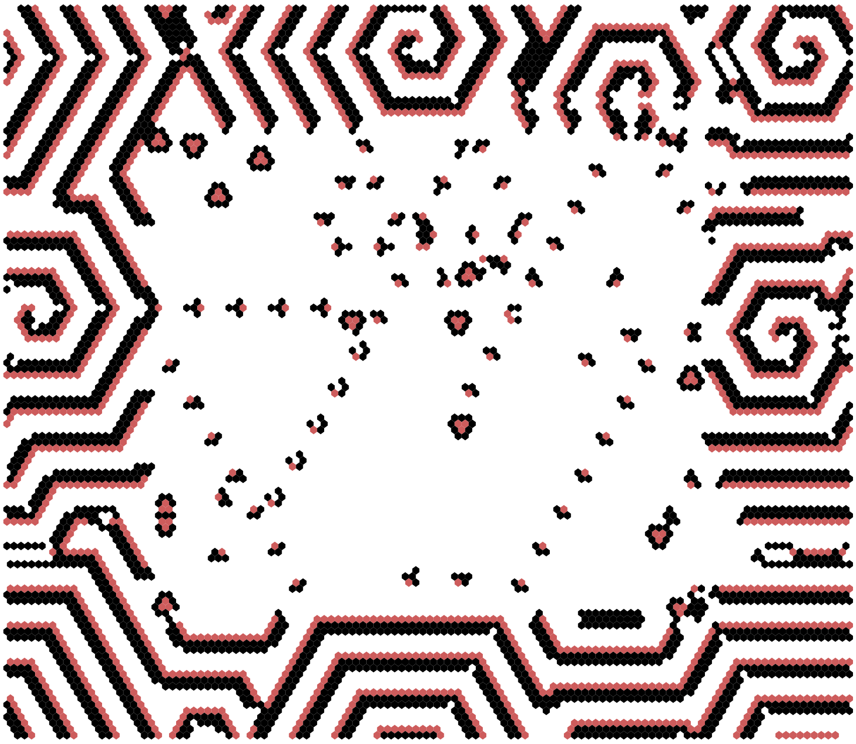


Figure 19.6: Space-time snapshot of a 2d CA inside 2d CA with different k. The 2d $k = 7$ CA spiral-rule (80×80) was loaded into another 2d $k = 6$ complex CA (120×120) which generates spiral structures. (see also figure 32.9)

19.6 Printing network data to the xterm/printer or file

If **p** (followed by **w**, **r** or **b** as applicable) is selected in section 19.1, the following prompt is presented,

for Linux-like systems
network text data: to xterm-p, file-f both-b:
for DOS
network text data: to printer-p, file-f both-b:

Enter **p** to print the data to the xterm window in Linux-like systems, or to a printer in DOS. Enter **f** to save the ascii data to a `.dat` file, or **b** to print and save simultaneously. Before saving, a top-right window will prompt for the file name (default `my_net.dat` – see Filing, chapter 35).

The way the data is presented depends on the selections **w**, **r** or **b** in section 19.1, mixed-k or homogenious-k, the type of rule, and if in TFO-mode. The data starts with two introductory lines. The first shows the value-range, k or the k-mix, and the network dimensions i, j, h and size n, for example,

```
v2k5, 1d n=7    ... for v=2, homogeneous-k, 1d network
v3kmix(4-1), 2d 55 n=25    ... for v=3, k-mix, 2d network 5×5
v2k4, 3d 6x6x6 n=276    ... for v=2, homogeneous-k, 3d network 6×6×6
```

The second line is a reminder that labels the data and gives the data order, for example,

```
cell.  wiring(3-0), rcode(hex) ld ld-r Z C A ...for rcode
cell.  wiring(13-0), kcode(hex) rcode(hex) ld ld-r Z C A ... for kcode+rcode
cell.  wiring(9-0), tcode(hex) rcode(hex) ld ld-r Z C A ... for tcode+rcode
cell.  wiring(5-0), kcode(hex) ... for kcode in TFO-mode (or tcode)
```

label ... *what they means*

`cell` ... 1d index of the cell.

`k` ... for a k-mix only, the neighborhood size k for the cell.

`wiring(3-0)` ... wiring scheme of the pseudo-neighborhood in this case for 3,2,1,0 for k=4.

`kcode` ... kcode in hex (if applicable).

`tcode` ... tcode in hex (if applicable).

`rcode` ... rcode in hex (if applicable) – kcode or tcode will be shown before the rcode if a totalistc rule was selected in section 13.1.1.

`ld` ... λ parameter.

`ld-r` ... λ ratio.

`Z` ... Z parameter.

`C` ... canalizing shown in two parts (for k=3) firstly the fraction of canalizing inputs (0/3, 1/3 *dotts*), then which of the k inputs are canalizing, i.e. *1* - input 1 is canalizing, 2** - input 2, *10 - both inputs 1 and 0. If there are no canalizing inputs the second part is not shown.

`P` ... Post functions info (see section ??) – 0 is shown if there are no Post functions.

The examples of network data below as shown in the terminal (xterm) are for small networks, though the data can be very lengthy for larger v, k and n.

```
v2k5, 1d n=7
cell.  wiring(4-0), kcode(hex) rcode(hex) ld ld-r Z C A
---------------------------------------------
6.  1 0 6 5 4, 33 e8818117 0.375 0.75 0.5 0/5 0
5.  0 6 5 4 3, 20 80000000 0.0312 0.0625 0.0625 5/5 43210 A[1]i
4.  6 5 4 3 2, 1c 7ffefee8 0.781 0.438 0.312 0/5 0
3.  5 4 3 2 1, 3e fffffffe 0.969 0.0625 0.0625 5/5 43210 A[0]i
2.  4 3 2 1 0, 31 e8808001 0.219 0.438 0.312 0/5 0
1.  3 2 1 0 6, 21 80000001 0.0625 0.125 0.125 0/5 0
0.  2 1 0 6 5, 3d fffefee9 0.844 0.312 0.312 0/5 A[0]3
average:ld-r=0.3125 Z=0.241071 average network parameters
weighted average:ld-r=0.3125 Z=0.241071 weighted according to the proportion of cell outputs
```

The example above is for kcode in the context of full lookup-tables (not TFO-mode) mode so both kcode and rcode are listed. The rule parameters (`ld ld-r Z C A`) follow the rcode. The data ends with a summary of average network parameters and the weighted average according to the proportion of cell outputs.

The example below is a k-mix and kcode in TFO-mode. Rule parameters (`ld ld-r Z C A`) do not apply in TFO-mode.

```
v3kmix(7-1), 2d 3x3 n=9
cell.  k, wiring(6-0), kcode(hex)
---------------------------------------------
8.  3, - - - - 6 8 7, 041061
7.  1, - - - - - - 7, 15
6.  5, - - 0 7 6 8 3, 028295984826
5.  7, 6 8 3 5 4 0 2, 41a402145666095140
4.  3, - - - - 5 4 3, 020a60
3.  2, - - - - - 4 5, 080a
2.  7, 5 4 0 2 1 8 7, 09625a59582169504a
1.  5, - - 4 2 1 0 7, 0141051129a0
0.  4, - - - 3 1 2 6, 02551448
```

Intentionally Blank

Chapter 20

The network-graph, and attractor jump-graph

DDLab includes powerful graph tools which have two distinct applications. Firstly, to represent the CA, RBN or discrete dynamical network itself - the network-graph, how the network elements are connected by their directed links, their wiring scheme. Secondly, to represent the stability of basins of attraction, the probability of jumping between basins due to perturbations to attractor states - the jump-graph[1] of the basin of attraction field, where nodes represent basins of attraction, and edges represent jump probability, including jumps from a basin back to itself. The underlying graph data can also be shown as a table, called the "network-table" or "adjacency-matrix" for the network-graph, and the "jump-table" for the jump-graph. As the graph manipulation options are the same for the network-graph and the jump-graph, they are described together in this chapter. The attractor jump-graph also applies to the attractor histogram (section 31.19.2 and figure 31.17.3) where basin properties are found statistically by running a network forwards from many random initial states.

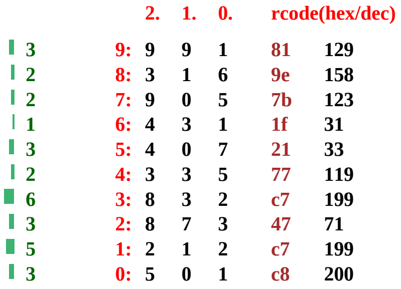
The network-graph does not allow changes to the underlying network (see chapter 17 to do this), but for both the network-graph and jump-graph, flexible methods are available for rearranging and unraveling the graph, including dragging vertices and defined components to new positions with elastic links, rescaling nodes and links, and changing links just within the graph.

Both the network-graph and jump graph can be shown without links (layout only) which is applied to present space-time patterns or basins of attraction in arbitrary layouts. The layouts can be saved/loaded (`*.grh` file) to restore a particular graph. The graphs, including basins of attraction positioned according to graph nodes, can be saved as vector Postscript files.

20.1 Unraveling the jump-graph and the network-graph

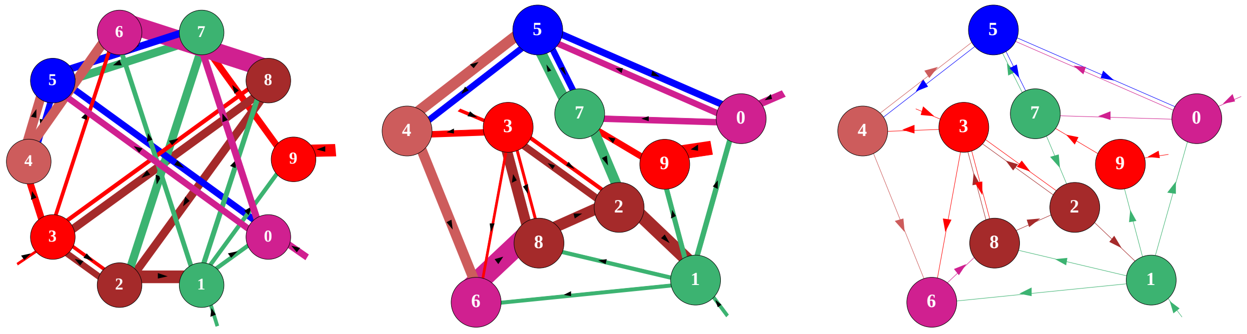
The layout of the graphs can be rearanged and unravelled to reveal the topology. Single nodes, connected fragments, or whole components, can be dragged with the mouse, according to inputs and/or outputs. The distance of fragment links from a node can be restricted, i.e. dragging the node + its immediate links (step 1), the node + immediate links + their immediate links (step 2), etc. Arbitrary 1d, 2d and 3d blocks can also be defined and dragged. Nodes with the fewest links can be automatically moved to the outer edges.

[1] In the July 2001 DDLab manual some of the jump-graphs examples were incorrect.



		2.	1.	0.	rcode(hex/dec)	
3	9:	9	9	1	81	129
2	8:	3	1	6	9e	158
2	7:	9	0	5	7b	123
1	6:	4	3	1	1f	31
3	5:	4	0	7	21	33
2	4:	3	3	5	77	119
6	3:	8	3	2	c7	199
3	2:	8	7	3	47	71
5	1:	2	1	2	c7	199
3	0:	5	0	1	c8	200

Table 20.1: The network "matrix" of the RBN $v2k3$, n=10, used in some of the examples: the network-graph (figure 20.1), the adjacency-matrix (table 20.2), the jump-graphs (figures 20.4, 20.12), "ant hits" (figure 20.5.3), and the jump-table (table 20.3). Section 17.2 describes how a network matrix is generated.



(a) default circle layout

(b) nodes rearranged

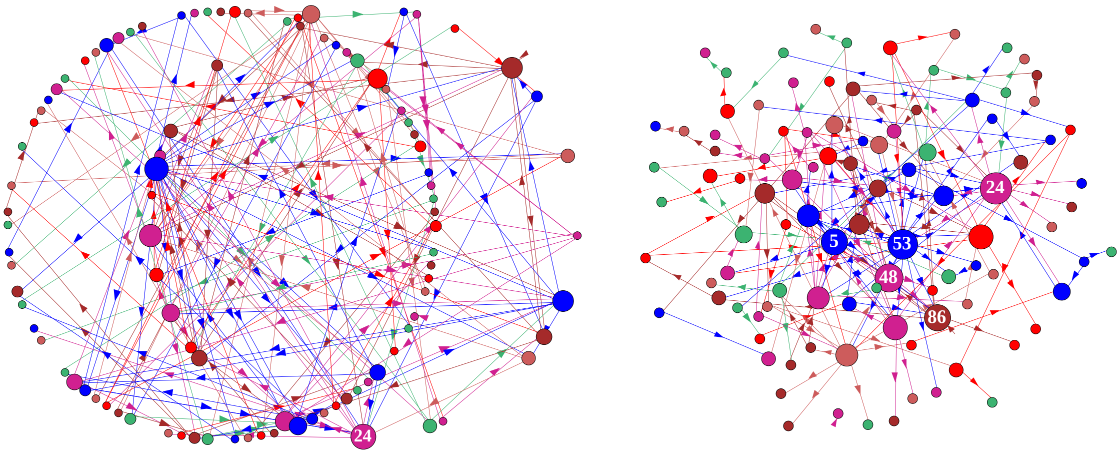
(c) links reduced to simple lines

Figure 20.1: Network-graphs of the same RBN $v2k3$, n=10, defined in table 20.1 showing alternative presentations (a) The default circle layout where edge width reflects the nodes's outputs, and node diameter reflects k, the node's inputs, all equal in this example. (b) Layout rearanged by dragging nodes with the mouse. (c) Directed links shown as thin lines with arrows.

The pre-programmed graph layouts available are a circle of nodes, a spiral, 1d, 2d or 3d, and random. Any layout can be "shaken" to randomly reposition nodes close to their current position. The graph can be rotated, expanded, contracted, and flipped. The graph is displayed in a large window across the lower part of the screen, smaller for the jump-graph to allow space to show the basin of attraction field above the window.

The default presentation for the jump-graph or 1d network-graph is a circle layout (figures 20.1(a), 20.2). The default network-graph layout for a 2d or 3d network is 2d or 3d (figures 20.7 - 20.9). The default sizes of nodes are scaled according to the neighborhood size k for the network-graph, or basin volume for the jump-graph. The width of links/edges can be scaled according to both node size and the proportion of available outputs comprising the link. If a node has connections to itself, this is represented by a short stub projecting from the node, also scaled as above. Proportional link/edge width is the initial default for the jump-graph, and uniformly thin lines with arrows for the network-graph, but this can be toggled. The nodes can also be toggled between proportional and a uniform size, and node numbering can also be toggled..

The nodes are numbered (if big enough) according to the cell order, 0 to n-1 for the network-graph, 1 to the number of basins for the jump-graph. Successive nodes are assigned different colors cycling through six colors. Outgoing edges are colored according to the parent node, and aligned asymmetrically clockwise relative to the parent, which ensures that outgoing and incoming edges do not overlap if two nodes link to each other; the directed edges are separated by a central gap.



(a) circle laout, one node dragged with its 2-step inputs (b) unscrambled

Figure 20.2: Two versions of the network-graph for the same network with a power-law wiring distribution, both inputs (k=1 to 10) and outputs, n=100. To set up power-law wiring see sections 9.8.2 and 17.8.5. Nodes are scaled according to k. (a) A circle layout, but with one node dragged, together with its 2-step inputs. (b) the same network unscrambled – nodes rearranged as described in in figure 20.6.

20.2 The network-graph

The network-graph is selected from the network architecture prompt (section 17.1), or from the space-time pattern pause prompt (section 32.14). The network architecture prompt itself is displayed at various stages in DDLab, firstly after special wiring is set (chapter 12), or after both the wiring and rules have been set in the main sequence of prompts, and at later stages. If **g** is entered a the network architecture prompt the following preliminary prompt appears,

no links -N, show links -def:

If **g** is entered at the space-time pattern pause prompt, the preliminary prompt has the opposite default,

show links -L (caution for large networks), layout only -def:

Showing the network-graph in "layout only", without links/edges, is useful for an alternative presentation of space-time patterns (section 32.15), which can be in any. If "layout only" is selected The network-graph options differ accordingly.

20.2.1 The network-graph reminder

The top-right network-graph reminder of the first set of options available is presented as follows,

NETWORK-graph: drag-(def) PScript-P ant-a unscram-u tog:win-w rank-k settings-S rotate-x/X flip-h/v exp/contr:nodes-e/c links-E/C both-B/b Unreach-U tog:table-t/T nodes-n/N links-l arrows-A/</> layout:file-f circle/spiral-o/O 1d/2d/3d-1/2/3 rnd-r/R quit-q:

For "layout only" (no links) some options in the reminder are omitted,

NETWORK-graph (no links): drag-(def) PScript-P tog:win-w rank-k settings-S rotate-x/X flip-h/v exp/contr:nodes-e/c links-E/C both-B/b tog:nodes-n/N
layout:file-f circle/spiral-o/O 1d/2d/3d-1/2/3 rnd-r/R quit-q:

For large network-graphs (with links) the following top-right message may be displayed first, with an inset window that monitors progress such as $\boxed{\textbf{55\%}}$,

computing network-graph, 1600 nodes, quit-q, or wait...

The network-graph is shown in a large window occupying most of the screen. These options are summarised in section 20.4. Section 20.5 covers options for dragging nodes and fragments.

20.3 The jump-graph of the basin of attraction field

Perturbations due to noise or external signals are most likely to take affect once a system has relaxed to its attractor, because thats where the dynamics spends the most time. Taking single-bit or single-value perturbations to attractor states as the simplest case, the jump-graph represents the probabilities of jumping between basins, and gives some insight into the stability and adaptability of the dynamics, which is especially relevant in attractor models of memory in neural networks and of cell differentiation in genetic networks.

The jump-graph algorithm analyzes the basin of attraction field to track where all possible single-bit/value flips to attractor states end up, whether to the same, or to which other basin. The information is presented in two ways: as a graph with weighed vertices and edges (figures 20.3 – 20.5), and as a jump-table – a matrix showing the jump probabilities between basins (20.12). The attractor jump-graph algorithm applies to any network (with either synchronous or sequential updating), for CA (with compression suppressed), for RBN or DDN, and also for random maps, and to any reverse algorithm including exhaustive testing.

An alternative use of the jump-graph is a method for just laying out attractor basins in any arbitrary postion. For this, jump edges are unnessecary – if omitted (layout only) attractor nodes can be dragged more efficiently and CA compression to be retained.

As well as the complete basin of attraction field, the jump-graph can also be computed for the attractor histogram (see sections 31.18 - 31.19.2), which gathers data on attractors and their relative sizes by statistical methods. This is done by running a network forwards from many random initial states, identifying different attractors, and creating a histogram of the frequency of falling into each. The method can be applied to large networks, especially RBN/DDN – too large to generate the basin of attraction field (see section 1.6 for network size limitations).

20.3.1 Selecting the jump-graph

FIELD-mode only - see also section 24.3

The jump-graph prompt is labelled **jump-j** in the "output parameters" sequence of prompts (section 24.1). To go directly to the jump-graph prompt enter **j** at the first output parameter prompt, or arrive there by viewing the output parameters in sequence. The following top-right prompt is presented,

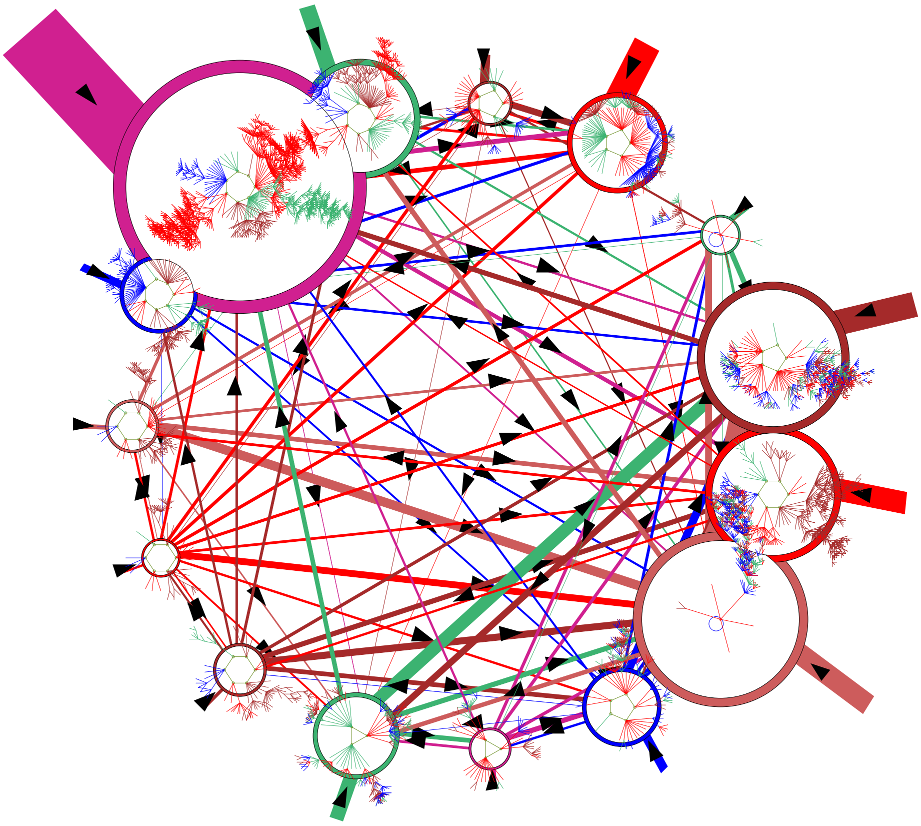


Figure 20.3: The jump-graph, with basins redrawn within its nodes (RBN k=3, n=13, as in figure 2.4). The jump-graph shows the probability of jumping between basins due to single bit-flips to attractor states. Nodes representing basins are scaled according to the number of states in the basin (basin volume), and can be rearranged and dragged. Links are scaled according to both basin volume and the jump probability. Arrows indicate the direction of jumps. Short stubs are self-jumps. When jump-graph links are eliminated this becomes a layout-graph, providing a method for arbitrarily arranging the basin of attraction field.

jump: attractor jump-graph -j, no edges layout only +L:

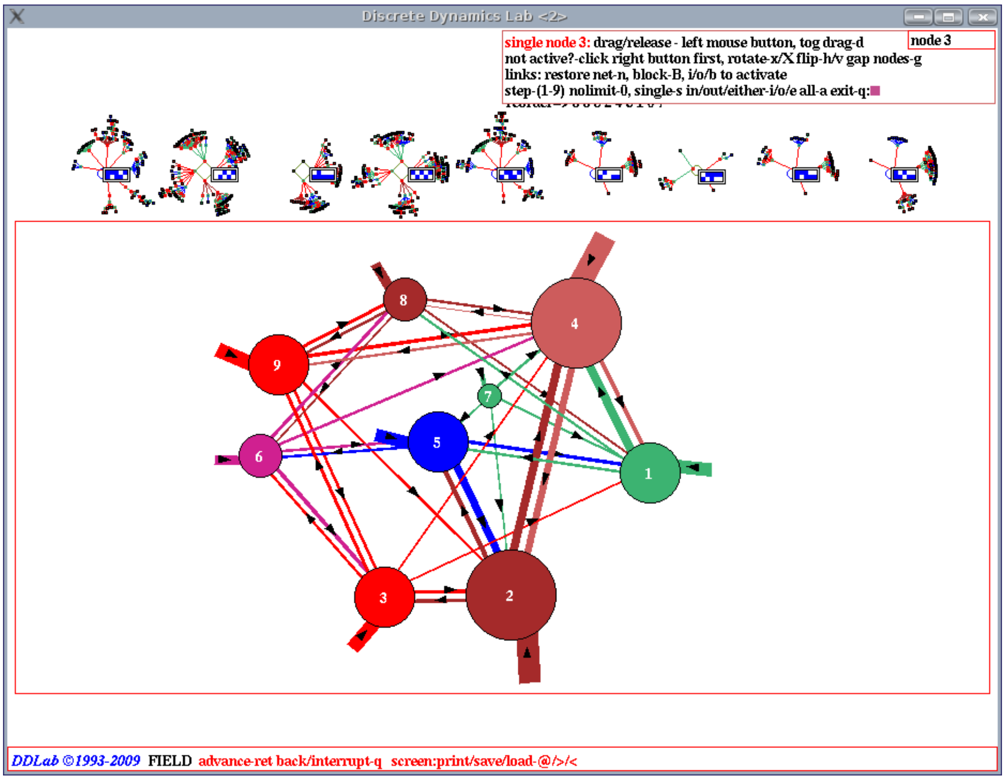
Enter **j** to show the jump-graph, or **jL** for "layout only" – the jump-graph without edges is applied for just laying out basins in a basin of attraction field in any arbitrary position. If the jump-graph (with edges) is selected, "compression" for CA will be turned off automatically, with the following message,

. . . **- compression OFF:** *(if compression was on, Enter* **return** *to continu)*

20.3.2 The jump-graph reminder

If selected in section 20.3.1[2] (once the basin of attraction field is complete) the jump-graph is automatically generated, and the top-right jump-graph reminder of the first set of options available is presented as follows,

[2] The jump-graph can also be selected in section 31.19.2 for the attractor histogram.



single node 3: drag/release - left mouse button, tog drag-d
not active?-click right button first, rotate-x/X flip-h/v gap nodes-g
links: restore net-n, block-B, i/o/b to activate
step-(1-9) nolimit-0, single-s in/out/either-i/o/e all-a exit-q:

node 3

DDLab ©1993-2009 FIELD advance-ret back/interrupt-q screen:print/save/load-@/>/<

Figure 20.4: A screen shot of the basin of attraction field at the top of the screen and its jump-graph in a large window across the lower part of the screen. The jump-graph was adjusted and nodes were dragged to new positions with the left mouse button. Drag options are shown in the top-right "drag reminder" (section 20.5.1). This is the RBN $v2k3$, n=10, defined in table 20.1. Basins were set to a small scale and repositioned near the top of the screen (chapter 25) so as not to be concealed behind the jump-graph window, though this can be togged to show hidden basins.

JUMP-graph: drag-(def) PScript-P ant-a unscram-u tog:win rank-k
settings-S rotate-x/X flip-h/v exp/contr:nodes-e/c links-E/C both-B/b
basins-i/I/s Unreach-U tog:table-t/T nodes-n/N links-l arrows-A/ </>
layout:file-f circle/spiral-o/O 1d/2d/3d-1/2/3 rnd-r/R quit-q:

For "layout only" (no edges) some options in the prompt are omitted,

JUMP-graph (no edges): drag-(def) PScript-P tog:win rank-k
settings-S rotate-x/X flip-h/v exp/contr:nodes-e/c links-E/C both-B/b
basins-i/I/s tog:nodes-n/N
layout:file-f circle/spiral-o/O 1d/2d/3d-1/2/3 rnd-r/R quit-q:

Jump-graphs (with edges) may take some time to compute if the basin of attraction field (or attractor histogram) has many long period attractors. The following top-right message may be displayed first, with an inset window that monitors progress such as $\boxed{\textbf{44\%}}$,

computing jump-graph, 25 basins, quit-q, or wait...

You might want to see a sequence of mutant basin of attraction fields and their jump-graphs – enter **q** to exit followed by **return**. The required mutation is selected in chapter 28.

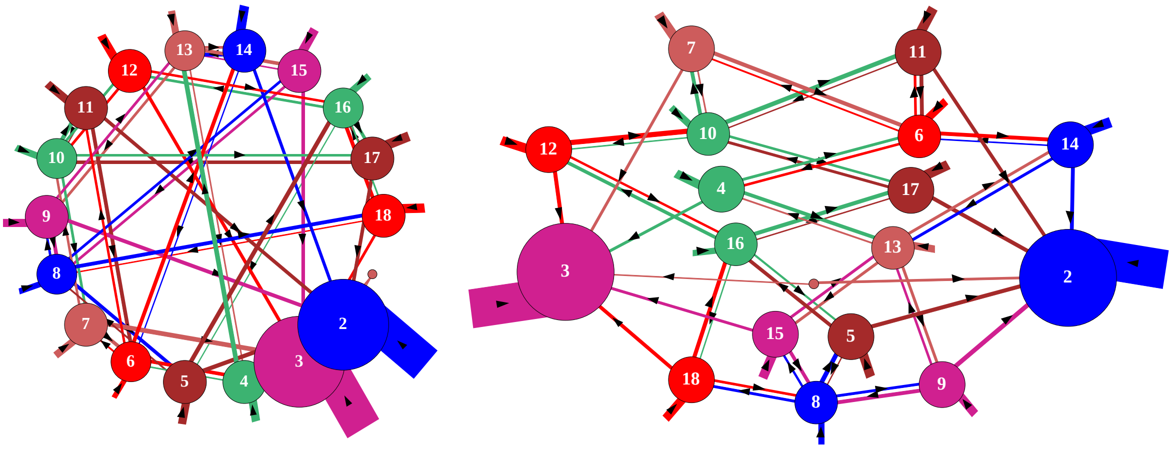


Figure 20.5: The jump graph for a 1d CA $v2k3$, n=10, rule (dec)141. *left* the default circle layout, and *right* rearranged with the mouse to clarify the jumps, where the two largest basins (2 and 3) are completely stable.

20.4 Initial graph options

A summary of the initial graph options in the network-graph and jump-graph reminders (sections 20.2.1 and 20.3.2) is given below. There is also a second set of options related for dragging nodes and fragments summarized in section 20.5. Further sections and figures in this chapter describe some of the options in greater detail. The options are essentially the same for the network graph and jump-graph, except for **basins inside-i/I/s** which applies only to the jump-graph. For a graph with **no links** or **no edges** the relevant options are omitted. Key hits take effect immediately, without entering **return**.

options . . . *what they mean*

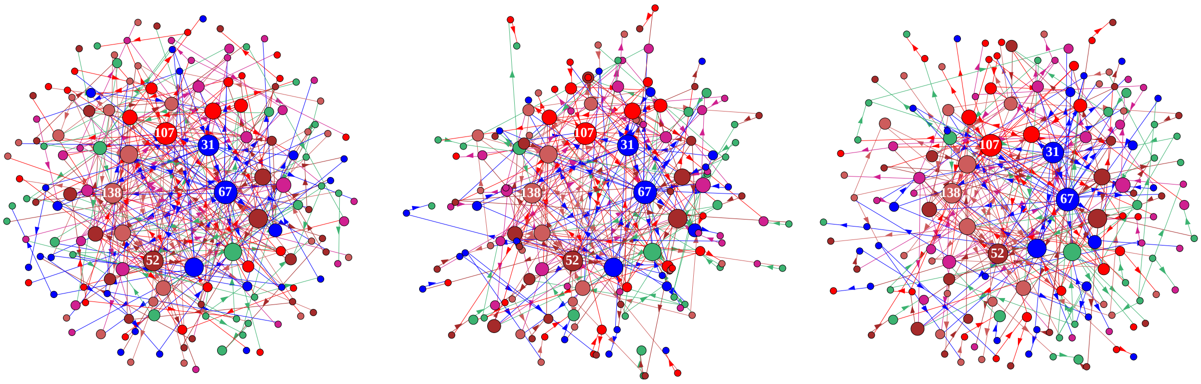
drag-(def) . . . enter **return**, or click the left or right mouse button, for the“drag” options (section 20.5). The drag reminder will replace the network-graph or jump-graph reminders – enter **q** to revert. Click on a node to activate it first in the drag options.

PScript-P . . . to save the graph as a vector PostScript file (section 20.8).

ant-a . . . to launch a projectile – a probabilistic “ant” – in the graph to trace a (Markov chain) path according to the link probabilities, keeping track of the frequency of visiting vertices (see section 20.6).

unscram-u . . . to automatically unscramble the graph, placing weakly connected nodes on the outer edges, nearby their connections. This works best in the circle or spiral layout (figure 20.6).

tog:win-w . . . to toggle the entire graph window – press any key to restore the graph. For the jump-graph this is usefull to uncover basins and and basin details – a top-right inset will appear **restore jump-graph -any key**



(a) show as a spiral, and rank (b) automatically unscramble (c) shake, then drag nodes

Figure 20.6: One possible method of unscrambling the network-graph of a scale-free RBN, n=150. (a): enter **O** to show the network as a spiral, then **k** to rank the nodes, placing nodes with the highest k near the center. (b): enter **u** to automatically unscramble the layout, possibly more than once. (b): enter **r** to "shake" the layout, then make final adjustments by dragging single nodes or fragments.

tog:rank-k . . . to toggle between the default and ranked node positions in the graph. The nodes are ranked (reordered) according to inputs (k) for the network-graph, or basin volume for the jump-graph.

settings-S . . . to revise the default settings for the rotation angle, and the scaling factors for nodes, links and arrows (see section 20.10).

rotate-x/X . . . enter **x** or **X** to rotate the graph clockwise or anti-clockwise (about the window center) by the default angle of 15 degrees, or a revised angle.

flip-h/v . . . enter **h** or **v** to flip the graph horizontally or vertically.

exp/contr:nodes-e/c . . . enter **e** or **c** to expand or contract the size of nodes and width of links by a default factor (1.2), or a revised factor.

exp/contr:links-E/C . . . enter **E** or **C** to expand or contract the length of links by a default factor (1.1), or a revised factor.

exp/contr:both-B/b . . . enter **B** or **b** to expand or contract both nodes and links at the same time by the default factors above.

basins-i/I/s . . . (*jump-graph only*) enter **i** or **I** to redraw the basins of attraction at the jump-graph nodes, or **s** to change the basin scale (see section 20.7). **i** shows just the basins, so is an alternative method of presenting the basin of attraction field with any layout created in the jump-graph, as in figure 20.11. **I** keeps the graph and draws basins inside the nodes, as in figures 20.3 and 20.12.

Unreach-U . . . enter **U** to identify and separate unreachable or hard to reach nodes in the graph, (see section 20.11).

table-t/T . . . to toggle between the graph and its corresponding table, shown in the same window. This is the network-table (adjacency-matrix) for the network-graph, or the jump-table for the jump-graph. **t** shows the numbers of links, **T** the fractions of total links, between each ordered pair of nodes (see section 20.12).

nodes-n/N . . . enter **n** to toggle between scaled and unscaled nodes. Enter **N** to toggle node numbers. Numbers only appear on those node that are big enough – about .7 of the current text height. The numbering is 0 to n-1 for the network-graph, 1 to the number of basins for the jump-graph.

links-l . . . to toggle between scaled links/edges and thin lines (see figure 20.1).

arrows-A/</> . . . enter **A** to toggle showing arrows. Enter < or > to decrease or increase the arrow size by a default factor (0.07), or a revised factor.

layout: *options* . . .

file-f . . . to save or load the current graph layout of node positions, a `.grh` file (see Filing, chapter 35).

circle/spiral-o/O . . . enter **o** or **O** to show the graph as a circle or spiral. Circle layout is the the default (except for the 2d or 3d network-graph). Spiral layout requires at least 30 nodes to be effective

1d/2d/3d-1/2/3 . . . enter **1**, **2** or **3** to show the graph arranged in 1d, 2d or 3d. This is especially relevant for a network-graph where the underlying network is 2d or 3d – then the graph in 2d or 3d is the default. For 2d with hexagonal connections the 2d network-graph is also hexagonal (examples in figures 10.6 - 10.8).

rnd-r/R . . . enter **r** to "shake" the layout, repositioning nodes randomly nearby their current position. Enter **R** for a completely random layout.

quit-q . . . enter **q** to quit the graph. For a network-graph this returns to the network architecture prompt in section 17.1. For a jump-grap this returns to the "Attractor basin complete prompt" section 30.4 – enter **return** for the next mutant.

20.5 Dragging nodes or fragments

Enter **return**, or click the left or right mouse button at the network-graph or jump-graph reminders (sections 20.2.1 and 20.3.2) to enable dragging nodes or fragments with the mouse, and other features; the "drag reminder" replaces the top-right network-graph or jump-graph reminders – enter **q** to revert. Click on a node to activate it first in the drag options.

The initial setup allows single nodes to be dragged. A node is activated and dragged with elastic links by clicking and holding down the left mouse button, releasing the node in a new position. Alternatively, elastic links can be suppressed and the active node dragged leaving a trail – on release the links (and fragment nodes if any) snap into position – for large graphs this is a more efficient method. Sometimes a node will not activate, if so, click on it alternately with the right then left mouse button. When a node is activated its number appears in the drag prompt, **node 5**, for example.

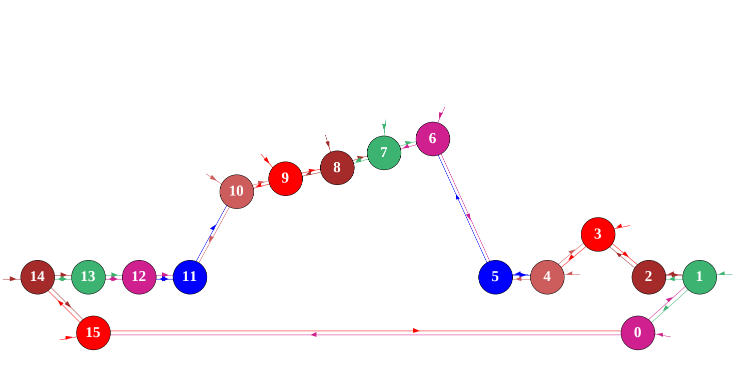


Figure 20.7: Dragging nodes and fragments of a 1d CA n=16, k=3, starting with a network-graph with 1d layout *left*), and circle layout (*right*). Single nodes have been dragged, and in both cases node 8 and its 2-step neighbors have been dragged and and rotated.

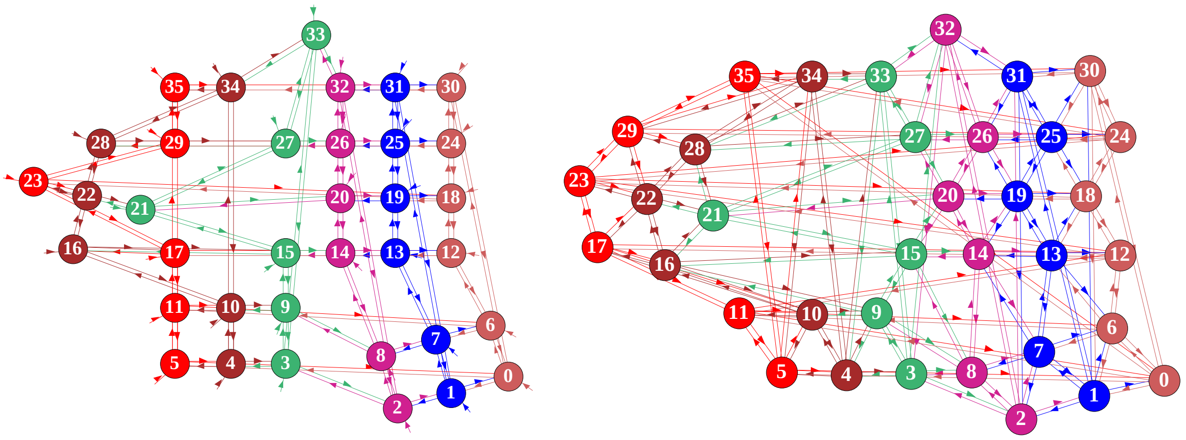


Figure 20.8: Dragging nodes and fragments starting with a regular 2d network-graph of a 2d CA 6×6, square k=5 (*left*), and hexagonal k=6 (*right*). In both cases, a node has been dragged from the top row, node 22 and its 1-step neighbors have been dragged and and rotated, and a fragment (0 – 8) has been dragged and and rotated from the lower right hand corner.

As well as dragging a single node, there are options for dragging connected fragments or arbitrary blocks. The fragments are defined as being at a given distance from a node according to input links, output links, or either inputs or outputs. Fragments can be separately dragged, rotated and flipped. Single nodes can have their inputs and/or outputs links cut. Directed links between pairs of nodes can be cut or added. So called “gap nodes”, which have only inputs, or only outputs, can be disconnected from the graph, showing the effective components, because gap nodes cannot transmit information. Note that breaking links and creating new links affects only the graph, not the original underlying links which can be restored.

20.5.1 The drag reminder

There are 4 types of interchangeable drag situations/reminders, listed below together with their main functions and how to select one from another,

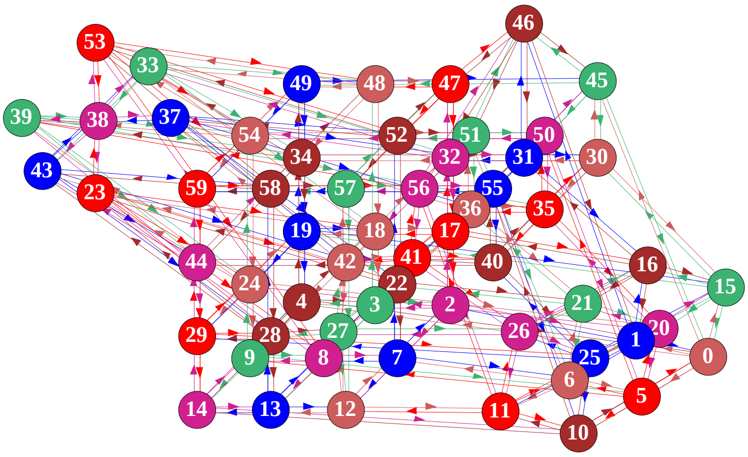


Figure 20.9: Dragging nodes and fragments starting with a regular 3d network-graph 5×3×4, k=6. Node 46 has been dragged from the top level, node 38 and its 1-step neighbors have been dragged and rotated, and a fragment (0 – 26) has been dragged and and rotated from the lower right hand corner.

type of drag . . . *functions and selection*

single node: . . . to drag a single node – enter **single-s** at any time.

links: . . . to drag a node and nodes linked to it by inputs, outputs, or either – enter **in/out/either-i/o/e** at any time.

block: . . . to drag a predefined block, the default is defined by the last two mouse clicks on nodes. To activate, first change to *single node*, then enter **block-B**.

all nodes: . . . to drag the complete graph – enter **all-a** at any time.

Enter **return**, or click the left or right mouse button in the network-graph or jump-graph prompts (sections 20.2.1 and 20.3.2) for the initial *single node* drag reminder – clicking on a node will select that node, clicking in empty space gives **node 0** (**exit-q** to revert at any time),

```
single node: drag reminder
node 0, single: drag - left button, tog drag-d (node 1 for the jump-graph)
not active?-click right button first, rotate-x/X flip-h/v gap-g
links: restore net-n, block-B
step-(1-9) nolimit-0 single-s in/out/either-i/o/e all-a exit-q:
```

The prompt changes if **i**, **o** or **e** (for inputs, outputs, or either) is entered, for example, entering **i**, and clicking node 4, then node 7, gave the following,

```
links: drag reminder (inputs, outputs, either) – step=nolimit or 1-9
node 7, links-inputs, step=nolimit: drag - left button, tog drag-d  (or outputs or either)
not active?-click right button first, rotate-x/X flip-h/v gap-g
links 7: cut/restore/net-c/r/n, link 7-4: cut/add/restore-C/A/R
step-(1-9) nolimit-0, single-s in/out/either-i/o/e all-a exit-q:
```

As the last node clicked was 7, links to that node can be cut. The one-but-last node clicked was 4, so **link 7-4** appears in the prompt – links between node 7 and 4 can be cut or added. When adding an input link its direction will point from 4 to 7. When adding an output link its direction will point from 7 to 4.

step=nolimit signifies that dragging a node will also drag the connected component, indirectly linked by either inputs, outputs, or either, along with it. This can be changed by entering a number **step-(1-9)**, to limit the size of the linked fragment. For example, entering 1 results in **step=1** in the top line, and only the nodes immediately linked to the active node will be dragged.

Enter **s** to revert to dragging single nodes. The option **block-B**, to define an arbitrary block to be dragged, is only available if **single** or **Block** is active – which appears in the top line. If **B** is entered, the block is first defined in 1d, 2d or 3d (see section 20.5.3). The default block is the block between the last 2 mouse clicks. Then the drag reminder changes to look like this (for example),

block: drag reminder
node 7, Block 4-7: drag/release - left mouse button, tog drag-d
not active?-click right button first, rotate-x/X flip-h/v gap-g
links: restore net-n, block-B
step-(1-9) nolimit-0, single-s in/out/either-i/o/e all-a exit-q:

The defined block can the be independently dragged from any node in the block, but dragging any single node will also drag an active block. Enter **B** for a new block.

20.5.2 Drag graph options

The following is a summary of the drag options, which take effect as soon as the key is hit, without entering **return**. "fragment" refers to a connected component or defined block.

options . . . *what they mean*

drag - left button . . . click the left mouse on a node to activate it and hold it down to drag the node or fragment – then release in a new position. Initially, to activate a node it may be necessary to click the right button first, then the left, possibly a few times. Hence the reminder,

not active?-click right button first

An active mouse cursor points North West instead of North East.

tog drag-d . . . enter **d** to toggle between two methods of dragging nodes or fragments. By default, dragging will move a node or fragment with elastic links/edges. Alternatively, elastic links can be suppressed and the active node dragged leaving a trail – on release the links (and fragment nodes if any) snap into position and the trail disappears. For large graphs this is a more efficient method. Enter **d** to toggle between the two methods.

rotate-x/X . . . enter **x** or **X** to rotate the graph or fragment by 15 degrees clockwise, or anticlockwise. Rotation is centered on the active node. The default rotation angle can be revised in section 20.10.

flip-h/v . . . enter **h** or **v** to flip the graph or fragment horizontally or vertically.

gap-g . . . enter **g** to cut all links to "gap nodes". Gap nodes have only inputs, or only outputs. Disconnecting them from the graph will show the effective components, because gap nodes cannot transmit information.

if **single node** *or* **Block** *is active*

restore net-n . . . enter **n** to restore the links in the graph (to the underlying network or jump-graph) that may have been cut.

block-B . . . enter **B** to define a block, as described in section 20.5.3. The outer edges of the default block are defined by the last two nodes that were activated with the mouse, but prompts are displayed to set the block by hand.

if **inputs, outputs** *or* **either** *is active*

links 23: cut/restore/net-c/r/n . . . *for example*

enter **c** to cut (disconnect), **r** to restore, the active node from/to the network by cutting/reconnecting all its input, output, or any link, depending on which option, **inputs, outputs** or **either** is active. Enter **n** to restore the graph to the underlying network or jump-graph.

link 23-19: cut/add/restore-C/A/R . . . *for example*

cut-C . . . enter **C** to cut, **A** to add, **R** to restore, links between the active node and the previously active node. The node numbers pair of nodes appear in the prompt. The links that are cut/added/restored are either input, output, or any link depending on which option, **inputs, outputs** or **either** is active.

in all cases

step-(1-9) . . . enter a number between **1** and **9** to limit a fragment by the distance from the active node. This relates to a continuous chain of directed edges – inputs, outputs, or any link, depending on which option, **inputs, outputs** or **either** is active. For example, enter **1** to limit the fragment to immediate links, **2** to include indirect links 2 steps away, etc., (up to 9 steps). The current status is shown in the drag reminder, for example **step=1**

nolimit-0 . . . enter **0** to define a fragment by a continuous chain of inputs, outputs, or either inputs and outputs, however indirectly, relative to the active node, depending on which option, **inputs, outputs** or **either** is active. Gap nodes would interrupt such a chain. **step=nolimit** is shown in the drag reminder.

single-s . . . enter **s** to restore dragging single nodes. The heading changes to **single node 23:** (for example).

in/out/either-i/o/b . . . enter **i**, **o** or **e** to to define the type of link – inputs, outputs, or either (i.e. any link), for dragging fragments or cutting links. The drag reminder will show the active status – **inputs, outputs** or **either**.

all-a . . . enter **a** to drag all nodes, the whole graph. The prompt heading changes to **all nodes:**.

exit-q . . . Enter **q** to exit the drag reminder and return to the the network-graph or jump-graph reminder (sections 20.2.1 20.3.2). Its easy to flip between the two reminders to implement alternative functions.

20.5.3 Defining a block

If **single node** or **Block** is active in the drag reminder (section 20.5.1), enter **B** to define a block[3] in 1d, 2d or 3d. For a 2d or 3d network-graph, a block can be defined according to its 2 outer corners. For a jump-graph or a 1d network-graph, the block is simply a range of nodes between an upper and lower limit (as in section 17.5.3). The easiest way to set the block is to click (activate) the outer corners to set the defaults. Make sure these are realy activated by checking that the node number appears in the drag reminder, for example **node 301**. Any pair of opposite corners can be defined by clicking in any order, but this will be converted automatically to the lower right and upper left corners.

One of the following top-right prompts is presented to accept defaults or set the block by hand, depending if the type of graph,

for a 1d network n=62, or a jump-graph

```
define 1d block:
1d block (0-61, def 49-59), (values shown are examples)
low:
high:
```

for a 2d network, 22x22

```
2d network: define block: 1d-1, 2d(def):
2d block (this=5,2 max=21,21), (values shown are examples)
low corner (def 5,2) i:     j:
high corner (def 7,5) i:     j:
```

for a 3d network, 6x6x6

```
define 3d network 1d-1 3d-(def):
3d block (this=0,3,0 max=5,5,5), (values shown are examples)
low corner (def 0,3,0) i:     j:     h:
high corner (def 5,3,5) i:     j:     h:
```

A 1d block is defined by entering the low and high node index. This also applies for a 2d or 3d network if **1d-1** is selected first, otherwise enter the coordinates of the opposite (low and high) corners to define the block, or accept the defaults – the last two nodes that were activated with the mouse, which are shown in the prompt. Enter **return** to accept each default. Once the block is defined, dragging any node (whether inside or outside the block) will also drag the block, and **rotate-x/X**, **flip-h/v**, will apply just to the block.

[3] The methods are similar to defining a block in chapter 17 "Reviewing network architecture" (sections 17.5.3, 17.6.5 and 17.7.5).

20.6 Probabilistic "ant"

Enter **a** in section 20.4 to launch a projectile or probabilistic "ant" in the graph. The ant has a red body and leaves a black trail on the center line between nodes. The default path starts at a randomly selected node and traces a Markov chain, a path that takes the next link according to the output probabilities. The "ant" keeps track of the frequency of visiting/hitting vertices for an arbitrary sample[4] of ant hits. While active, a number of ant options are presented in a top right prompt as follows,

ant-prompt
probabilistic ant: quit-q speed-< / > pause-p
restart rnd -r, redraw-d
show hits -h, tog:show ant(ON) -a, rnd ant(OFF) -s:

The meaning of these prompts are listed below,

options . . . *what they mean*

quit-q . . . quit the ant routine and return to the the options in section 20.4.

speed-</> . . . enter < to slow the ant down, or > to speed it up.

pause-p . . . to pause the ant – enter **p** again to resume.

restart rnd -r . . . restart the ant at a node selected at random – continue to keep track of node visits.

redraw-d . . . redraw the graph to delete the ant trail – continue to keep track of node visits.

show hits -h . . . puse the ant to show the frequency of ant hits so far. See section 20.6.1 for more details.

tog: show ant(ON) -a . . . toggle the display of the ant and its trail. The current status (ON or OFF) is indicated. If the ant graphics are off, the statistics of hits are gathered much faster.

tog: rnd ant(OFF) -s . . . toggle between a continuously moving the ant (the default) and restarting at random (biased) nodes for each step. The current status, ON or OFF, is indicated. See below for more details.

The **rnd ant(ON)** produces a different kind of sample; at each step the ant is re-started at a random node, biased by k (network-graph), or by the relative size of the basin of attraction that the node represents (jump-graph). Thus the ant can hit all reachable nodes even if they occur in separate components (ergodic sets) of the jump-graph.

By contrast, the default **rnd ant(OFF)** starts at a random node and follows a Markov chain, a continuous path according to the output probabilities in the graph. Some graphs are fully interlinked, but in a graph consisting of distinct components, if the path starts within such a component, it will be trapped inside. Note that the eigen vectors of the jump table of a component (which may be the whole graph) should be the same as the list of hit frequencies of the component.

[4] The maximum sample of ant hits is the maximum size of an unsigned long int -1 = 4294967294. If this is reached "Show ant hits" (section 20.6.1) is activated automatically.

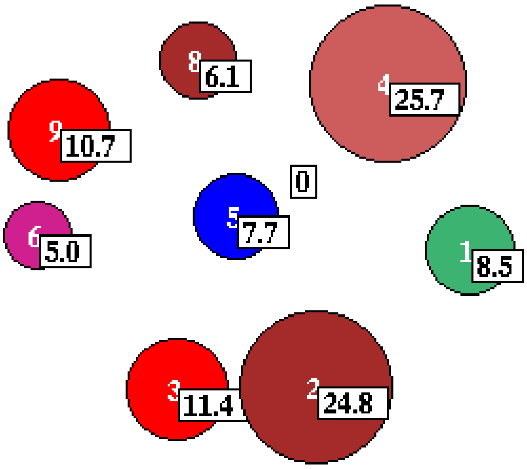


Figure 20.10: Probabilistic ant hits, or node visits (enter **h** followed **n** in section 20.6). This example is for the RBN $v2k3$, n=10, defined in table 20.1. 60 million jumps took a few seconds. The percentage of hits/visits to each basin is shown, and can be compared with basin volume in figures 20.4, 20.12 and table 20.3. Node 7 is unreachable from other nodes so has zero hits.

20.6.1 Show ant hits

Enter **h** while the probabilistic ant is active to show the number of hits so far and the percentage at each node. The nodes will be redraw and scaled according to hit frequency – edges, and nodes with zero hits are not shown. The following top-right options are presented, for example,

hit-prompt
showing % hits on each basin *(or* **each node** *for a network graph)*
total hits so far=1453697
show % frequency -n noNodes-N, quit-q cont-ret:

options . . . *what they mean*

show % frequency -n . . . enter **n** to add the frequencies of hits (as a percentage of total hits so far) to the scaled nodes (as in figure 20.5.3).

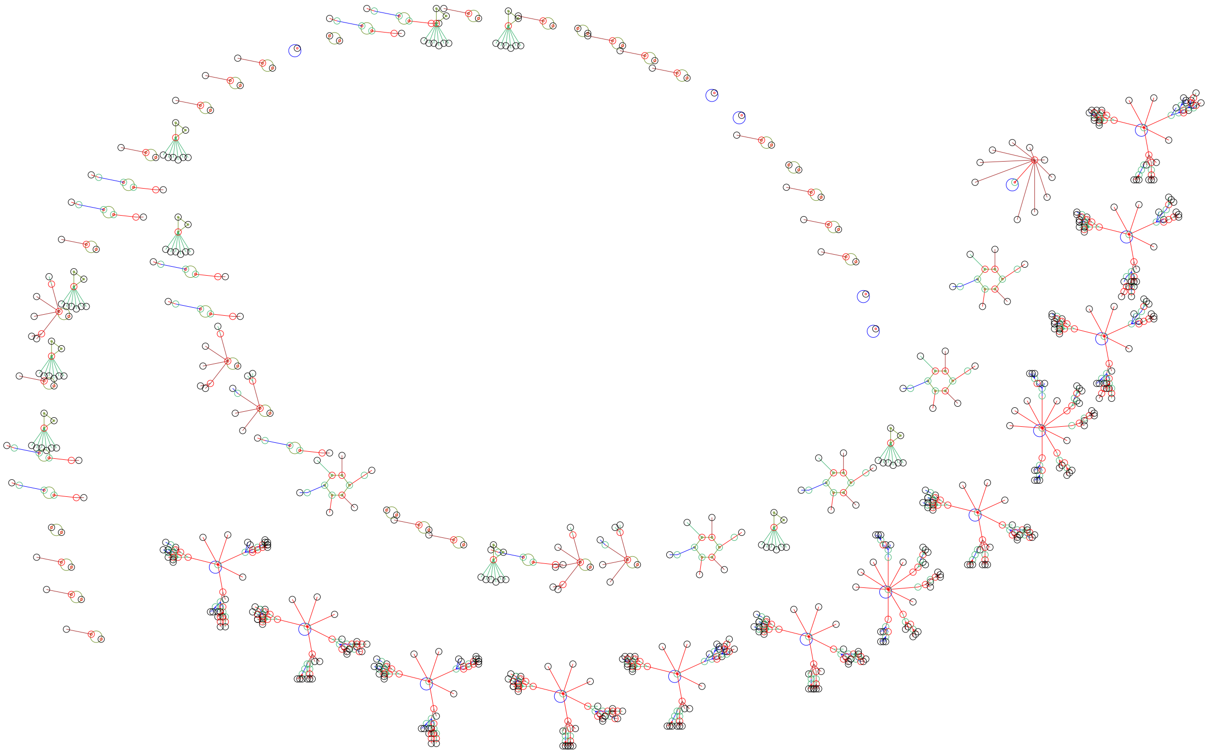
noNodes-N . . . enter **N** to show just the frequencies without showing the nodes, which might be necessary if large nodes obscure the data.

quit-q . . . enter **q** to quit the ant/hit-options and return to the graph reminders (section 20.2.1 or 20.3.2).

cont-ret . . . enter **return** to continue accumulating the ant-hits, returning to the ant-prompts (section 20.6).

20.7 Redraw basins at jump-graph nodes

When drawing the basin of attraction field, the number of basins, their attractor periods, transient lengths, in-degree – so the sizes and shapes of basins, are hard to predict. A compact layout without overlaps is somewhat difficult using the methods described in chapter 25, which describes the layout in the main window. An alternative flexible layout method is to select the jump-graph (with or without edges). As illustrated in figure 20.11, a jump-graph without edges allows a compressed basin of attraction field showing just the prototype basins (section 26.1).



Above: all 73 basins – nodes and fragments were dragged starting from a circle layout.
Below: the 11 prototype basins – starting from a square layout.

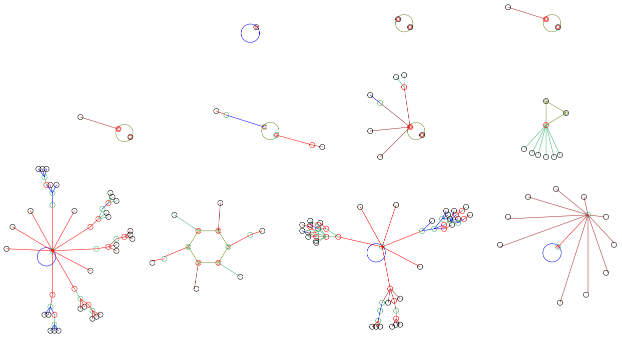


Figure 20.11: Inserting basins in the jump-graph for a 1d CA $v2k3$, n=10, rule (dec)133. For a 1d or 2d CA, if edges were omitted in the jump-graph (section 20.3.1), the basin of attraction field will be compressed by default (section 26.1) to include just the prototype (non-equivalent) basins. Otherwise all the basins in state-space are shown.

Once the basin of attraction field has been drawn in the main window, revealing the size/shape information, the jump-graph can be rearranged for an appropriate layout, and basins can be redrawn at the jump-graph nodes. The scale of basins can be changed just for this proceedure. The basin of attraction field with its new scale and layout will be saved as a PostScript file if this option was set in section 24.2.

Enter **i** in section 20.3.2 to redraw the basins centred at the jump-graph nodes positions, but without the jump-graph itself – supressing nodes and/or edges, as in figure 20.11. Enter **I** to keep the jump-graph, drawing basins within the nodes as in figure 20.12.

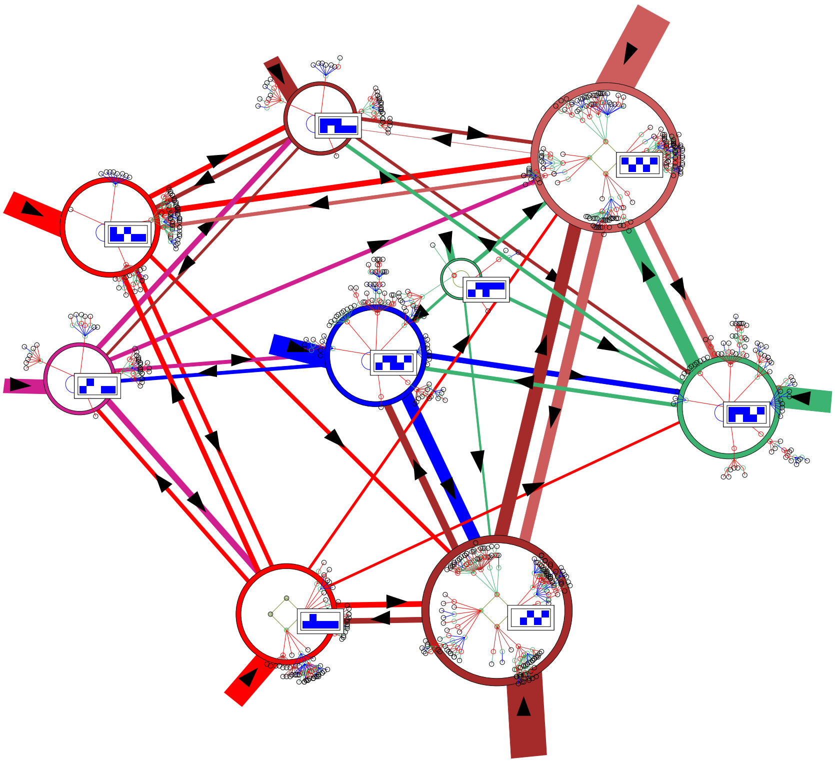


Figure 20.12: Basins of attraction redrawn at the jump-graph nodes, retaining the jump-graph itself. In this example one state in each attractor is also displayed. To create this figure as a vector PostScript file, the separate jump-graph and basin postScript files were combined This example is for the RBN $v2k3$, n=10, defined in table 20.1.

Enter **s** in section 20.3.2 to first rescale the basins to be redrawn in the jump-graph, the following top-right prompt is presented,

change basin scale for graph
rad main window=27.7, in graph=9.0
select attractor rad (min 0.24 def 9):

Enter the new attractor radius which can be estimated in relation to the basins in the main window. Enter **return** to revert to the options in section 20.4 and redraw the jump-graph.

20.7.1 PostScript of jump-graph basins

Creating a vector postScript file of attractor basins, whether drawn in the main windows or in the jump-graph, is activated in the "output parameters" sequence of prompts, labelled **PScript-P** in section 24.1. To go directly to the PostScript prompt enter **P** at the first output parameter prompt, or arrive there by viewing the output parameters in sequence.

The following top-right prompt is presented (see also section 24.2),

if saving to postScript is currently active
save basin to postScipt (now p): greyscale-P color-p cancel-0: *(for example)*

if saving to postScript is currently inactive
save basin to postScipt (now OFF): greyscale-P color-p: *(for example)*

Enter **0** to deactivate, **P** or **p** to activate – a filename prompt will be presented (section 35.2). If saving to postScript is active, a new file will be created and overwritten to the selected filename each time an attractor basin is drawn (default filename `my_bPS.ps`). To conserve the file of the basins in the jump-graph, the selected filename should be renamed with utilities[5] outside DDLab before a new attractor basin is generated.

The postScript files of the attractor basin and the jump-graph (section 20.8) are two seperate ascii files, but can be combined (as in figure 20.12) by appending the basin file (without headers) at the end of the jump-graph file in an external editor. Alternatively, concatenate the files in the terminal – "`cat my_jgPS.ps my_bPS.ps > combined_file.ps`" – for example. .

20.8 PostScript of the network-graph or jump-graph

Enter **P** at the network-graph or jump-graph reminders (sections 20.2.1 and 20.3.2) to save the graph as a vector postScript file. The following top-right prompt is presented,

postScript: greyscale-P color-p:

Enter **P** or **p** for greyscale or color. A filename prompt will be presented (section 35.2). The default filename is `my_ngPS.ps` for a network-graph and `my_jgPS.ps` for a jump-graph. The program reverts to the graph reminders.

20.9 Graph layout file

It is sometimes useful to save a graph layout where time and effort have gone into a particular arrangement such as an unravelled network-graph or a jump-graph designed for redrawing basins at node positions (section 20.7).

Enter **file-f** at the network-graph or jump-graph reminders (sections 20.2.1 and 20.3.2) to save the current graph layout, or load a layout where the number of nodes in the file and current graph must correspond. The following top-right prompt is presented,

graph layout file: save-s load-l:

Enter **s** or **l** to save or load. A filename prompt will be presented (section 35.2). The default filename is `my_grh.grh`. When loading, if the number of nodes in the file and current graph do not correspond, the following top-right message is shown,

can't load: file nodes=10, graph nodes=17: *(for example)*

[5] For example, rename in a terminal, look at the postScript file in "GhostView".

If the file is successfully loaded, the current graph will snap into the new layout. The program reverts to the graph reminders.

20.10 Revise settings

Enter **S** in section 20.4 to change the default settings for the rotation angle, the expand/contract factors for both nodes and links, and the arrow size for links shown as thin lines with arrows as in figure 20.1. The following top-right prompts are presented in sequence,

settings:rotation angle (start=15), now 15.00 degrees:
expand/contract nodes: factor (1 to 2, start=1.2), now 1.20:
expand/contract links: factor (1 to 2, start=1.1), now 1.10:
change arrow size: (0 to .3, start=0.07), now 0.07:

If required, enter the new setting, which may be decimal. The allowed expand/contract factor ranges from 1.0 to 2.0. The arrow size range is from 0 to 0.3, where 0 will suppress arrows entirely, but arrows can also be suppressed and sizes changed from initial graph option in section 20.4 – **arrows-A/</>**. Revised setting become the defaults – **start=** shows the initial defaults.

20.11 Unreachable nodes

Enter **U** in section 20.4 to show up unreachable or hard to reach nodes in a network-graph or basins in a jump-graph by disconnecting and/or isolating them from the rest of the graph. A node/basin is unreachable if it has no incoming links from other basins – in the adjacency-matrix or jump-table (section 20.12) there would be a blank column.

An unreachability threshold of zero is the default, but this can be reset to a higher value, so that a node/basin is isolated if it has x or fewer incoming links. The following top-right prompts are presented in sequence,

links from unreachable nodes: suppress u: restore-(ret):
if **u** *is entered, the following further prompts are presented*
isolate unreachable -i:
change unreachable threshold (now 0):

Enter **u** above to suppress outgoing links to nodes/basins whose incoming links are at or below the current threshold, or **return** to restore these links. If **u** is entered, at the next prompt enter **i** to isolate the unlinked nodes, which will be repositioned randomly on the left side of the window. At the next prompt enter a new threshold if required.

20.12 The adjacency-matrix and jump-table

Enter **t** or **T** in section 20.4 to show a table of outputs from each node. This is called the "adjacency-matrix" for the network-graph, and the "jump-table" for the jump-graph. The table can be presented either according to the number of outputs (enter **t**), or the fraction of the total outputs from each node (enter **T**). The table is presented in the graph window – enter **return** to revert back to the graph.

In an adjacency-matrix (table 20.2), the rows (0 to n-1) show the number (or fraction) of output wires from each network element (cell) to each network element (columns 0 to n-1). Two further columns headed **out** and **k** show the total outputs and inputs from/to each network element.

	0	1	2	3	4	5	6	7	8	9	out	k
0:	1	.	.	.	.	1	.	1	.	.	3	3
1:	1	1	.	.	.	.	1	.	1	1	5	3
2:	.	2	.	1	.	.	.	.	.	.	3	3
3:	.	.	1	1	2	.	1	.	1	.	6	3
4:	.	.	.	.	.	1	1	.	.	.	2	3
5:	1	.	.	.	1	.	.	1	.	.	3	3
6:	.	.	.	.	.	.	.	.	1	.	1	3
7:	.	.	1	.	.	1	.	.	.	.	2	3
8:	.	.	1	1	.	.	.	.	.	.	2	3
9:	.	.	.	.	.	.	.	1	.	2	3	3

Table 20.2: The adjacency-matrix of the RBN $v2k3$, n=10, defined in table 20.1. The rows (0 to n-1) show the number of output wires from each cell to other cells (columns 0 to n-1). This can also be shown as fractions of the total node output. Zero values are show as dots. The last two columns show the total outputs and inputs (k) for each cell.

In a jump-table (table 20.3), successive rows are labeled 1 to N, were N is the number of basins in the basin of attraction field. For each basin, the columns, also labeled 1 to N, show the number (or fraction of total) jumps to each basin. The meaning of the four further columns labeled **P**, **J**, **Volume** and **Self** are as follows,

label . . . *what it means*
P . . . the period of the attractor.
J . . . the total number of possible 1-bit flips or jumps, which equals the network size multiplied by the attractor period, $n \times P$.
Volume . . . the total number of states in the basin of attraction, and its percentage of state-space, for example **110=10.74%**.
Self . . . the percentage of total jumps **J** that are self-jumps.

	1	2	3	4	5	6	7	8	9	P	J	Volume	Self
1:	5	.	.	3	1	.	.	1	.	1	10	110=10.74%	50.00%
2:	.	24	4	8	4	.	.	.	.	4	40	235=22.95%	60.00%
3:	1	5	24	2	.	4	.	.	4	4	40	106=10.35%	60.00%
4:	4	8	.	24	.	.	.	1	3	4	40	233=22.75%	60.00%
5:	1	3	.	.	5	1	.	.	.	1	10	108=10.55%	50.00%
6:	.	.	2	1	1	5	.	1	.	1	10	54=5.27%	50.00%
7:	3	3	.	3	4	.	7	.	.	2	20	18=1.76%	35.00%
8:	1	.	.	1	.	1	.	5	2	1	10	56=5.47%	50.00%
9:	.	1	1	1	.	.	.	1	6	1	10	104=10.16%	60.00%

Table 20.3: The jump-table of the jump-graph in figure 20.12 showing the number of jumps from each basin (rows) to each basin (columns). This example is for the RBN $v2k3$, n=10, defined in table 20.1. Zero values are show as dots.

20.12.1 Printing and scanning tables

When the jump-table or adjacency-matrix is visible, the following top-right options allow the table to be printed in the terminal (not for DOS), and scanned to see all parts of a large table where the data does not all fit inside the window.

scan table: exit-ret xterm-x jump-j reset-s down-d up-u right-r left-l

The meaning of these prompts is listed below,

xterm-x ... *(not for DOS)* to print the table in the terminal.[6]

exit-ret ... exit the table and return to the jump-graph and prompts in section 20.4.

jump-j ... jump to place a given row and column in the top left position of the table. The following top-right prompt is presented,

jump: row: column:

reset-s ... redraw the default table starting with the first row and column.

down-d ... move down, increase the first row index.

up-u ... move up, decrease the first row index.

right-r ... move right, increase the first column index.

left-l ... move left, decrease the first column index.

[6] Displaying the table in the terminal (together with the graph) can also be selected from sections 20.3.1 and 20.2.

Chapter 21

The Seed or initial state

To run a network forwards, or to generate a single basin or subtree – (i.e. if **s** was selected in section 6.2.2), or if in TFO-mode, an initial state or "seed" is required. This chapter describes how the seed is specified and amended, its presentation, saving and loading, and vector PostScript. Note that a seed is not required to generate a basin of attraction field.

Section 10.6 described how the network is indexed and arranged in a regular 1d, 2d or 3d lattice (see figures 10.6 - 10.8). The seed is indexed in the same way, as illustrated in figures 21.1 - 21.3, and these figures indicate the presentation while setting the seed at random in sectin 21.3 or as bits/values in section 21.4. Specifying a seed will assign a value from 0 to v-1 to each cell in the lattice, colored according to the default colors (chapter 7). The methods are similar to "setting a single rule" in chapter 16.

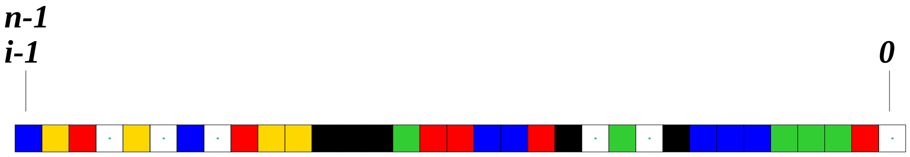


Figure 21.1: A 1d seed (initial state) is indexed as in network indexing (section 10.6.1). This example is for a random seed, n=33, v=6.

21.1 The seed prompt

To set the seed a lower left window is displayed in the main prompt sequence. When reseting a seed while generating space-time patterns or basins, the window appears in the lower centre-right. The size of the window varies according to previous parameters, but can be resized. The exact wording of the prompt depends on the context (see examples below). In general, when first setting the seed the default is **rnd-r** – useful for a random seed or block with a given density (section 21.3). If a seed was previously set the default is **bits*-b** – useful for amending or drawing the seed (section 21.4).

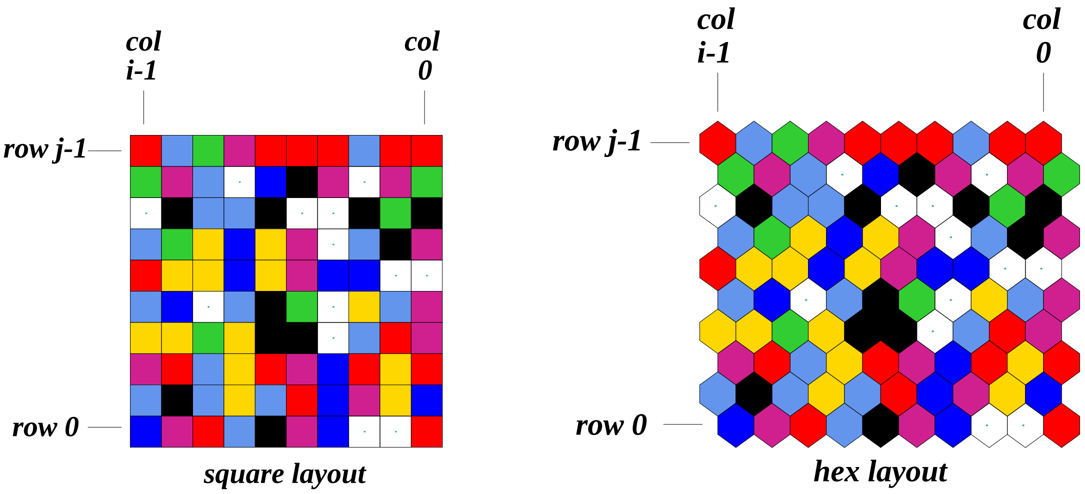


Figure 21.2: A 2d seed (initial state) is indexed as in network indexing (section 10.6.2), showing both square and hex layouts. This example is for a random seed $[i, j]$=[20,20], v=8.

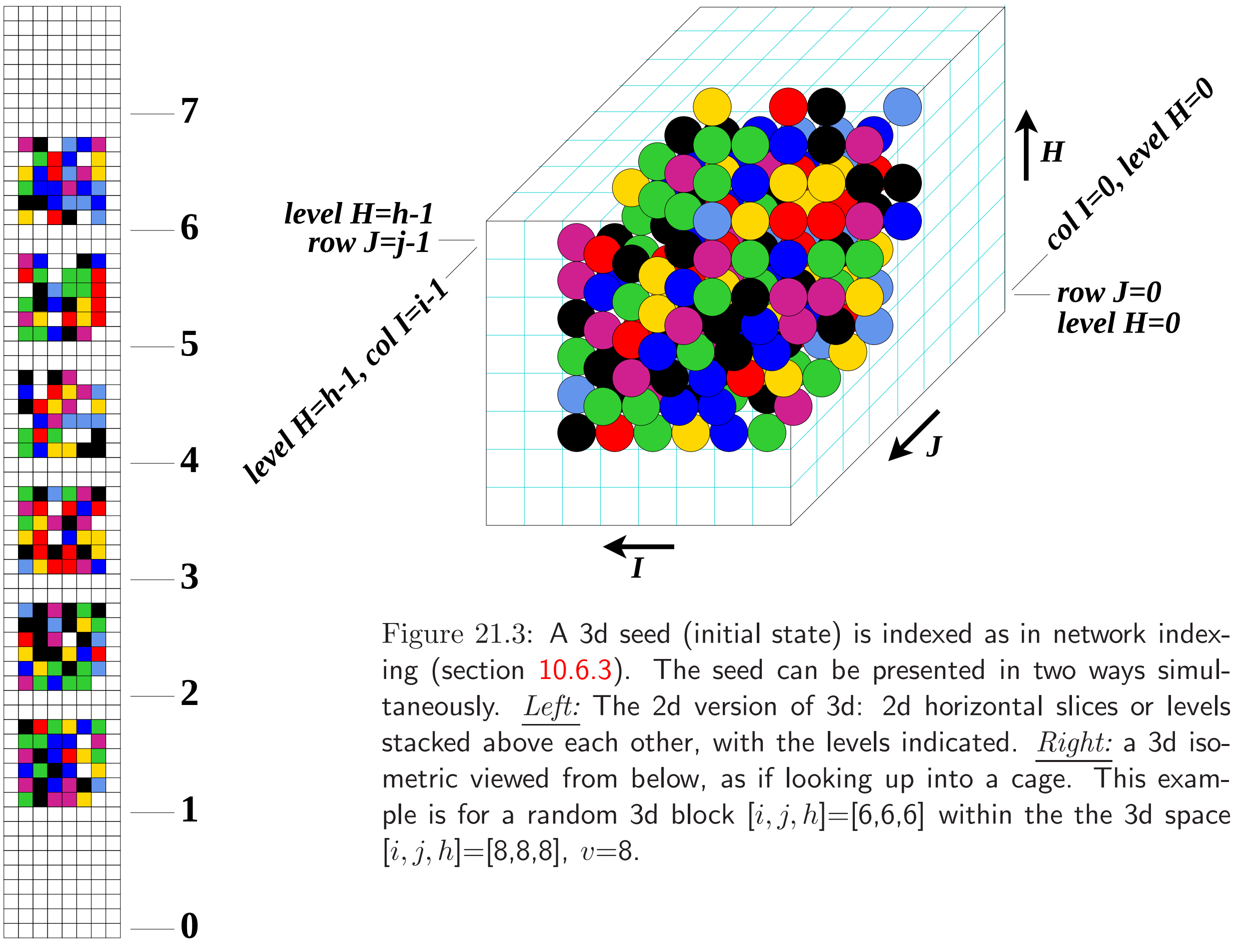


Figure 21.3: A 3d seed (initial state) is indexed as in network indexing (section 10.6.3). The seed can be presented in two ways simultaneously. *Left:* The 2d version of 3d: 2d horizontal slices or levels stacked above each other, with the levels indicated. *Right:* a 3d isometric viewed from below, as if looking up into a cage. This example is for a random 3d block $[i, j, h]$=[6,6,6] within the the 3d space $[i, j, h]$=[8,8,8], v=8.

example for a 1d network
Select SEED (v6 1d n=33), win-w empty-e fill-f prtx-x rnd-r bits1d-b bits2d-B hex-h dec-d repeat-p load-l (def-r):

example for a 2d network
Select SEED (v8 ij=20,20), win-w empty-e fill-f prtx-x rnd-r bits2d-b hex-h repeat-p load-l (def-r):

example for a 3d network
Select SEED (v8 3d ijh=8,8,8), win-w empty-e fill-f prtx-x rnd-r bits2d+3d-b hex-h repeat-p load-l (def-r):

21.2 Methods for setting a seed

The meaning of the prompts for setting a seed[1] in section 21.1 are summarised below. More details are given in the rest of this chapter.

options . . . *what they mean*

win-w . . . to resize the seed window, which may be too small for effectively setting the seed in bits or hex, subsequent prompts are presented, for example,

change window size (max-m), width (now 480): height (now 550):

Enter the new window width and height (minimum 50 × 50), or enter **m** for the maximum in each case to fit the screen.

empty-e . . . to reset all cells to 0.

fill-f . . . to fill the seed with any valid value with the top-right prompt,

fill with value 0-4 (def 4): *(if v=4)*

This allows a clean slate for setting bits or values in section 21.4.

prtx-x. . . to show the seed in the terminal, both in hexadecimal and as a string of values.

note . . . *After the prompts above the program reverts to the seed prompt in section 21.1. The prompts below set a seed and the program comtinues.*

dec-d . . . *(if applicable)* to set the seed in decimal (section 21.6).

rnd-r . . . to set the seed at random, for a given density of a central block – which is displayed in the density window (section 21.3).

[1] These methods are similar to setting a rule in section 16.2.

bits*-b . . . to set the seed as bits or values (section 21.4) by "drawing" using the mouse or keyboard as follows,
bits1d: on a 1d graphic array for 1d networks. Subsequent prompts allows just a segment to be set (section 21.4.5).
bits2d: on a 2d graphic array for 2d networks.
bits2d+3d: on a 2d graphic version of the 3d array (shown simultaneously) for 3d networks, as in figure 21.3.

bits2d-B . . . *(for 1d networks)* to set the seed as bits or values as above, but on a 2d graphic. A subsequent prompt allows changing i, j (section 21.4.6).

hex-h . . . to set the seed in hexadecimal, in a mini "spread sheet" (section 21.5).

dec-d . . . *(if applicable)* to set the seed in decimal (section 21.6).

repeat-p . . . a repeat of the last seed that was set in the current session..

load-l . . . loaded from a `.eed` file (section 21.7).

Enter one of the responses above at the prompt in 21.1 or accept the default. Some of these methods are described in more detail in the rest of this chapter.

21.3 Setting the seed at random

If **r** is selected in section 21.1, a seed, or just a a central block, which applies if $i \geq 20$ for 1d or 2d, and $i \geq 5$ for 3d, can be set at random, with the ability to tune the density bias of non-zero values in the seed or just the block. If applicable, a subsequent prompt is first presented to set the block size,

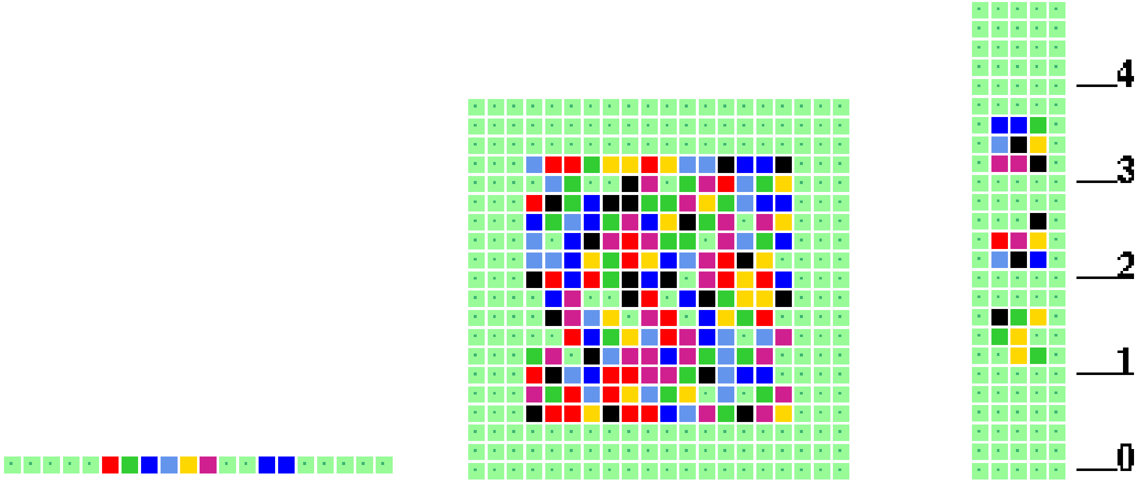
central 1d block (def diam 4), all-a (max size 18): *(for a 1d network, n=20)*
central 2d block (def diam 4), all-a (max size 18): *(for a 2d network, 20×20)*
central 3d block (def diam 1), all-a (max size 3): *(for a 3d network, 5×5×5)*

The default block size is about $i/5$ for for 1d or 2d, and $i/3$ for 3d. Enter the size of the random block or accept the default, or enter **a** for a random seed across the whole network. The pattern is created at random according to the current density bias which appears in a top-right window – initially there is an equal probability of each value, unless this was reset in section 14.1.1. The seed is displayed as a bit/value pattern in 1d or 2d as in figure 21.4. The seed is also shown in decimal (if applicable) (see section 21.6, and in hex (as much as will fit).

Various changes and transformations[2] can be made – repeatedly until **return** or **q** is entered – to accept the seed. The following subsequent reminder is shown,

rotate-l/r another-n density-s comp-c goback-q accept-ret *(for 1d)*
or
rotate-l/r/u/d another-n density-s comp-c goback-q accept-ret *(2d and 3d)*

[2] These option are similar to setting a random initial state or seed in section 16.3.



1d n=20 – block=10 2d 20×20 – block=14×14 3d 5×5×5 – block=3×3×3

Figure 21.4: Examples of the graphic output when setting a seed with a central random block, for 1d, 2d, and 3d. Note that this presentation is less versatile than from "Setting the seed as bits or values" (section 21.4).

21.3.1 Further random seed options

options . . . *what they mean*

tog gaps-g . . . to toggle gaps between sucessive blocks of 8 bits/values.

rotate-l/r/u/d . . . enter **l** or **r** to rotate the seed left or right by one cell, or **u** or **d** to rotate the seed up or down by one cell – for 2d and 3d only. Rotation is subject to periodic boundaries, a ring of cells in 1d, a torus in 2d.

another-n . . . for another random seed or block with the same density bias.

density-s . . . to change the density bias (section 16.3.1).

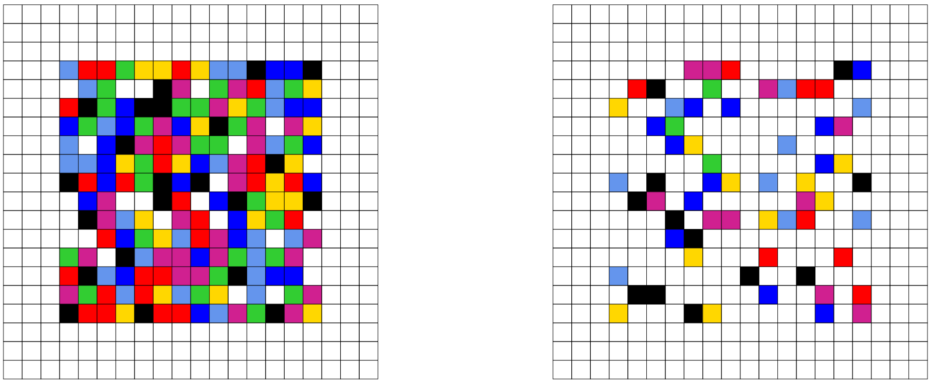
comp-m . . . to toggle/transform the seed into its compliment. For binary this changes 1s to 0s and vice versa. For $v > 2$ each value a is changed to its compliment $a_c = (v - 1) - a$ (see section 16.20).

back-q . . . to backtrack to the first seed prompt (section (21.1) – the latest seed is remembered, and can be amended with **bits-b** or **hex-h**.

accept-ret . . . to accept the rule. Once accepted, the seed can be saved in section 21.8.

21.3.2 Non-zero seed density bias

The density of non-zero values in the random seed or block (expressed as a percentage) is displayed in a lower top-right density window. The default is an equal probability of each value, so 50% 1s for binary. This example is for v=8, a 2d seed $[i, j]$=[20,20], and a block [14,14],



default density bias=87.245%

reset density bias=33%

Figure 21.5: Setting the v=8 density bias for a central block 14×14 within a 2d seed $[i,j]$=[20,20], showing the default and reset density. The graphic output would appear as in figure 21.4, but is presented here from "Setting the seed as bits or values" (section 21.4).

(the density window)

```
density (7-1s)exact=171/400=42.750%, block=171/196=87.245%, bias-87.500%
```

where "density (7-1s)" indicates the non-zero values, "exact" the current method of setting the density as opposed to "prob", then the actual density of the seed and the block (if defined), and the default bias or the bias requested. The bias applies to a block if defined in section 21.3, otherwise to whole seed.

Enter **s** in section 21.3.1 to change the density-bias. There are two alternative methods, **exact** or **prob** (**exact** is the initial default). The following two stage top-right prompt is presented, for example,

(for v=2, 22% entered, with **prob** *as the default method)*
bias-density: enter % (def 50.000% prob):22 exact-e: *(enter* **e** *to change to* **exact***)*

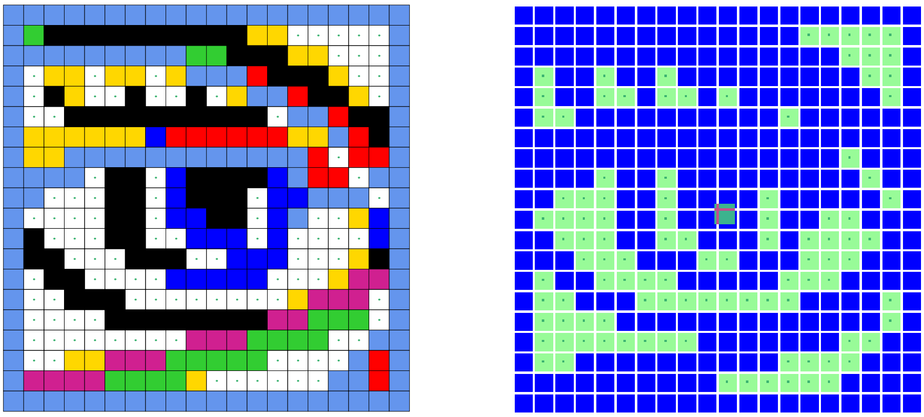
(for v=8, 33% entered, with **exact** *as the default method)*
bias-density: enter % (def 87.500% exact):33 prob-p: *(enter* **p** *to change to* **prob***)*

Enter the new density-bias as a percentage – the result updates the density window – this example relates to figure 21.5.

(the density window)

```
density (7-1s)exact=64/400=16.000%, block=64/196=32.653%, bias=33.000%
```

For binary this is the probability of setting 1s. For v¿2 it is the probability of setting non-zero values – set without bias. The new density-bias becomes the new default. If the **prob** method is active the density bias requested is applied as a probability, whereas the **exact** method will apply the requested bias as closely as possible. To change the default method between **prob** to **exact**, enter **e** or **p** as indicated. The density-bias and method will be maintained for further random seeds or blocks generated with **another-n** in section 21.3.1, updating the density window.



v=8 revised presentation

v=2 default presentation

Figure 21.6: Drawing bits or values on a 2d seed [i, j]=[20,20]. *left*: The v=8 eye was drawn with the mouse and keyboard. *right*: the default presentation of the eye for v=2; light green 0s and white divisions; the flashing cursor was moved to the centre. The v=8 eye was loaded into a v=2 base seed.

21.4 Setting the seed as bits or values

If **b** or **B** is selected in section 21.1 the seed can be set as bits/values in 1d, 2d or 2d+3d. 1d networks can be displayed in both 1d (enter **b**) and 2d (enter **B**). Bits/values are set with the keyboard or drawn with the mouse on the seed graphic, which could be 1d, 2d, or 3d, similar to figures 21.1 - 21.3. The default scale of the bit/value pattern varies according to the dimension and size of the network, but can be changed. The current position on the grid is indicated by a small flashing cursor (initially top left as in figure 21.6(*left*) and also in a top-right inset window.

The colors correspond to the value colors (chapter 7) but 0s are initially colored light green. If a rule is already current, it can be reset to all-0s with **empty-e** or to any uniform value with **fill-f** insection 21.1 to provide a clean slate for setting bits/values. Alternatively, use this bit/value setting method for fine adjustments to a seed set by other methods.

21.4.1 Seed: bits/values reminder window

Bits or values in the seed are set with the mouse and keyboard. Similar methods apply for the rule in section 16.4. During the procedure, a top-right reminder window, and inset, display reminders of various options and current settings (summarised below) which are context dependent and differ between 1d, 2d and 3d seeds, for example,

1d seed, v=2, n=150

the inset while drawing with the mouse → left button: draw 1s width=1

the inset with current settings ↘

keys: set/select/draw value 1-0, vert-v count=0 1d I=149, scale=5 rot=1
mouse: move-click draw-drag width-w move-arrows, PScript-P file-F
tog: gaps/0color/dots/divs/divcolor-g/∧/./i/! exp/contr-e/c
rotate-l/r/+/- flip-h comp-m back/accept-q/ret

2d seed, v=3, 2d seed i, j=20×20

keys: set/select/draw value 2-0, vert-v count=0 2d IJ=19x19, scale=5 rot=1
mouse: move-click draw-drag width-w move-arrows, PScript-P file-F
tog: gaps/0color/dots/divs/divcolor-g/∧/./i/! U/D exp/contr-e/c
rotate-l/r/u/d/+/- spin-s flip-h/V comp-m toghex-x back/accept-q/ret

3d seed, v=8, i, j, h=15×15×15

keys: set/select/draw value 7-0, vert-v count=0 3d IJH=14x14x14, scale=5 rot=1
mouse: move-click draw-drag width-w move-arrows, PScript-P file-F
tog: gaps/0color/dots/divs/divcolor-g/∧/./i/! U/D exp/contr-e/c/E/C
rotate-l/r/b/f/u/d/+/- spin-s flip-h/V comp-m update3d-t back/accept-q/ret

21.4.2 Seed: bits/values options summary

options . . . *what they mean*

keys: select/draw 7-0 . . . enter a valid number to select the value/color, and keep the key pressed to draw a horizontal line.

vert-v. . . enter **v** for a vertical line, and keep **v** pressed to draw a vertical line downwards in the selected value/color.

mouse: move-click . . . left click on the rule pattern to reposition the small cursor and activate drawing – sometimes the right button also needs to be clicked (or right-left a few times) to activate.

draw-drag . . . draw the selected value/color by dragging the cursor with the left mouse button pressed – release the button to stop. The right mouse button draws the complimentary value/color (section 16.20) in the same way.

width-w . . . enter **w** to change the width (initially 1) of the line to be drawn, where the max width is the number of rows for 1d, or the smallest axis for 2d, including 1d shown as 2d, and the 2d version of 3d. The following top-right prompt is presented (for example),

reset line width (now 1) max 15:

The width is shown in the drawing inset, and stays for the current drawing session until revised (figure 16.3 for rule gives examples).

move-arrows . . . all four arrow keys move the cursor around the seed (up/down arrows for multilpe rows).

PScript-P . . . to save the seed as a vector PostScript image (section 21.4.7)

file-F . . . to load/save a seed (section 21.4.7).

tog: gaps-g . . . enter **g** to toggle gaps between sucessive blocks of 8 bits/values (as in figure 16.2 for rules) .

tog: 0color-∧ . . . to toggle the zero value color between light green and white (figure 21.6).

tog: dot-. . . . enter a dot to toggle a dot at the centre of each zero cell (figure 21.8).

tog: divs/divcolor-i/! . . . enter **i** to toggle division lines between the cells. Enter **!** to toggle the division line color between white and black (figure 21.8).

U/D . . . *(for 2d and 3d only)* to pan a 2d seed graphic up and down. 2d will pan by 1 row. A 2d version of 3d (see figure 21.3) will pan by 1 level. This allows different parts of large seeds to be brought into focus for setting bits.

exp/contrect-e/c/E/C . . . enter **e** to expand, **c** to contract the scale by one pixel for 1d and 2d. Similarly, for 3d **E/C** for just the 3d presentation, and **e/c** for the 2d version of 3d, which are treated separately.

rotate-l/r/b/f/u/d/+/- . . . The seed can be "rotated" about the periodic axes in 1d, 2d and 3d by one cell. **l/r** rotates left and right on the i axis, For 2d **u/d** rotates up and down on the j axis. For 3d **u/d** rotates levels up and down on the h axis, and **b/f** rotates backwards and forwards on the j axis. Enter **+** or **-** to increace or decreace the rotation interval – shown in the current settings inset.

spin-s . . . *(for 2d and 3d only)* For 2d where $i = j$ the seed is spun clockwise by 90° so 4 spins returns to the start. Similarly for 3d where $i = j$, each level h is spun clockwise by 90°. If $i \neq j$ a square part including $[I, J]$=[0,0[will be spun. For 2d hexagonal layout there may be some distortion.

flip-h/V . . . enter **h** for a horizontal flip or reflection. For 1d this reflects the i axis, 2d and 3d each i axis is reflected. For 2d and 3d enter **V** for a vettical flip of each j axis.

comp-m . . . to toggle the seed into its compliment. For binary this swaps 1s and 0s. For $v > 2$ the values are shifted; each output a changes to a_m by subtraction from the maximum value $v - 1$, so $a_m = (v - 1) - a$, and the maximum value changes to zero (section 16.20).

update3d-t . . . *(for 3d only)* to update the the 3d view after drawing bits/values on the 2d version of 3d.

back/accept-q/ret . . . enter **ret** to accept the rule, **q** to backtrack.

21.4.3 Seed: bits/values current settings inset

count=0 3d IJH=14x14x14, scale=5 rot=1 ← *the inset with current settings, for example*

current settings . . . *what they mean*

count= . . . the current cursor position, starting from 0 in the top left.

IJH . . . the current cursor coordinates.

scale= . . . the current scale of cells in pixels, which can be expanded and contracted.

rot= . . . the current interval by which the rule-table can be rotated.

21.4.4 Seed: setting bits/values with the keyboard and mouse

The active position (to be updated) in the seed graphic is highlighted by a small flashing cursor initially in the top left. For 3d, the flashing appears only on the 2d version of 3d. Its position is displayed in the top-right inset window. The flashing cursor is repositioned with the mouse by clicking either the left or right button on the new position[3], or moved with the left/right/up/down arrow keys.

Setting or drawing values on the seed graphic is done with the keyboard or mouse. This applies to the 2d version of 3d – the 3d view can be updated by pressing **t**. To set (and activate) a value at the cursor position, press a valid number key (without return) – the cursor will then advance to the right by one unit. To draw a horizontal line towards the right keep the key pressed. While the key is pressed the line will continue to the next row or jump back to the top left. To draw a vertical line downwards with the active value, press **v**.

To draw the active value with the mouse, drag the cursor anywhere over the seed with the left button pressed – release to stop. To draw the compliment of the active value (see section 16.20) drag with the right button pressed. While a mouse button is pressed, the inset in the reminder window changes to show which button, the current active value, and the current width of the line, for example,

left button: draw 2s width=3 *or* right button: draw 8s width=1

Initially the left button draws the value v-1 and the right draws 0, so for binary 1 and 0. To activate drawing with the mouse it is sometimes necessary to click the left and right button alternately. To accept the seed enter **return**. Once accepted, the rule is also displayed in decimal (if applicable), in hex for small seeds less than 64 bytes.

21.4.5 setting bits/values: 1d segments

for 1d seeds only

For a 1d network, if **b** is selected at the seed prompt (section 21.1), the following prompts allows just a segment of the seed to be defined,

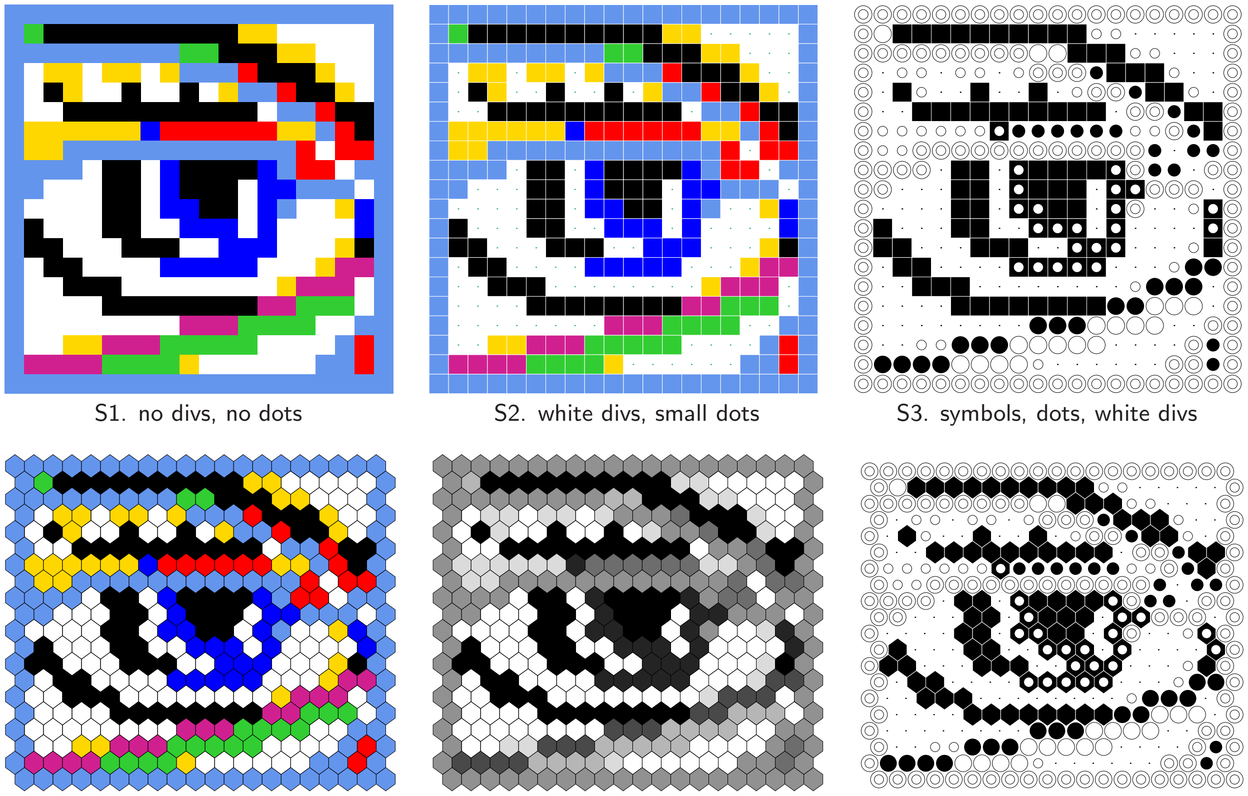
segment: enter length (max/def=200): *(for n=200)*

Enter **return** for the whole seed, or enter the segment size whereupon a further prompt positions the segment within the network,

seglength 5: enter start position from left 0-149, def centred: *(for example)*

Enter **return** for a centered segment, or a position from the left. A bit pattern representing just the segment is displayed for setting bits/values. There is no limit to adding and overwriting segments.

[3] There is a slight difference between Linux-like systems and DOS. In Linux the mouse cursor changes direction within the rule-table pattern, pointing north-west instead of north-east, and within the pattern, left or right mouse clicks reposition the flashing cursor. In DOS the mouse cursor is confined within the rule-table pattern and clicking the left or right buttons sets values as well as repositioning the flashing cursor.



S1. no divs, no dots

S2. white divs, small dots

S3. symbols, dots, white divs

H1. black divs, no dots

H2. greyscale, black divs, no dots

H3. S3. symbols, dots, white divs

Figure 21.7: Examples of PostScipt seed output, 2d seed, the eye as in figure 21.6, $[i,j]$=[20,20], v=8. A variety of options are available: color, greyscale, symbols, hex, square, black or white divisions between cells or none, variable size zero dots or none. *Top row*: square layout. *Bottom row*: hex layout.

21.4.6 setting bits/values: 1d shown as 2d

for 1d seeds only

For a network, 1d, if **B** is selected at the seed prompt (section 21.1), the 1d seed is shown in 2d as a square or rectangular bit/value pattern, $i \times j$. Default j (the number of rows) is set automatically as the highest factor of n, $\leq \sqrt{n}$. The following prompt allows the default i to be revised,

> **now 20 x 10, reset i-axis:** *(for n=200)*

The 1d seed is broken up into successive rows starting with the maximum cell index in the top left. If i is not a factor of n, some cells will be missing from the bottom row. The 2d arrangement can be saved as a 2d PostScript or seed file, and loaded back into another 2d seed file (section 21.4.7).

21.4.7 Setting bits/values: filing and PostScript

The seed can be saved as a vector PostScript image, or as a 1d, 2d or 3d seed file, by selecting *PScript*-**P** or *file*-**F** in section 21.4.1 – both options follow similar the methods, and are similar to the rule options in section 16.4.3. File extentions and default filenames (chapter 35) as follows,

PostScript files . . .	`*.ps` files, default `my_sPS.ps`.
seed files . . .	`*.eed` – default `mysee_vv.eed` where v = value range.

One of the following prompts appears in a top-right window,

for PostScript, PScript-**P**
PostScript 1d: save all-a, save patch-p: *(for 1d)*
PostScript 2d: save all-a, save patch-p: *(for 2d)*
PostScript 3d: save all 2d-a, save patch 2d-p, only 3d-3: *(for 3d)*

for a seed file, file-**F**
SEED: load-l, save all-a, save patch-p: *(for 1d, 2d or 3d)*

for a 1d seed which can be saved as a 2d seed file, file-**F**
SEED: load-l, save1d>2d-x, save all-a, save patch-p:

If *bits2d*-**B** was selected at the 1d seed prompt (section 21.1) to display the seed in 2d, there is an extra option **save1d>2d-x** to save the 1d as a 2d seed. The i, j coordinates selected might give an incomplete rectangle; this is acceptable for both the 2d seed and PostScript.

These options are summarised below,

load-l . . .	load a seed file (section 21.7). As many seed files as required can be loaded into the current seed. Different v ($v_{file} \leq v_{base}$), dimensions and sizes are possible subject to the constraints listed in section 21.7.1.
save-a . . .	save the seed, as a seed file or PostScript. For PostScript and 3d, **save-a** applies to the 2d version of the 3d image, as in figure 21.3.
save patch-p . . .	to save part of the seed as a seed or PostScript file, first defining the limiting coordinates of the patch (in 1d, 2d or 3d) in section 21.4.8.
only 3d-3 . . .	*(for PostScript of 3d only)* save the 3d isometric of the seed (figure 21.3).
save1d>2d-x . . .	*(for 1d seed file)* save a 1d seed as a 2d seed. The seed must be displayed with **bits2d-B** in section 21.1.

A top-right prompt gives various options for the PostScript presentation (section 21.4.9. Files are loaded and saved from the filing prompt (section 35.2).

21.4.8 Setting bits/values: saving a patch

Part only of the seed (1d, 2d, or 3d) can be saved, as either a seed file or as a vector PostScript image file. The limiting coordinates of the patch are set by the last two mouse clicks on the bit/value seed graphic, or by entering coordinates.

If **p** is selected in section 21.4.7, one of the following pairs of prompts are shown in sequence in a top-right window,

example for a 1d network
1d:max i=149, revise coords-ret, accept patch 55-37 -p:
start i: end i:

example for a 2d network

2d:max i,j=39,39, revise coords-ret, accept patch 26,23-14,14 -p: file:i,j=6,6
start i: end i: start j: end j:

example for a 3d network

3d:max i,j,h=8,8,8, revise coords-ret, accept patch 6,6,5-2,1,3 -p:
start i: end i: start j: end j: start h: end h:

Enter **p** to accept the default coordinates according to the last two mouse clicks, or **return** to set the coordinates manually on the second prompt line, which otherwise does not appear. **return** at a manual setting gives the default mouse click coordinates. The patch seed file can then be loaded back into any position within the same or another seed, as described in *Loading a seed*, section 21.7. The facility to save and load a patch is useful to set up special configurations such as glider collisions.

21.4.9 Setting bits/values: PostScript prompt

When creating a PostScript image of a seed, there are a variety of presentation possibilities as illustrated in this chapter (figures 21.1 - 21.10) and other figures in the manual. Enter **a** (or **p** for part of the seed) at the prompt in section 21.4.7 to save the seed as a vector PostScript (`*.ps`) file (default filename `my_sPS.ps`) . The following top-right prompt is displayed, where **direct-d** applies only when interrupting space-time patterns[4], for example,

create a PostScript image for 2d, 20x20 *(or* **1d***,* **3d***)*
symbols-s greyscale-g color-c direct-d exit-q (def-c):

These options are summarised below (then subsequent options continue),

options . . . *what they mean*

symbols-s . . . to show values as symbols (figures 21.7 and 21.8).

greyscale-g . . . to show values in greyscale (figures 21.7).

color-c . . . *(the default)* to show values in color.

direct-d . . . *(only when interrupting space-time patterns)* to scan cell colors directly from a 1d space-time pattern, or 2d, or 3d as 2d snapshot. If **direct-d** is selected the second line is repaced with,

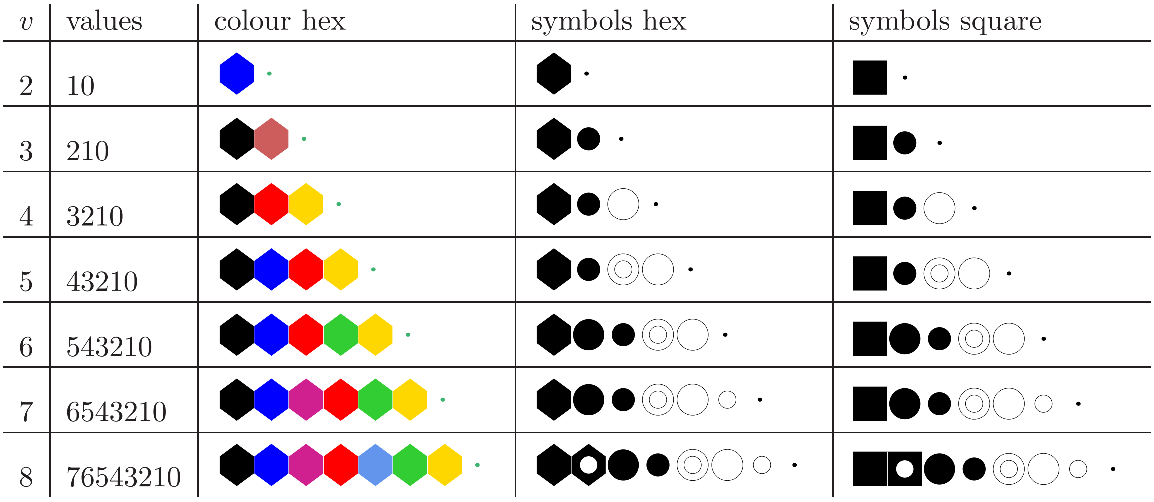
direct: greyscale-g color-c (def-c): *(***symbols-s** *does not apply)*

For interrupted 1d space-time patterns, once the selection is made the next two prompts limit the part of the image to be saved, for example,

1d size (max/def=150): time-steps (max/def=669):

Once these options have been selected, the prompt to amend other setting is presented, for example,

[4] From the space-time pattern pause options (section 32.14) enter **e** to revise the original, last, or current state which gives the seed prompt in section 21.1. Alternatively enter **P** for an immediate PostScript prompt similar to section 21.4.9. For 1d space-time patterns, **P** gives the direct option only.



v	values	colour hex	symbols hex	symbols square
2	10	⬢ ·	⬢ ·	■ ·
3	210	⬢⬢ ·	⬢● ·	■● ·
4	3210	⬢⬢⬢ ·	⬢●○ ·	■●○ ·
5	43210	⬢⬢⬢⬢ ·	⬢●◎○ ·	■●◎○ ·
6	543210	⬢⬢⬢⬢⬢ ·	⬢●•◎○ ·	■●•◎○ ·
7	6543210	⬢⬢⬢⬢⬢⬢ ·	⬢●•◎○∘ ·	■●•◎○∘ ·
8	76543210	⬢⬢⬢⬢⬢⬢⬢ ·	⬢⬢●•◎○∘ ·	■■●•◎○∘ ·

Figure 21.8: Symbols for PostScipt seed output for v = 2 to 8. Enter s in the PostScipt prompt (section ??) to show values as symbols replacing colors. Symbols come in the two versions shown, hex and square, depending on the bits/values display in section 21.4.2 which is toggled with x. The value zero is shown here as a dot, but this can be blank or the dot size changed.

cellscale=5.00 dots(on)=0.70 divs(2), amend settings-a:

Enter **return** to accept the defaults, or enter **a** for the following prompts presented in sequence,

change: cellscale: togdots-x: dotscale: divs(0,1,2):

Enter changes required or **return** to accept defaults, which follow the current bits/values presentation. The options are summarised below,

options ...	*what they mean*
cellscale ...	enter a new cell width in pixels.
togdots-x ...	enter **x:** to toggle zero dots on/off.
dotscale ...	*(if dots are on)* enter a new width for zero dots in pixels, which can be a decimal number.
divs(0,1,2) ...	the initial default division (color) depends on the bits/values presentation, and is shown in the prompt: (**none**, **1**, **2**) where **none** means that there is no division and adjoining cells touch. To change, enter **0** for none, **1** for black, and **2** for white. The new designation becomes the default.

Cells with value zero (0color) are initially colored light green in the bits/values presentation (section 21.4), which can be togged to white with *0color-∧* in section 21.4.1 – the PostScript file will follow the current 0color.

Once these choices are complete, the `*.ps` file is saved from the filing prompt (section 35.2). Section 36.1 explains how to view, edit, and crop the PostScript image.

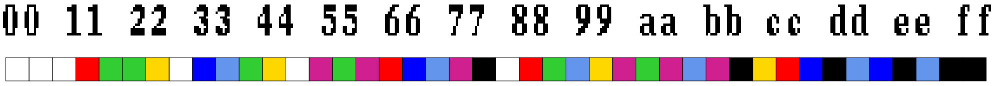
21.5 Setting the seed in hex

If **h** is selected in section 21.1, the seed is defined in hexadecimal (hex) which can be useful for creating repetitive patterns. The method is the same as setting a rule in hex in section 16.5. The hex expression of the current seed (initially just 0s) is displayed. Each hex character (0 – 9 then a – f) shows the value of 4 bits, and the characters are displayed in one byte pairs. During the hex setting procedure, a top-right reminder window displays the following,

enter hex, arrows-move $\boxed{\text{hex count=14}}$ ← *inset shows the current hex position, from the top left*
rotate-l/r, accept-ret

The hex character to be updated is highlighted by a flashing cursor, initially in the top left. Its position is displayed in the top-right inset window. The flashing cursor is moved with the arrow keys, left/right and up/down for more than one hex line. To overwrite, enter a hex character from the keyboard, without **return**. This automatically moves the cursor one position to the right. Hex characters entered which exceed the current value-range will be automatically corrected downwards to the maximum value after the hex string has been accepted. Enter **l** or **r** to rotate the underlying seed by one cell as in sections 21.3 and 21.4, which will be immediately reflected in the hex presentation.

To accept the hex seed enter **return**, whereupon seed will be presented as bits/values displayed in 1d, 2d or 2d+3d (section 21.4), where it can be reconfirmed or amended.



```
00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
```

Figure 21.9: In this example (v=8, n=42) the seed was first set in hex from the possible hex characters (0 – 9 and a – f). The figure shows the result once accepted, where the bits/values option (section 21.4) activates below the hex presentalion, allowing the seed can be reconfirmed or amended.

21.6 Setting the seed in decimal

applicable only for a limited range of v and system size n

The decimal option remains valid as long as the decimal equivalent of the bit/value string representing the seed is less than an unsigned long int (2^{31}-1). The same limits apply to the size of a rule-table S when setting a rule in decimal, and are set out in table 16.1 in section 16.6.

If **d** is an available option and is selected in is selected in *The seed prompt* (section 21.1), the seed can be specified by its decimal equivalent. The following prompt is displayed,

decimal seed, enter 0-4782968 (def-rnd-dec): *(example for n=14, v=3)*

Enter a decimal number, or enter **return** for a random number within the permitted range, which will be displayed. If the number entered is outside the permitted range, the program presents an error message, for example,

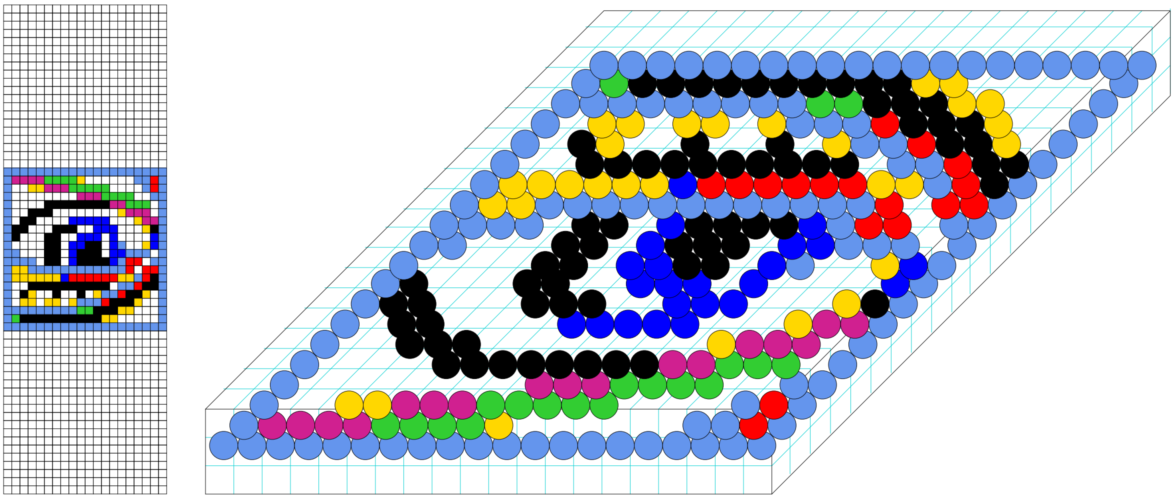


Figure 21.10: The v=8 2d 20×20 "eye" in figure 21.6 was loaded into a 3d seed 20×20×3, then flipped upside-down with **V**. The result is shown as it appears simultaniously in the 2d version of 3d, and in the 3d isometric.

```
decimal seed, enter 0-4782968 (def-rnd):5555555 - too big! cont-ret:
```

Enter **return** to revert to *The seed prompt*. Once successfully selected, the decimal rule is presented again as bits/values (section 16.4), where it can be reconfirmed or amended.

21.7 Loading a seed

Enter **l** in section 21.1, or **F** followed by **l** in section 21.4.7 to load a seed (`.eed` file)[5] After loading (from the top-right filing prompt, section 35.2) DDLab will know the file parameters: value-range v, dimensions 1d, 2d, or 3d, and the size of the coordinates $[i, j, h]$. File-seeds that differ from the base-seed in any of these parameters can still be loaded, but a top-right window warns of different v, or prompts to position a small file-seed (of any dimension) that fits within a larger base-seed. A bigger file-seed dimension is reduced to the base-seed dimension. If any file-coordinate $[i, j, h]$ is bigger than the corresponding base-coordinate, a part of the file is loaded automatically without further prompts. Section 21.7.1 below lists these constraints in more detail.

21.7.1 Constraints for loading a seed

all parameters equal . . . if all file-parameters are the same as all base-parameters, the file is loaded without further ado.

value-range v . . . if file-v is different from base-v, the values loaded will be adapted to correspond to base-v. The following top-right warning is presented, for example)

[5] A seed was saved in sections 21.4.7 or 21.8, or just part of a seed, a patch, in section 21.4.8.

```
file-v(2) != base-v(8), abandon-q, load anyway-ret:
```

Any file-v > base-v will be changed to the maximum base-v. Note that base colors will conform to the base color key (section 7.1).

file-*dim* < base-*dim* ... a smaller file-seed dimension is increases to the base-seed dimension, by adding coordinates of size 1. For file-2d, h is set to 1 for base-3d. For file-1d, j is set to 1 for base-2d, and $[j, h]$ is set to [1,1] for base-3d.

file-*dim* > base-*dim* ... a bigger file-seed dimension is reduced to the base-seed dimension. For file-3d, its 2d version will load correctly if the base-2d has the corresponding $[i, j]$ coordinates. Likewise a file-2d or file-3d will appear correctly in a 1d-base shown in the 2d with corresponding $[i, j]$ coordinates.

file-$i, j, h \leq$ base-i, j, h ... if file-$[i, j, h]$ fits inside base-$[i, j, h]$, then if any file-axis is smaller than the base-axis, a prompt is presented in a top-right window to locate the file-seed within the base[6], for example,

example for a 1d network

```
1d: i=100, file size=20, enter start pos
(def 40, max 80, rnd-r):
```

example for a 2d network

```
2d:i,j=100,100, file:i,j=6,6, enter start coords
(def 47,47, max 94,94, rnd-r) i:    j:
```

example for a 3d network

```
3d:i,j,h=40,40,40, file:i,j,h=15,15,15, enter start coords
(def 12,12,12 max 25,25,25, rnd-r) i:    j:    h:
```

Enter the coordinates in the base-seed to locate file-seed's lower right cell, **return** for the default central position, or **r** for a random coordinate. Any number of file seeds can be loaded.

file-$i, j, h >$ base-i, j, h ... for a 2d or 3d base, if any file-coordinate $[i, j, h]$ is bigger than the corresponding base-coordinate, the larger coordinate is automatically centred on the smaller base – any smaller axis starts at index 0. For a 1d base the file-seed starts at index 0.

21.8 Saving a seed

Once the seed has been accepted, a top-right window shows the density of non-zero bits/values (see section 21.3.2) and gives another chance to revise or save the seed,

```
density 3-1s=303/400=75.750%, bias=75.000%
SEED: revise-q save-s cont-ret
```

(example for v=4)

[6] This is similar to loading a sub-network in section 19.4.3.

Enter **q** to revise, or **s** to save the seed (as a `.eed` file) from the top-right filing prompt (section 35.2). If just **return** is entered, the seed will be saved automatically with the default filename `mysee_v`*v*`.eed` where v is the value range. The seed could also have been saved from the reminder window when setting bits/values (section 21.4, enter **F** followed by **s**).

21.9 Seed file encoding

The `.eed` seed file is encoded[7] starting 5 leading bytes as follows, where n is the network size, (i, j) the axes of a 2d network, and (i, j, h) the axes of a 3d network,

byte 0 . . . value-range v.

byte 1,2 . . . network size, n.

byte 3,4 . . . $[i,\ j]$ which gives the dimensions and axis sizes, If $i = 0$ and $j = 0$ the seed is 1d. If both $i > 1$ and $j > 1$ the seed can be either 2d or 3d. $h = n/(i \times j)$. If $h > 1$ the seed is 3d, otherwise 2d.

The rest of the file consists of the seed values from 0 to $n-1$, where the number of bits required for each cell (V_{bits}) is as follows:

v=2 . . .	1 bit ($V_{bits} = 1$)
v= 3 or 4 . . .	2 bits ($V_{bits} = 2$)
v= 5, 6, 7, 8 . . .	3 bits ($V_{bits} = 3$)

A seed encoding requires 5 leading bytes plus the bytes for the seed itself R, where $R = \left\lceil \frac{n \times V_{bits}}{8} \right\rceil$ bytes, the upper absolute value, minimum 1 byte.

[7] Seed files saved in this multi-value DDLab are not compatible with the old binary DDLab, and vice-versa. For single rule encoding see section 16.16 and for wiring+rulemix encoding see section 19.3.

Chapter 22

The Derrida plot

Once the rule or rules have been set, the Derrida plot (and Derrida coefficient) can be selected from the wiring graphic in section 17.4 – enter *Derrida*-**D**. Any type of network may be plotted, a CA, RBN or DDN, in 1d, 2d, or 3d, in normal and TFO mode.

The main application of the Derrida plot has been as an order-chaos measure for large RBN networks in the context of models of genetic regulatory networks[9, 12], where the canalizing inputs can be tuned to move the dynamics between order and chaos (see section 15). The Derrida plot provides a statistical measure of the divergence/convergence of network dynamics in terms of "Hamming distance" (H). H between two binary states of equal size, n, is the number of bits that differ. The normalised Hamming distance is H/n. For multi-value, H is the number of values that differ, but exactly the same principles apply.

The Derrida plot is generated as follows

1. Randomly select a pair of initial states, I_1, I_2, separated by a small Hamming distance of H_0, at time-step t_0.
2. Iterate each state forward independently, according to the network architecture, by usually one, or more time-steps, i (i must be 1 to compute the Derrida coefficient).
3. Measure the Hamming distance, H_i, at time-step i, between the pair of final states, F_1, F_2.
4. Repeat the above for a sample of pairs of initial states with the same initial Hamming distance H_0, and plot normalized H_0 (x-axis) against the mean normalized value of the H_i's, (y-axis).
5. Repeat the whole procedure for a range of increasing initial Hamming distances. The increase (Ham steps) must be 1 to compute the Derrida coefficient, but may be set to larger intervals otherwise.

22.1 Selecting the Derrida plot

To select the Derrida plot, first display the network as a wiring graphic aftr rules are set (section 17.1). One of the top-right wiring graphic options (section 17.4) is the following,

... Derrida plot-D:

Enter **D** to select the Derrida plot.

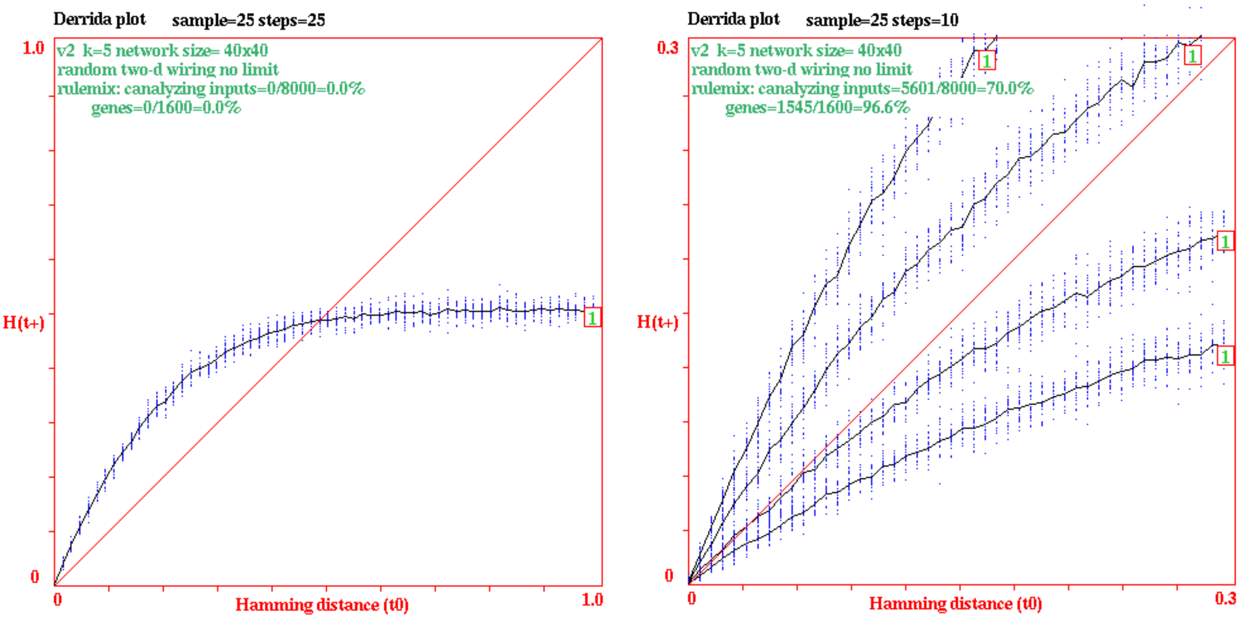


Figure 22.1: Examples of Derrida plots for a 2d random Boolean network, $k = 5$, $n = 40 \times 40$. Data about the network are shown in the top left hand corner of the graph (section 22.3). Derrida data is shown at the top of the graph (parameters: sample and steps). The Derrida coefficient (section 22.6) would also be shown, but is not in this case because steps>1. The number of iterations is shown at the end of each curve.
left: Max$H$=1, showing the full range of Hamming distance, sample=25, steps=25.
right: 4 superimposed plots for increasing Canalizing settings (0%, 20%, 50% and 70%), giving progressivly lower slopes. MaxH=0.3, so zooming in for the most significant first part of the graph, sample=25, steps=10. Canalizing can be changed before a new plot, and previous plots kept.

22.2 Derrida plot options

If **D** is selected in section 22 above, the following series of options are presented in sequence in a top right window to change the Derrida plot parameters or keep the defaults. The revised parameters generally become the new defaults,

Derrida plot: quit-q, Dc sample (now 15, max 320): *(for a* 40×40 *RBN)*
followed by parameter prompts as follows (values shown are examples)
maxH (def 1.00): sample (def 25) tog spread (ON)-s: details-d
iterations (def 1): Ham steps (def 1 net=1600): keep-k:

These parameters are described in section 22.2.1 below.

22.2.1 Derrida plot parameters

The meaning of the parameters in the Derrida plot options (section 22.2) are given below, where values in are brackets initial defaults or examples.

parameters . . . *what they mean*

quit-q . . . quit the Derrida plot.

Dc sample (now 15, max 320): . . . the number of inital points that will form the basis of the Derrida coefficient, derived from the initial slope (see section 22.6). The initial default depends on the size of the network, but is 15 for large networks.. Generaly, between 10 and 20 points are appropriate. Enter **return** to accept the default or enter a new value which becomes the new default.

maxH (def 1.00) . . . the maximum normalized Hamming distance of the plot defining the extent of the xy axis (maximum 1). A small $maxH$ gives a detail of just the left hand corner of the plot, which is perhaps the most significant. Only a small initial part of the plot is required to calculate the Derrida coefficient, enough to contain the **Dc sample** (section 22.6). Enter **return** to accept the default (initially 1) or enter a new decimal value ($0 < maxH \leq 1$) which becomes the new default.

sample (def 25) . . . the sample size for each initial Hamming distance. A smaller sample gives a quicker plot. A larger sample provides a more accurate measure. Enter **return** to accept the default (initially 25) or enter a new value which becomes the new default.

tog spread (ON)-s: *or* **(OFF)** . . . to toggle plotting the spread of each sample. Enter **return** to accept the default (initially **(ON)**) or enter **s** to toggle – this becomes the new default.

details-d . . . enter **d** to pause at each sample pair and show detailed data in a top-right window (mainly for debugging and diagnostic purposes), for example,

sample=4 H(t0)=21 H(t1)=16 q-quit: *(for example)*

In this example, at the 4th sample point, the initial Hamming distance is $H_{t0} = 21$ (not normalized), the final Hamming distance, is H_{t1}=16, at time-step 1. For system size $n \leq 40$ the bit/value pattern of the initial and final pairs of states are also shown above the prompt. Enter **return** at the prompt above for the next sample pair, or **q** to disable the pause and continue without interruption.

iterations (def 1) . . . the number of forward time-steps required. Enter **return** to accept the default (initially 1) or enter a new value which becomes the new default. Iterations must be 1 to compute the Derrida coefficient.

Ham steps (def 1 net=1600) . . . the interval between successive initial Hamming distances (not normalised). The minimum and default is 1. A small interval provides a more accurate plot. A larger interval provides a quick and sketchy plot. Enter **return** to accept the default (initially 1) or enter a new value which becomes the new default. The system size n is shown as a reminder as it will be subdivided by the number of Ham steps. Ham steps must be 1 to compute the Derrida coefficient.

keep-k . . . enter **k** to keep a previous plot and superimpose a new plot (as in figure 22.1 *right*). The new plot may have revised network parameters.

22.3 Data within the Derrida plot

Within the Derrida plot, some data is shown relating to both the network, and to Derrida parameters, as in figures 22.1 and 22.2, described below.

22.3.1 Network data

The top left hand corner of the graph will show context dependent information as follows.

vv . . .	the value-range, for example **v2**.
k=k . . .	for homogenious k.
k-mix=k_1**-**k_2 . . .	the range of k for a k-mix .
network size=n . . .	for a 1d network.
network size=$i \times j$ . . .	for a 2d network.
network size=$i \times j \times h$ . . .	for a 3d network.
one-d CA wiring . . .	for regular 1d CA wiring.
two-d CA wiring . . .	for regular 2d CA wiring.
three-d CA wiring . . .	for regular 3d CA wiring.
random one-d wiring . . .	for random 1d wiring.
random two-d wiring . . .	for random 2d wiring.
random three-d wiring . . .	for random 3d wiring.
no limit . . .	random wiring is not restricted.
zone=r . . .	random wiring is restricted within a radius r.
homogeneous CA rule . . .	for a network with a single rule.
rulemix: canalizing . . .	*for a rulemix*
inputs=c/c_{max}**=**c**%** . . .	the percentage of canalizing inputs.
genes=g/n**=**g**%** . . .	the percentage of canalizing genes.

22.3.2 Derrida data

Some Derrida data is shown at the top of the graph, for example,

Derrida plot sample=55 slope=46.4deg Dc=0.072 (15 H)

Sample is explained in section 22.2.1. **Slope**, **Dc** and **(15 H)** relate to the Derrida coefficient, described in section 22.6, and appear only if **iterations=1** and **Ham steps=1** in section 22.2. The number or **iterations** are shown at the end the curve.

22.4 Interrupting the Derrida plot

While the Derrida plot is in progress, the following top-right message is displayed,

> **plotting Derrida, sample=25, iteration=1 step=1**
> **quit/pause-q** *(values shown are examples)*

If the plot is interrupted with **q**, the following top-right prompt is presented,

> **interrupted at Hamm=0.297, quit now-q, cont-ret**:
> *enter* **q** *to stop the plot at the point reached – the next prompt is in section 22.5*
> *enter* **return** *to continue the plot, but these parameters (as in section 22.2) can be reset first*
> **sample (now 25): tog spread(ON): details-d** *(values shown are examples)*
> **iterations (now 1): Ham steps (now=1 net=1600): keep-k:**

22.5 Completing the Derrida plot

When the Derrida plot is complete, or if interuppted and stopped in section 22.4, the number of iterations is displayed at the end of the curve, and the following prompts are displayed in a top-right window,

> **Derrida plot complete)**
> **reset-r canalizing-C:**

Enter **r** to reset the plot. The options in section 22.2 are presented, but with an additional option **redraw (no spread)-r** – for a quick redraw of the plot from memory without the sample spread. Enter **C** in section to revise the network's canalising settings. For a mixed rule network see section 15. For a single rule network, see section 18.5. Previous plots can be retained (enter **k** as in section 22.2) to produce a multiple plot as in figures 22.1(*right*) or 22.2.

22.6 The Derrida coefficient

The Derrida coefficient (Dc), analogous to the Liapunov exponent in continuous dynamical systems, is derived from the initial slope of the Derrida plot, a tangent to the curve at the origin, which indicates whether the network dynamics is convergent (ordered) or divergent (chaotic) and to what extent.

A slope of 45° ($Dc = 0$) indicates the dynamics is ballanced between order and chaos. A slope above the 45° diagonal (positive Dc) is in the chaotic regime and correlates with increasing chaos. A slope below 45° (negative Dc) in the ordered regime and correlates increasing order. The initial slope is based on the first few points of the Derrida plot. The Derrida coefficient is only calculated if **iterations=1** and **Ham steps=1** in section 22.2.

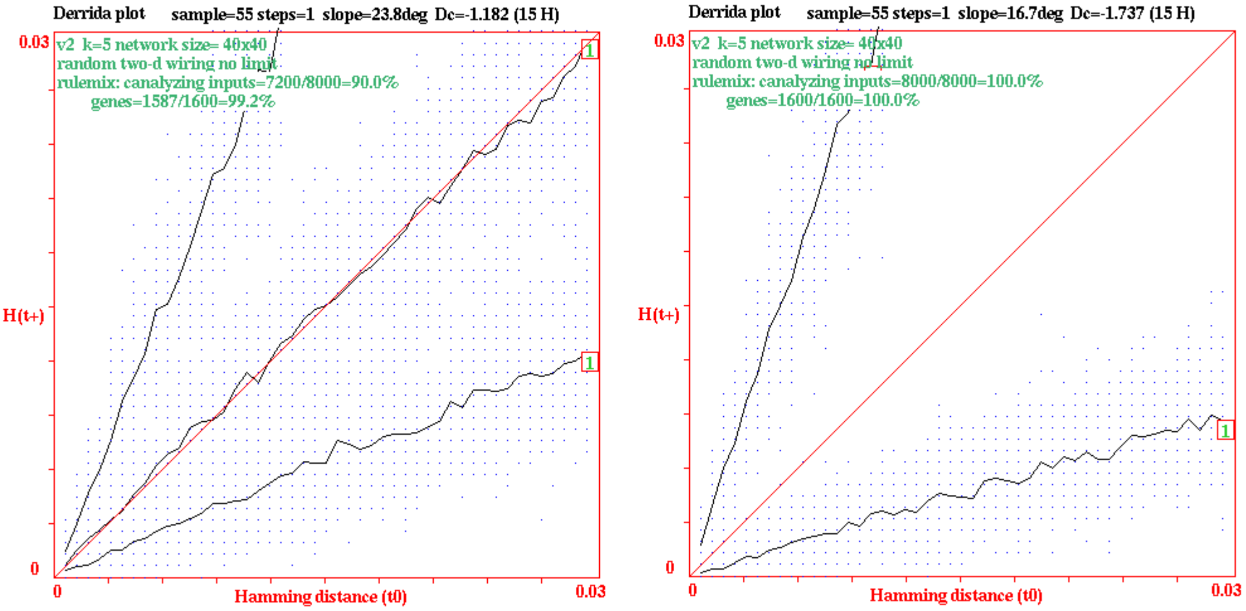


Figure 22.2: Examples of Derrida plots and their Derrida coefficients (Dc) for a 2d RBN, k=5, n=40×40 with various canalising, settings $C\%$, changed in section 22.5 for a range of behaviour. Other parameters were as follows: Dc sample=15, MaxH=0.03, sample=55, iterations=1, steps=1. The avarage slope, $\delta°$, and the Derrida coefficient, Dc, for the plots are given below.

left: 3 superimposed plots:chaos/balance/order.

$C\%$	slope $\delta°$	Dc
0	67.7°	1.288
51	46.4°	0.072
90	23.8°	-1.182

right: 2 superimposed plots:extreme chaos (chain-rules) and order.

$C\%$	slope $\delta°$	Dc
0	71.4°	1.572
100	16.7°	-1.737

The number of points to determine the slope at the origin (the Dc sample) is selected to lie approximately on a straight line – the default is 15 for large networks. The average slope between these points and the origin is taken as the initial slope. If the initial slope = $\delta°$, then the Derrida coefficient, $Dc = log_2(tan(\delta°))$.

Figure 22.2 gives examples for a k=5 RBN, with varying degrees of canalising (section 15) to demonstrate a range of behaviour.

Chapter 23

Graphic conventions for attractor basins

Figure 23.1 explains the idea of basins of attraction in discrete dynamical networks. Attractor basins are represented by state transition graphs, where nodes – network states, are linked by directed edges – state transitions. States in deterministic networks have one successor but possibly a number of predecessors (pre-images), so trajectories occur within trees. In a finite network a trajectory must encounter a repeat state, defining its attractor. Attractor basins are thus made up of transient trees rooted on attractor cycles.

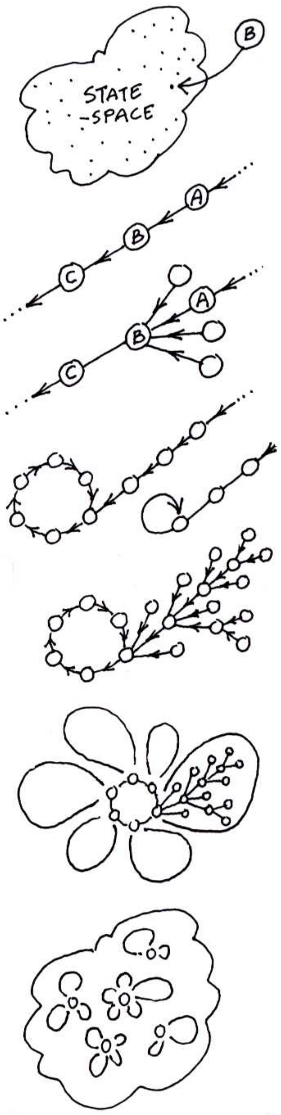
The attractor cycle may have an arbitrary period, including a period of just one, a point attractor cycling to itself. Transient trees may consist of just one node outside the attractor, or a number of nodes with arbitrary in-degree (but one out degree) at each node. Nodes with zero in-degree, the leaves of the trees, are the so called garden-of-Eden states. The graphic conventions for drawing attractor basins and subtrees are described below, and illustrated in figures 23.2 - 23.5, and in other figures in the manual.

23.1 Network states, nodes

Network states in attractor basins are usually represented by circular nodes, but may also be shown as a bit/value pattern in 1d or 2d, or the decimal or hex value of the state, or they may not be shown (section 26.2).

23.2 Attractor cycles

Where 3 or more states make up an attractor cycle, this is represented as a regular polygon with nodes at the vertices (figure 23.5). The direction of time is clockwise. The "last" vertex, from which the first subtree is generated, is shown due east, so the first vertex at which the cycle is entered is one node clockwise from due east. A point attractor is shown as a node (positioned north-east) on a circle, indicating that the node cycles to itself (figure 23.2). A two state attractor is shown as two nodes on a circle (positioned south-east and north-west), where the south-east node is the "last" node. The diameter of attractor cycles increases asymptotically with system size n, up to the maximum selected, which also determines the scale of the entire basin.



For a binary network size n, an example of one of its states B might be $1010\ldots0110$. *State-space* is made up of all $S = 2^n$ states ($S = v^n$ for multi-value) – the space of all possible bitstrings or patterns.

Part of a *trajectory* in state-space, where C is a successor of B, and A is a *pre-image* of B, according to the dynamics of the network.

The state B may have other pre-images besides A, the total number is the *in-degree*. The pre-image states may have their own pre-images or none. States without pre-images are known as *garden-of-Eden* states.

Any trajectory must sooner or later encounter a state that occurred previously - it has entered an attractor cycle. The trajectory leading to the attractor is a *transient*. The period of the attractor is the number of states in its cycle, which may be just one - a point attractor.

Take a state on the attractor, find its pre-images (excluding the pre-image on the attractor). Now find the pre-images of each pre-image, and so on, until all garden-of-Eden states are reached. The graph of linked states is a *transient tree* rooted on the attractor state. Part of the transient tree is a subtree defined by its root.

Construct each transient tree (if any) from each attractor state. The complete graph is the *basin of attraction*. Some basins of attraction have no transient trees, just the bare "attractor".

Now find every attractor cycle in state-space and construct its basin of attraction. This is the *basin of attraction field* containing all 2^n states in state-space, but now linked according to the dynamics of the network. Each discrete dynamical network imposes a particular basin of attraction field on state-space.

Figure 23.1: This figure explains the idea of basins of attraction in discrete dynamical networks. Given an invariant network architecture and the absence of noise, the dynamics is deterministic, and follows a unique trajectory from any initial state. When a state that occurred previously is re-visited, which must happen in a finite state-space, the dynamics become trapped in a perpetual cycle of repetitions defining the attractor (state cycle) and its period (minimum one, a stable point). The approach is analogous to Poincaré's "phase portrait" in continuous dynamics. These systems are dissipative. A state may have multiple "pre-images" (predecessors), or none, but just one successor. The number of pre-images is the state's "in-degree". In-degrees greater than one require that transient states exist outside the attractor. Tracing connections backwards to successive pre-images of transient states reveals a tree-like topology where the "leaves" are states without pre-images, known as garden-of-Eden states. Conversely, the flow in state-space is convergent. The set of transient trees and the attractor on which they are rooted make up the basin of attraction. *Local* dynamics connects state-space into a number of basins, the basin of attraction field, representing the system's *global* dynamics.

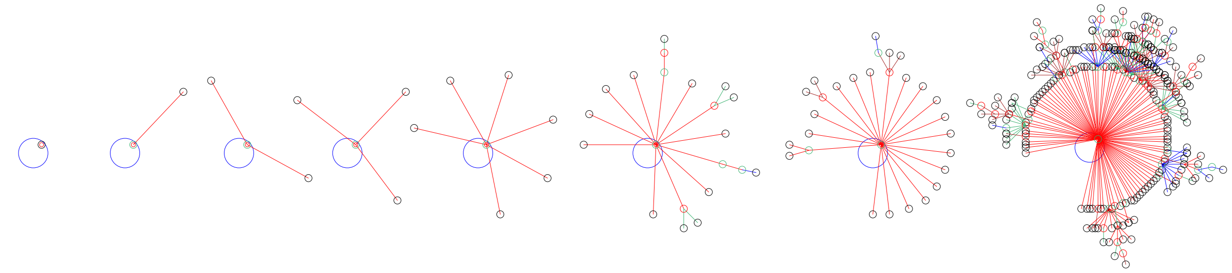


Figure 23.2: A point attractor (period=1) is represented as a node cycling to itself. The single node and the center line of the pre-image fan (level 1) are set at a default angle of 45°. The examples above show point attractors with increasing fan size, from 0 upwards (from rule 77 $v2k3n12$).

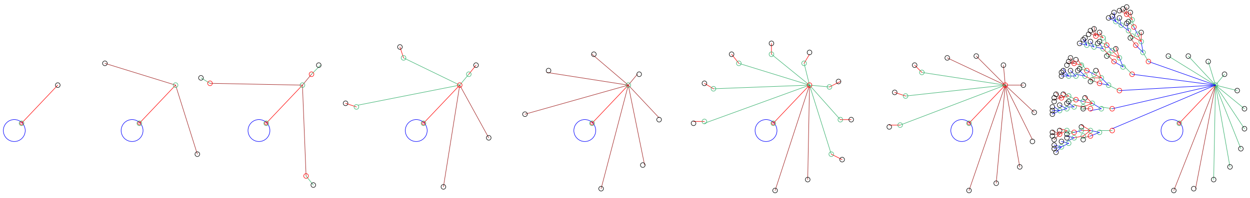


Figure 23.3: The pre-image fan (level 2) of a single pre-image (level 1) of a point attractor. The examples show increasing fan sizes, from 0 upwards (from rule 110 $v2k3$, n=(1-3) and n=(6-10)). In this case both the point attractor and its pre-image are uniform states (all0s and all1s) so the pre-image fan and attached trees are organised into equivalent groups (section 23.3.2.

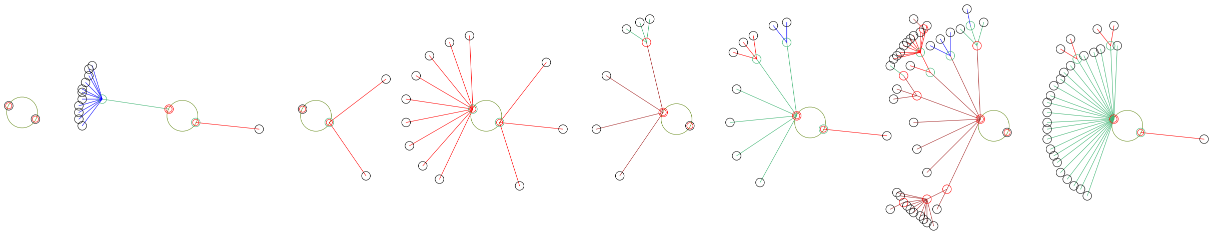


Figure 23.4: Period 2 attractors which cycle to each other. The two nodes are set at a default angle of -15°. Varying sizes of the pre-image fan to both attractor states are shown, from 0 upwards. A fan at level 1 is slightly angled to indicate that the direction of time is clockwise in the attractor – the center lines of subsequent fans are radial to the center of the attractor cycle (taken from rule 33 $v2k3n13$).

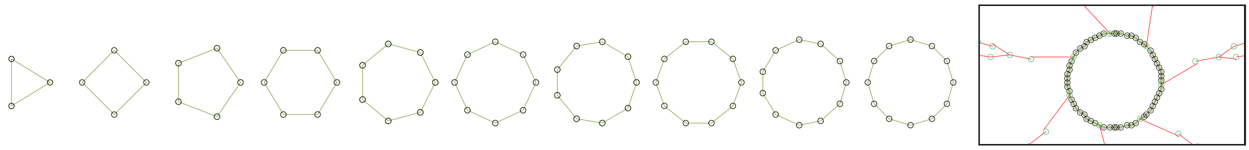


Figure 23.5: Attractors with periods≥3 are represented by polygons whose diameter approaches an upper limit with increasing period. The examples *left* have periods of 3 to 12 (from rule 248 $v2k3$, n=(3-12)). The framed example *right* is a fragment of a basin from rule 30 $v2k3n14$ with attractor period 63. A single edge or a fan at level 1 is slightly angled to indicate that the direction of time is clockwise in the attractor – the centre lines of subsequent fans are radial to the centre of the attractor cycle – see figure 2.5 for a clearer example.

23.3 Transient trees

The pre-images or predecessors (if any) of the "last" attractor state are computed first. The pre-image fan angle is set according to the in-degree and increases asymptotically to a maximum value with increasing in-degree. In general the centre line of the fan angle is radial to the attractor cycle centre point, but for fans rooted on the attractor, the fan angle is tilted slightly to indicate that the direction of time around the cycle is clockwise (figure 2.5).

The pre-image fan is computed and drawn for each node at each successive level in the tree. The endpoints of the fan where the nodes are drawn are positioned on notional concentric circles around the attractor cycle corresponding to successive levels in the tree. The distance between each successive concentric circle, and thus between tree levels away from the root, decreases asymptotically for 90 levels, thereafter remains constant.

The tree will grow away from the attractor, backwards in time. Once the tree is complete with all its garden-of-Eden states reached, the next tree (anti-clockwise on the attractor) is computed. (note that with compression on (see section 26.1), equivalent trees will be computed and drawn simultaneously).

23.3.1 Transient tree colors

A cycle of four colors is used to draw transition edges. The color scheme of the attractor basin depends on the start edge color which can be changed, resulting in four alternative color schemes for both edges and transition nodes (section 26.3.3).

The edges in the same pre-image fan are drawn in the same color (except for the uniform states – section 23.3.2 below). If the attractor period is 5 or less, successive fans are assigned a different color so that a given transient tree may have a mix of colors. For attractor periods of 6 or more, all fans in the same tree are assigned the same color, and the color is changed for the next non-equivalent tree. Note that with compression on (section 26.1) equivalent trees will be colored identically. Conventions for bit pattern node colors are described in section 26.2.1.

23.3.2 Transient trees for uniform states

for 1d and 2d CA

If CA compression is set, a special algorithm is employed to speed up computation of the subtree of the "uniform" states where all cells have equal value. When uniform states are on the attractor, the attractor period cannot exceed v (2 for binary). If a given state is a pre-image of a uniform state, then that state's rotation equivalents[19] must also be pre-images, and may be computed simultaneously by an appropriate rotational transformation.

The first level of a uniform state's tree is organized into groups of equivalents (shown in different colors). The subtree of each representative state in each group at this first level is computed in turn. Each subtree will be completed before the next is started. Successive pre-image fans in each non-equivalent subtree are assigned a different color. Equivalent subtrees will be computed and drawn simultaneously, and will be colored identically (figure 26.3).

23.3.3 Subtree only

A subtree only may be selected, running backward from a given state, or from a state a specified number of time-steps forward from a given state, because most states have no are pre-images (they are leafs, garden-or-Eden states). The first pre-image fan is evenly spread around a notional circle with a diameter equal to the current maximum attractor diameter. Successive pre-image fans are computed and drawn for each node at each successive level in the subtree.

If the subtree seed has just one pre-image, the pre-image fan at level 2 will be spread out as shown in figure 23.6 (the two subtrees top left) following similar layout principles as figure 23.3.

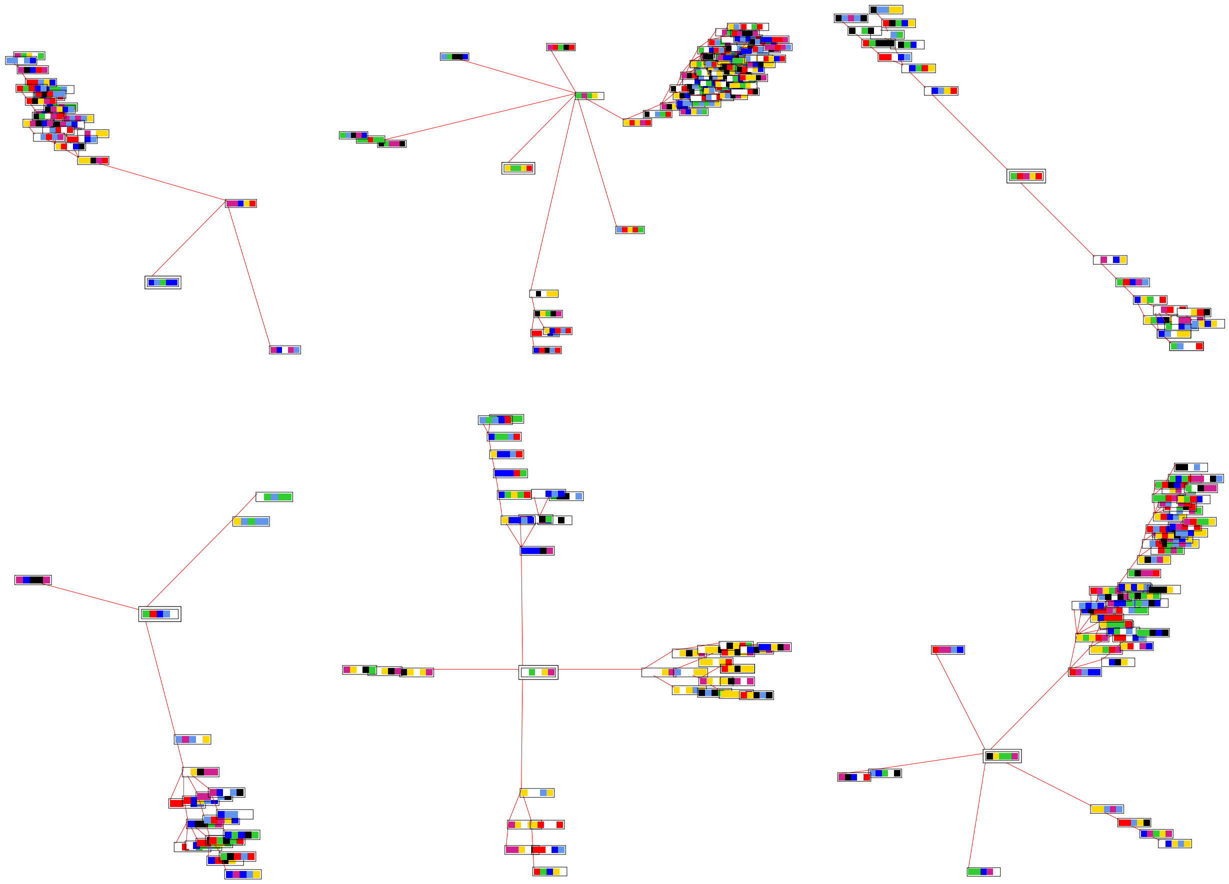


Figure 23.6: Examples of sub-trees from root states with in-degrees (at level 1) of 1 to 5. These examples are For CA $v8k3$, n=5 for various rules, and random initial states, running forward by about 3 time-steps before running backward. States are shown as value patterns – the root state is highlighted by default inside a double outine. For in-degrees 1 (at level 1 (the two subtrees top left) similar layout principles apply as in figure 23.3.

Intentionally Blank

Chapter 24

Output parameters for attractor basins

This chapter applies in SEED-mode or Field-mode, not TFO-mode.

There are many output parameter options and sub-options relating to attractor basins. Each can be looked at in turn, but for convenience they are divided into seven categories to allow jumping directly to the category where options need to be changed from the current default settings. All the options have defaults so other categories can be skipped.

The parameter categories (and relevant chapters) are,

options ...	*what they mean*
start/misc-ret ...	start/miscellaneous options difficult to categorize (section 24.4).
layout-l ...	layout of attractor basins – their scale, position and spacing (chapter 25).
display-p ...	display of attractor basins and nodes (chapter 26).
data/pause-t ...	pausing between trees/basins, specifying data to be recorded (chapter 27).
PScript-P ...	to create a vector PostScrip file of the attractor basin/s (section 24.2).
jump-j ...	*(FIELD-mode only)* to display the jump-graph (section24.3).
mutation-m ...	various mutation settings for the *next* network (chapter 28).

24.1 The first output parameter prompt for attractor basins

The first output parameter prompt is presented in a top-right window, and differs between SEED and FIELD modes. At the same time **output parameters** appears in a top center window.

SEED-mode

```
accept all basin defaults-d, space-time patterns only-s
restore defaults: all-a, layout only-L
revise from: start/misc-ret layout-l display-p
             data/pause-t PScript-P mutation-m:
```

FIELD-mode

```
accept all basin defaults-d
restore defaults: all-a, layout only-L
revise from: start/misc-ret layout-l display-p
             data/pause-t PScript-P jump-j mutation-m:
```

24.1.1 output parameter prompt for attractor basins – options summary

The meaning of the options in the first output parameter prompt for attractor basins (section 24.1) are explained in more detail below. As soon as a "**revise from**" option is selected, the reminder **accept defaults-d** appears in a top center window.

options . . . *what they mean*

accept all basin default-d . . . to accept all current output parameter defaults, and skip further prompts. This can be done at any stage in the prompt sequence. New settings generally become the new defaults, but can be reset to the original. The next prompt for a basin of attraction field will be presented in section 29.4, or for a subtree or single basin in section 29.1.

space-time patterns only-s . . . *(SEED-mode only)* enter **s** to skip all attractor basin options and go directly to the output parameters for space-time patterns (chapter 31). Note that a final chance to select space-time patterns is also given in section 29.1.

restore defaults: . . . *to restore defaults to their original settings*

all-a . . . to restore all defaults .

layout only-L . . . to restore just the layout defaults (chapter 25).

revise from: . . . *jumping directly to an option category*
At any point enter **q** (or **q** more than once) to backtrack to the first output parameter prompt (section 24.1).

start/misc-ret . . . enter **return** to start miscellaneous (hard to categorize) options, presented in sequence in a top-right window – section 24.4.

layout-l . . . for a sequence of prompts to set the scale, position and spacing of attractor basins – described in chapter 25.

display-p . . . for a sequence of display prompts, the compression of equivalent basins and trees, types of nodes, orientation, and in-degree angle – described in chapter 26.

data/pause-t . . . for a sequence of prompts for pausing attractor basins to re-adjust the layout on-the-fly, and to show/print data at various levels of detail – described in chapter 27.

PScript-P . . . to create a vector PostScript file of the attractor basin/s (default file name is `my_bPS.ps`) – section 24.2.

jump-j . . . *(FIELD-mode only)* to activate the jump-graph (section 20.3) – section 24.3.

mutation-m . . . for a sequence of prompts to set the type of mutation (to wiring or rules) to be applied to the *next* network generating the *next* attractor basin/s with the same parameters – described in chapter 28.

accept defaults-d . . . to accept all remaining defaults, and skip further output parameter prompts.

24.2 PostScript of attractor basins

Enter *PScript*-**P** at the first output parameter prompt (section 24.1), or arrive there by viewing the output parameters in sequence, to create a vector PostScript file of the basin of attraction field, single basin, or subtree, or a range of the above. The following top-left prompt is presented,

> *if PostScript is currently OFF, the initial case*
> **PScript: save basin to PostScript (now OFF): greyscale-P color-p:**
>
> *if PostScript is currently ON, either greyscale-***P** *or color-***p**
> **PScript: save basin to PostScript (now p): greyscale-P color-p cancel-0:**

Enter **0** to deactivate, **P** or **p** to activate – a filename prompt will be presented (section 35.2). If saving to postScript is active, a new file will be created and overwritten to the selected filename each time an attractor basin is drawn (default filename `my_bPS.ps`). To conserve the file, the selected filename should be renamed with utilities[1] outside DDLab before a new attractor basin is generated. Examples of vector PostScript images of basins are to be found throughout this manual, for example in chapters 25 and 26. For PostScript files of jump-graph basins refer to section 20.7.1.

24.3 Activate the jump-graph

FIELD-mode only - see also section 20.3.1

Enter *jump*-**j** at the first output parameter prompt (section 24.1) to go directly to the jump-graph prompt, or arrive there by viewing the output parameters in sequence. The following prompt is presented,

> **jump: attractor jump-graph -j, no edges layout only +L:**

Enter **j** to show the jump-graph, or **jL** for "layout only" – the jump-graph without edges is applied for just laying out attractors in any arbitrary postion. If the jump-graph (with edges) is selected, "compression" for CA will be turned off automatically, with the following message,

> . . . **- compression OFF:** *(if compression was on, Enter* **return** *to continu)*

The jump-graph represents the probabilities of jumping between basins due to single-bit or single-value perturbations to attractor states, and gives some insight into the stability and adaptability of the dynamics. An alternative use of the jump-graph is a method for just laying out the basins in a basin of attraction field in any arbitrary position. A complete description of the jump-graph and its functions is provided in chapter 20.

[1] For example, rename in a terminal, look at the postScript file in "GhostView".

24.4 Miscellaneous (hard to categorize) options

The rest of this chapter describes the first category of miscellaneous (hard to categorize) options,

options . . . *what they mean*

state-space matrix . . . an additional graphical method for representing states in attractor basins (section 24.5) which can also be activated/activated on-the-fly (section 30.3).

in-degree frequency . . . the frequency of different in-degrees in a subtree, basin or field shown as a histogram (section 24.6).

show/count majority . . . highlight and count states in basins or subtrees with a majority of a given value. For v=2 this provides an alternative measure of fittness in the "density classification problem"[6, 11] (section 24.7).

screensave demo . . . *(SEED-mode only)* a continuous demo of single basins or subtrees for different or mutant rules (sections 4.12 and 24.8).

G**-density,** Z **and** λ . . . graphs of the density of garden-of-Eden (leaf) states against system size, and how various rule parameters are distributed in rule-space (section 24.9).

backwards space-time patterns . . . to show space-time patterns being computed "backwards" as attractor basins are being drawn (section 24.14) which can also be activated/deactivated on-the-fly (section 30.3).

basin speed . . . to slow down drawing attractor basins and backwards space-time patterns or revert to full speed (section 24.15), which can also be done on-the-fly (section 30.3), while interrupting (section 30.2), and at the "attractor basin complete" prompt (section 30.4).

Enter **return** at the first output parameter prompt (section 24.1) to access these options in sequence.

24.5 State-space matrix

The state-space matrix plots each state in state-space on a 2d grid in the lower right corner of the screen, plotting the left half against the right half of each state bit/value string. The x-axis represents the left $n/2$ bits/values, the y-axis represents the right $n/2$ bits/values. If n is odd, the extra bit/value is included on the left, and the grid is a flat rectangle as in figure 24.2(*bottom*), otherwise the grid is square. The scale of the matrix is set automatically according to n, though this can be changed.

For a single basin, attractor cycle states and transient states are shown in different colors. For a basin of attraction field, each basin is assigned a different color (cycling through 15 colors). The start color for the color cycle may be reset to produce different color schemes. If the grid size is big enough, the decimal equivalent of the states are also printed (figure 24.1).

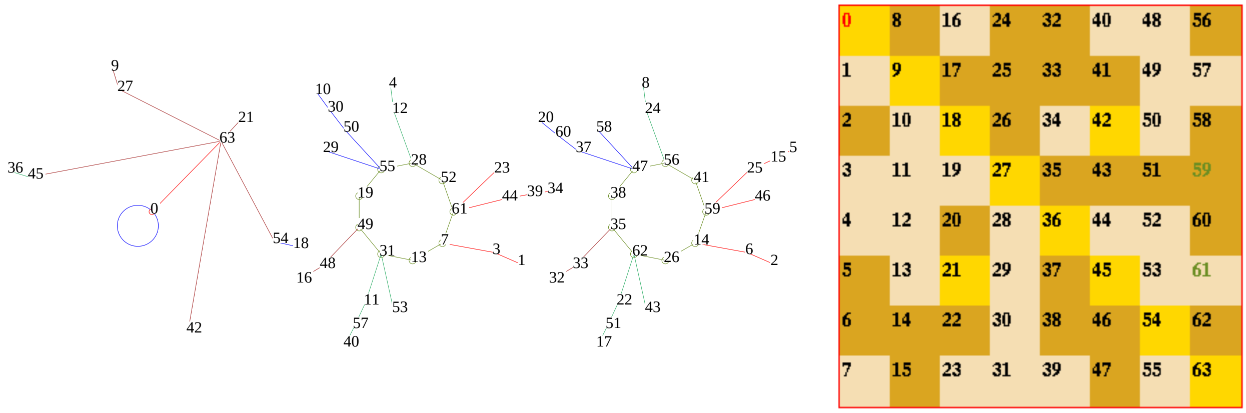


Figure 24.1: The state-space matrix of the basin of attraction field of a cellular automaton, $v2k3$ rule (dec)110, n=6. *Left:* the uncompressed state transition graph. decimal numbers representing states. *Right:* The state-space matrix, where the 3 basins are represented by 3 colors. The decimal equivalents of states are shown in both cases.

Enter **return** at the first output parameter prompt (section 24.1) for the first miscellaneous (start/misc:) option (section 24.4) for the state-space matrix,

> **start/misc: state-space matrix: all-states-m, attractor only-a**
> **show and change: matrix size-s, start color-c:**

24.5.1 state-space matrix – options summery

options . . . *what they mean*

all-states-m . . . to display the matrix, including all states in the subtree, single basin or field. For a single basin, attractor cycle states and transient states are shown in different colors. For a basin of attraction field, each basin is assigned a different color (cycling through 15 colors).

attractor only-a . . . to display the matrix showing just attractor states in the single basin or field. For a basin of attraction field, each basin is assigned a different color (cycling through 15 colors).

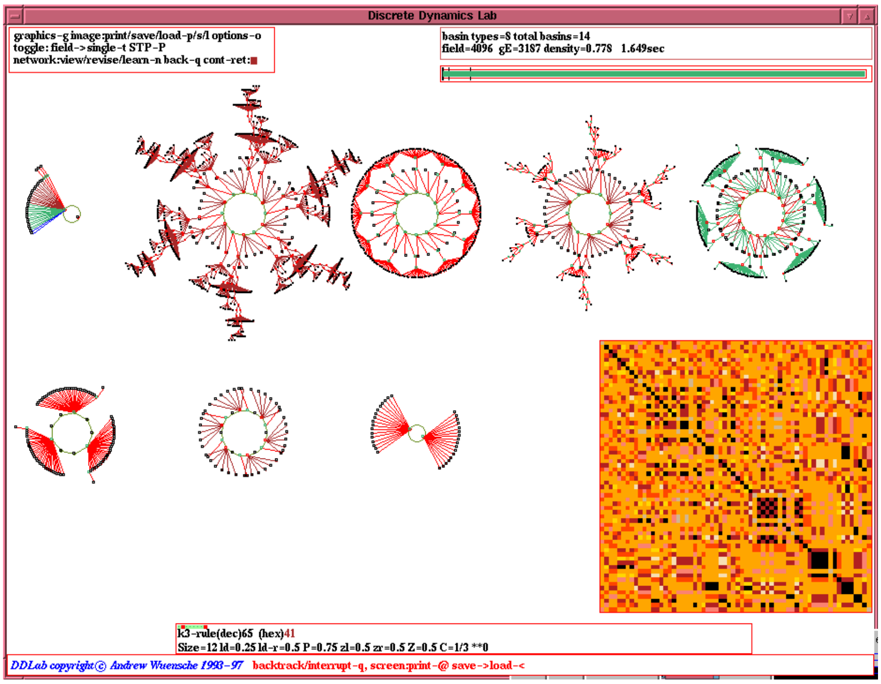
matrix size-s . . . to change the matrix size from the default. The following prompt is presented,

> **change matrix size: % of 1/2 screen(default 63%):**

Enter the new size as a percentage of half the screen width. If the matrix size is big enough, the decimal equivalent of states will be shown on the matrix.

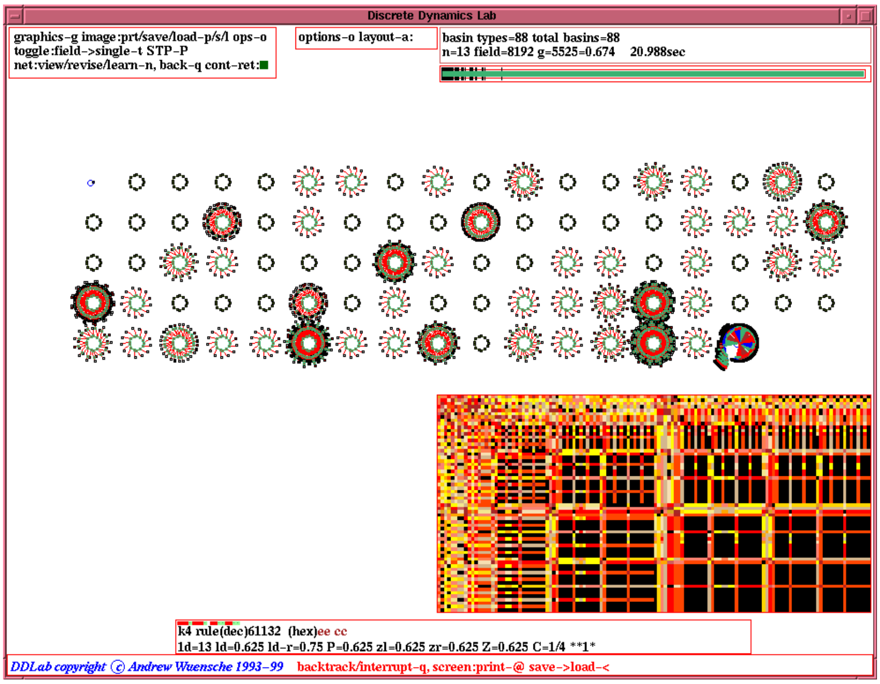
start color-c . . . to change the start color in the color cycle of 15 colors, and produce different color schemes. The following prompt is presented,

> **change start color (now 10): enter 1-15:**


Discrete Dynamics Lab
graphics-g image:print/save/load-p/s/l options-o
toggle: field->single-t STP-P
network:view/revise/learn-n back-q cont-ret:■
DDLab copyright© Andrew Wuensche 1993–97 backtrack/interrupt-q, screen:print-@ save->load-<

basin types=8 total basins=14
field=4096 gE=3187 density=0.778 1.649sec

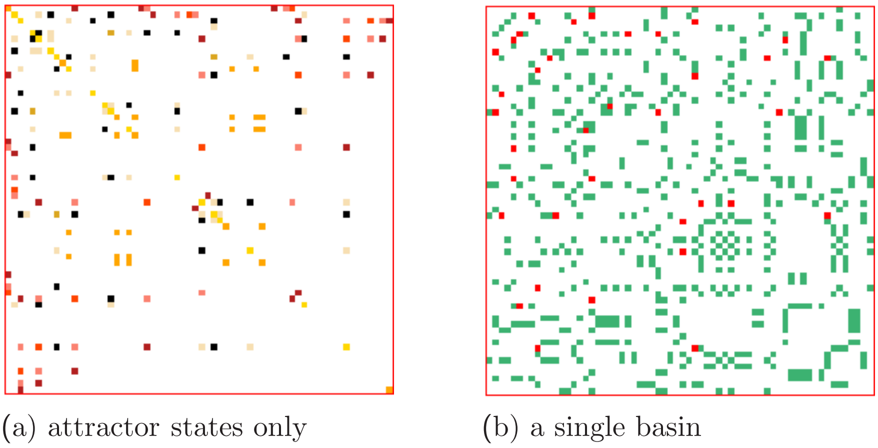
k3-rule(dec)65 (hex)41
Size=12 ld=0.25 ld-r=0.5 P=0.75 zl=0.5 zr=0.5 Z=0.5 C=1/3 **0


Discrete Dynamics Lab
graphics-g image:prt/save/load-p/s/l ops-o
toggle:field->single-t STP-P
net:view/revise/learn-n, back-q cont-ret:■
options-o layout-a:
DDLab copyright © Andrew Wuensche 1993–99 backtrack/interrupt-q, screen:print-@ save->load-<

basin types=88 total basins=88
n=13 field=8192 g=5525=0.674 20.988sec

k4 rule(dec)61132 (hex)ee cc
1d=13 ld=0.625 ld-r=0.75 P=0.625 zl=0.625 zr=0.625 Z=0.625 C=1/4 **1*

Figure 24.2: The state-space matrix (lower right on each screen) of the basin of attraction field of a CA, with compression off. *Top:* $v2k3$ rule (dec)65, n=12, the 8 basins types are represented by 8 colors in the matrix. *Bottom:* $v2k4$ rule (hex)eecc, n=13. Because n is odd, and the extra bit is included on the left, the grid is a flat rectangle. The 88 basins are represented by cycling through 15 colors.



(a) attractor states only (b) a single basin

Figure 24.3: The state-space matrix of the same CA in figure 24.2 *top*, $v2k3$ rule (dec)65, n=12.
(a) just the attractor states of the of the 8 basin types represented by 8 colors.
(b) just the states in the 5th basin type (top right in figure 24.2 *top*). The attractor and transient states are represented by 2 different colors.

24.5.2 Toggle the matrix on-the-fly

Whether set or not in section 24.5, the state-space matrix can be toggled on/off while attractor basins are being drawn. The following reminder will appear in the bottom title bar,

matrix-m STP-s scroll-# exp-e contr-c

Enter **m** to toggle the state-space matrix on-the-fly. Note that the state-space matrix can also be toggled on-the-fly when showing space-time patterns, forward only (section 32.11.7). Options #, **e** and **c** are described in section 24.16.

24.6 In-degree frequency histogram

DDLab can record the frequency of different in-degrees (the number of pre-images or predecessors) that occur in a subtree, basin or field. If this option is selected, the in-degree information is displayed as a histogram when the attractor basin is complete, or during a temporary pause (section 30.2).

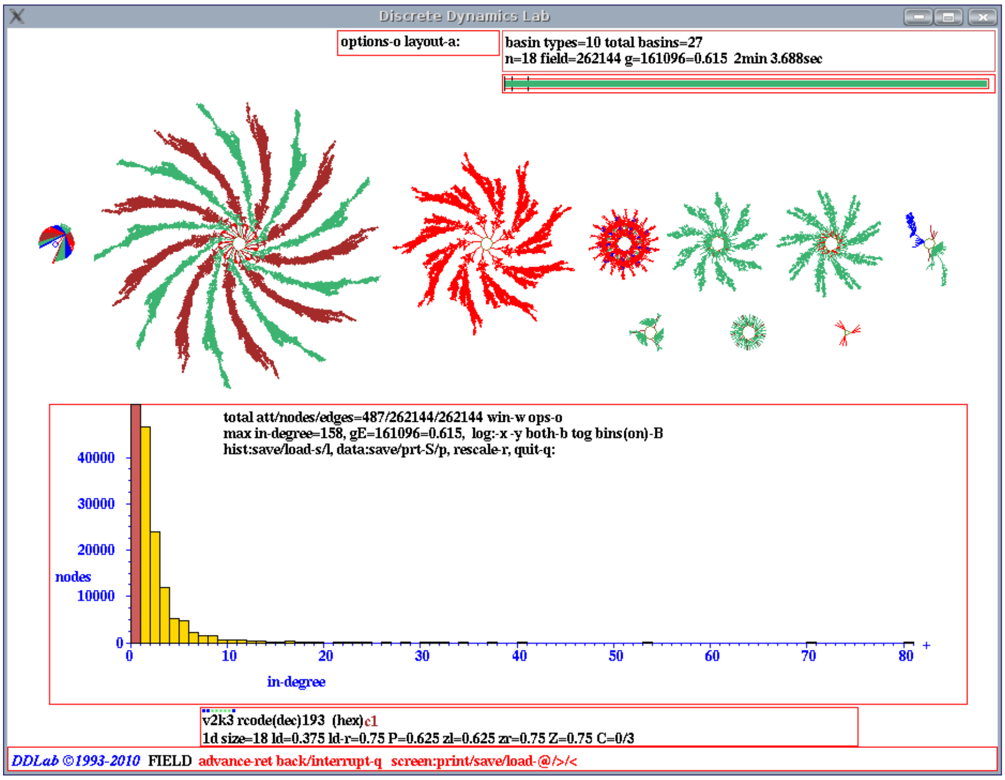
The following prompt is presented in (section 24.4)

in-degree show frequency histogram-h:

If **h** is entered, the histogram setting remains active for all further attractor basins. It may be deactivated by not entering **h** at this prompt, by restoring all "output parameter" defaults (see section 24.1), or by backtracking through the main prompt sequence beyond section 9.

24.6.1 In-degree frequency cut-off

If **h** is entered above, the following prompt allows setting an upper threshold (or cut-off) to the in-degrees recorded, to limit the size of the histogram. Any cut-off in the range of 50 to 5000 may



options-o layout-a:

basin types=10 total basins=27
n=18 field=262144 g=161096=0.615 2min 3.688sec

total att/nodes/edges=487/262144/262144 win-w ops-o
max in-degree=158, gE=161096=0.615, log:-x -y both-b tog bins(on)-B
hist:save/load-s/l, data:save/prt-S/p, rescale-r, quit-q:

v2k3 rcode(dec)193 (hex)c1
1d size=18 ld=0.375 ld-r=0.75 P=0.625 zl=0.625 zr=0.75 Z=0.75 C=0/3

DDLab ©1993-2010 FIELD advance-ret back/interrupt-q screen:print/save/load-@/>/<

Figure 24.4: In-degree histogram which has been rescaled from the default settings, of a a basin of attraction field of a CA, $k = 3$ (rule 193), $n = 18$. The attractor basins are shown above.

be set, the default is 200, or the current setting. The occurrence of in-degrees equal to or above the cut-off value will be added to the cut-off frequency "bin" in the histogram.

> **select cut-off in-degree bin (50-5000, default 200):**

24.6.2 In-degree frequency window

If the in-degree frequency option was selected, the histogram and data will be displayed in a window in the lower section of the screen. This occurs when attractor basins are complete, or if interrupted early, in which case a prompt similar to this is presented (described in section 30.2),

> **early exit - in pre-image fan inhist-h, next tree/basin-n, options-o, stop field-q, cont-ret:**

Enter **h** to see the in-degree frequency window, with the data and histogram corresponding to the basin generated so far. The x-axis represents the range of in-degrees, from in-degree zero (garden-of-Eden states) upward. The y-axis represents the number of states having a given in-degree. The last, black, frequency column (if applicable) represents combined in-degrees equal to the cut-off value and above.

The garden-of-Eden column is colored red, the other columns yellow. Initially the scales of the x and y axis are automatically set according to the range of in-degrees encountered, but can be separately re-scaled or shown in log form for a clearer view of the spread of frequencies or to amplify smaller in-degrees.

24.6.3 In-degree data and prompts

An example of the data displayed in the window, and initial prompts (as in figure 24.4 for the basin of attraction field of a k=3 CA, rule 193, n=18) is shown below,

total att/nodes/edges=487/262144/262144 win-w ops-o
max in-degree=158, gE=161096=0.615, log:-x-y both-b tog bins(on)-B
hist:save/load-s/l, data:save/prt-S/p, rescale-r, quit-q:

For a subtree the the prompt starts with **subtree: nodes/edges=** *dots*

24.6.4 in-degree window – data decode

The data shown in in-degree frequency window (section 24.6.2) signifies the following,

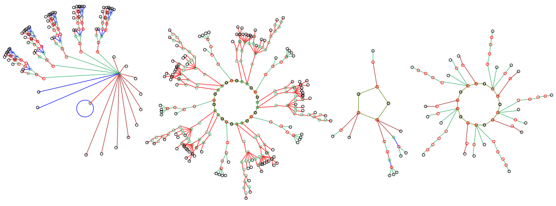
total att . . . the number of states on attractor cycles (does not apply to subtrees).
nodes . . . the number of nodes whose pre-images have been computed.
edges . . . the total number of edges computed so far.
max in-degree . . . the largest in-degree found, possibly more than the cut-off limit.
gE . . . the frequency and fraction of garden-of-Eden states (in-degree 0), usually the greatest frequency.

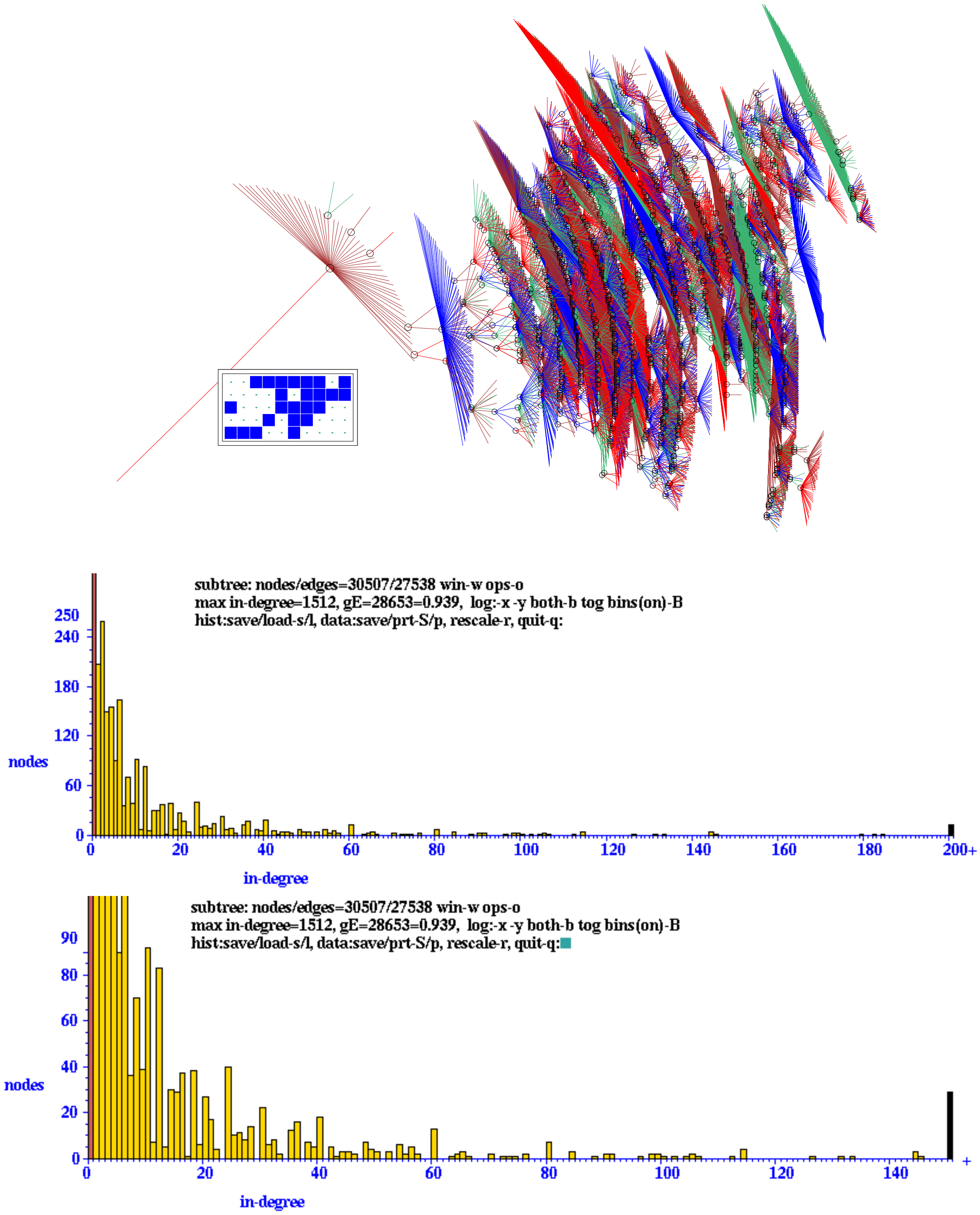
Note that for a complete basin (or field), nodes=edges, for a complete subtree, nodes=edges+1, and for an incomplete attractor basin, (i.e. if interrupted), nodes<edges.

24.6.5 in-degree window – options summery

win-w . . . to toggle the in-degree window itself, in case its hiding attractor basins.
ops-o . . . for further options described in section 30.4.
log:-x-y both-b . . . to transform to a log plot on either axis or both (section 24.6.6).
tog bins(on) . . . for a binned or simple log plot (section 24.6.6).
hist:save/load-s/l . . . to save histogram data as DDLab binary file, (default filename my_prh.prh), or load the data file and regenerate a new histogram.
data:save/prt-S/p . . . to save histogram data as an ascii file, (default filename my_data.dat), or print the ascii data to the terminal (not for DOS). The following example is for CA, $v2k3$ rule 193, n=10 (basin field shown).

```
pre-image frequency distribution
0: 442
1: 350
2: 136
3: 45
4: 30
6: 10
7: 10
17: 1
```





subtree: nodes/edges=30507/27538 win-w ops-o
max in-degree=1512, gE=28653=0.939, log:-x -y both-b tog bins(on)-B
hist:save/load-s/l, data:save/prt-S/p, rescale-r, quit-q:

subtree: nodes/edges=30507/27538 win-w ops-o
max in-degree=1512, gE=28653=0.939, log:-x -y both-b tog bins(on)-B
hist:save/load-s/l, data:save/prt-S/p, rescale-r, quit-q:

Figure 24.5: *Top:* A subtree of a CA, $v2k3$ rule (dec)193, with the n=50 1d root state shown in 2d 10×5), leaf nodes suppressed and the STG rotated by 90°. *Below:* Two versions of the in-degree histogram, with increasing rescaling (x and y axis) for a closer view. Excess in-degrees (indicated by "+") are included in the last black column on the right. Zero in-degrees (leaf states) are in the first (red) column on the left.

rescale-r . . . to rescale the histogram (section 24.6.7).

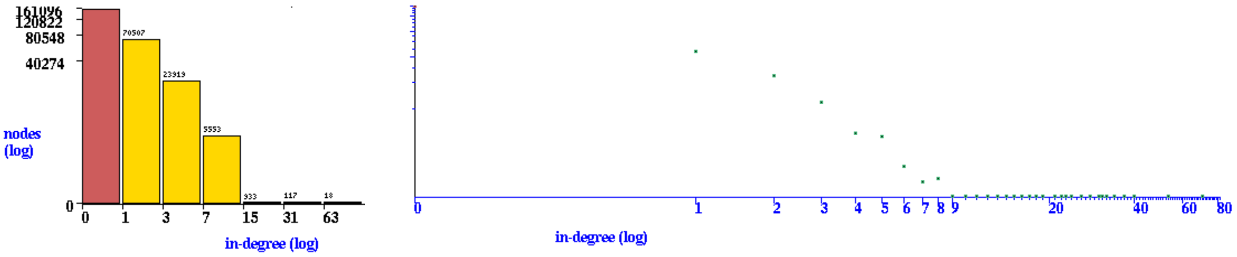


Figure 24.6: The in-degree histogram as a log-log plot can be implimented in two ways on the x-axis, *Left:* preserving a histogram (the default) with bins that capture an increasing range of in-degrees, 0, 1 – 2, 3 – 7, 8 – 15, etc. *Right:* a simple log-log plot. The x and y axis can also be treated separately. These plots are for the CA, $v2k3$ rule (dec)193, in figure 24.5.

24.6.6 In-degree log plot

The in-degree histogram can be transformed to a log-log plot base 2. There are two ways of treating the x-axis. The default (binned) method preserves a histogram with equal width bars, where each bar combines a range of in-degrees, 0, 1 – 2, 3 – 7, 8 – 15, etc. The alternative method is a simple log-log plot. Both are illustrated in figure 24.6. Enter **B** to toggle between the two methods – the in-degree histogram changes color, yellow for binned, otherwise green.

Enter **x**, **y** or **b** in section 24.6.3 to replot the histogram according to log2 scale for just the x-axis, just the y-axis, or both axes.

24.6.7 Rescaling the x/y-axis

Enter **r** in section 24.6.3 to rescale the x and y axes. The following additional prompts are presented, depending on x-max, the max in-degree found,

if x-max ≤ 30
re-scale y axis (def-d):y-max:

if x-max > 30, prompts in turn)
re-scale x/y axis (def-d):x-max (30-158): y-max:

Enter the rescaled settings, or **d** to restore the originals – the histogram will be redraw accordingly, as in figure 24.5.

The x-axis is initially scaled to show the spread of in-degrees from zero to the maximum found (or to the cut-off value). If the cut-off value is shown, + is added at the end of the x-axis indicating that in-degrees equal to or greater than the cut-off value have occurred. If the maximum in-degree found is greater than 30, the x-axis can be rescaled by entering a new maximum in-degree. Excess in-degrees (off the scale) will be shown as a black column on top of the maximum in-degree column, and + will be added to the x-axis. The y-axis is initially scaled to contain the highest in-degree frequency column in the histogram (usually column 0, leaf states). It may be rescaled to any value, for instance to show smaller in-degree frequencies at a larger scale.

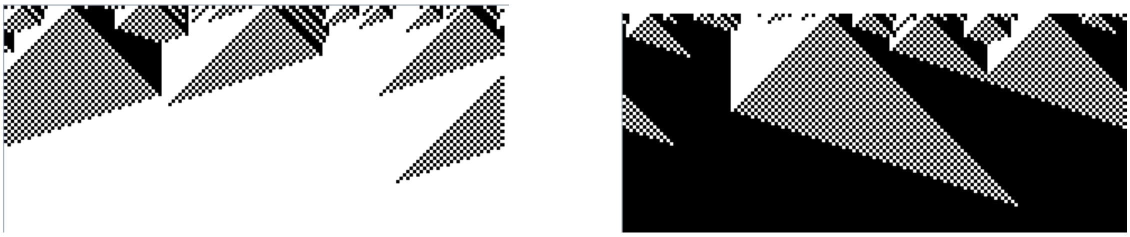


Figure 24.7: The density classification problem in emergent computation seeks to evolve a rule that is able categorise states between a majority 0s and 1s. In this example of typical space-time patterns (n=149), the rule is $v2k7$ (hex)(ff f0 8c f0 ff e0 00 f0 ff f0 00 f0 ef e0 00 e0), evolved by Raja Das[6].

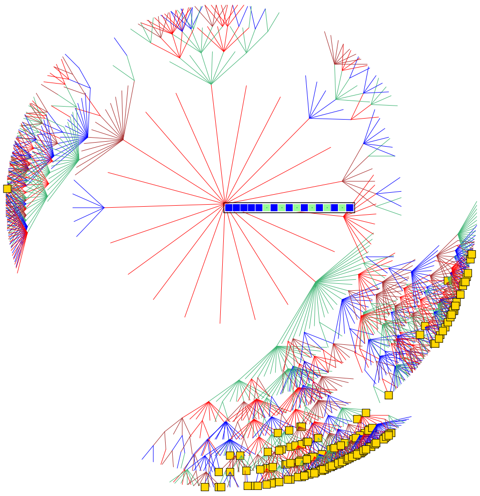


Figure 24.8: A subtree showing exceptions in the density problem. Using the same rule as in figuree 24.7, a subtree was generated from an initial state with 11 1's out of n=17. Most states in the subtree also have a majority of 1s, but there are 9.63% exceptions, indicated in the "basin complete window" (section ??), and by yellow squares with a black border in the subtree.

24.7 Density classification problem

The "density classification problem" is a favourite exercise in "emergent computation"[6, 11] which seeks to evolve CA rules able classify states between a majority 0s and 1s – the fitness is measured by running forward from many random initial states. This section presents an alternative method of measuring fitness, by generating basins or subtrees, and highlighting and counting the exceptions to the intended classification. A good density rule will have two point attractors, the all 0 state attracting the majority-0 states, and the all-1 state attracting the majority-1 states (figure 24.9). Likewise, a subtree with a given majority should attract the same majority value.

DDlab is able to test for a majority of any value (0 to 7) in basins or subtrees, but the examples in this section relate just to the majority of 1s and 0s.

The following prompt is presented in section 24.4,

show/count states with majority, enter (0 - 1): *(for v=2)*

Enter **1** or **0** to highlight these states. The highlighted states are displayed as small yellow squares with a black border (figure 24.4). If this option is selected, the number and percentage of "exceptions" to either of these conditions is displayed in the "basin complete" in-degree window.

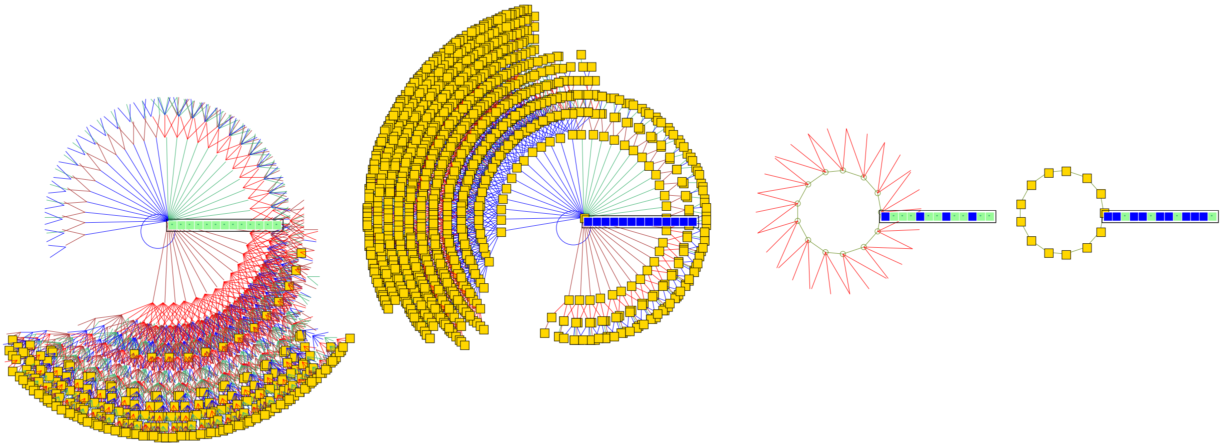


Figure 24.9: A basin of attraction field showing states with a majority of 1s. n=13, k=7, the rule is the same as in figure 24.8. There are 4 basins, but nearly all states belong to the all 0's and all 1's point attractors. States with a majority of 1's are indicated in the basins by yellow squares with a black border. The majority exceptions in the main basins are: all 0s 7.34%, all 1s 2.68%.

24.8 Screen-saver demo

If a single basin or subtree was selected at the first prompt in section 6.2, an option is presented for a continuously changing display where basins or subtrees are generated randomly in overlapping bubbles on the screen (figure 4.17). The following prompt is presented in section 24.4,

screensave demo -s:

Enter **s** to select the demo. An example is shown in figure 4.17. The backwards space time patterns and state-space matrix may also be set to work at the same time, or toggled on-the-fly (sections 24.5.2 and 24.14). Most setting, (network, layout, display, mutation) apply to the screen-saver demo. For a quick-start example see Section 4.12.

24.9 G-density, Z and λ

The G-density – of Garden-of-Eden (leaf) states – those without predecessors – in a subtree, basin or field, is a measure of convergence in network dynamics, which in turn relates to the quality of typical space-time patterns in CA, ordered-complex-chaotic[27].

A number of static parameters from the rcode-table itself also predicts convergence, to varying degrees. Though these the parameters are generalised for multi-value $v \geq 3$, in this section they are applied to binary, v=2, rcodes. If S is the rcode size, the parameters can be defined as follows,

λ-parameter . . . the fraction of 1s in S[14].

λ_{ratio} . . . the total of the lesser value (1s or 0s) divided by $S/2$[19].

Z-parameter . . . the probability that the next unknown cell in the CA reverse algorithm is determined[19, 27].

$0 \longleftarrow\!\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\!-\;\lambda_{ratio}\;-\!\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\!\longrightarrow 1$
$0 \longleftarrow\!\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\!- \; Z\text{-parameter} \; -\!\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\!\longrightarrow 1$
$\text{max} \longleftarrow\!\!\!- \text{ convergence}/G\text{-density } -\!\!\!\longrightarrow \text{min}$

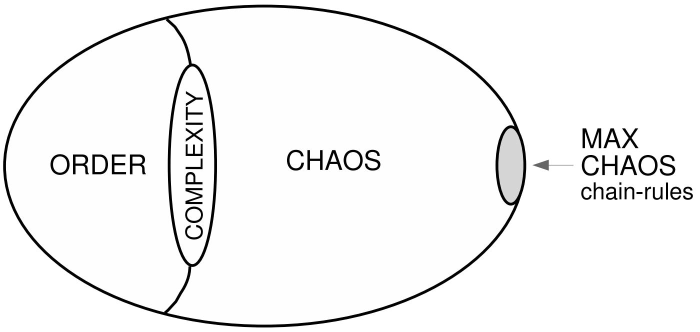


Figure 24.10: A view of rule-space (after Langton[14]). Tuning the Z-parameter from 0 to 1 shifts the dynamics from maximum to minimum convergence, from order to chaos, traversing a phase transition where complexity lurks. The chain-rules (about the square root of rule-space) on the right are maximally chaotic and have the very least convergence, decreasing with system size, making them suitable for dynamical encryption (section 16.11).

Figure 24.10 is a schematic view of rule-space illustrating how convergence and G-density relate to order/chaos – predicted by the Z-parameter and to a lesser extent by λ_{ratio} – only Z is able to identify the maximaly chaotic chain-rules (section 16.11). Complex rules occur at the order/chaos phase transition, but are difficult to identify by these static parameters[2].

λ_{ratio} is utilised instead of the λ-parameter because it can be compared directly to Z – both vary between 0 and 1 – where a low value indicates order and high convergence, and a high value indicates chaos and low convergence. λ_{ratio} approximates Z, which depends on the distribution as well as the proportions of values in the rcode – Z predicts behaviour more accurately (figure 24.11). To test these relationships, this section describes how graphs of G-density against Z and/or λ_{ratio} can be plotted for predefined CA rules.

Plots can be made of Z against λ_{ratio} (and Z_{left} against Z_{rigt}) to relate parameters for predefined CA rule samples. The proportions of rules in rule-space with various levels of canalizing inputs (chapter 15), λ_{ratio}, and the P-parameter[3] may be computed and displayed (figures 24.1 and 24.2) – these measures are of interest in predicting convergence and stability in random Boolean networks.

The variation of G-density with system size, n, is a useful measure of the order-chaos spectrum of CA behaviour[21, 35] (figure 24.13).

The following prompt is presented,

if v=2
G-density plot: Zpara-1, +Lambda ratio-2, size range-3
parameter plot: Zpara-Lda ratio (C-P data), Zleft-Zright -L:

if $v \geq 3$ *the G-density graph only*
G-density plot: size range-3

[2] Chapter 33 describes alternative methods for automatically classifying rule-space including complex rules.

[3] The P-parameter[9] is the fraction of 0s or 1s in the rule-table (whichever is more), P varies from 1 (order) to .5 (chaos) – a variation on the λ-parameter or λ_{ratio} favoured in theoretical biology.

24.9.1 G-density, Z and λ – options summery

options . . . *what they mean*

G-density plot: . . . *plots involving G-density*, for predefined rules . . .

Zpara-1 . . . enter **1** for a plot of Z against G-density (sections 24.10 – 24.10.2).

+Lambda ratio-2 . . . enter **2** for two simultaneous plots, Z against G-density, and λ_{ratio} against G-density (sections 24.10 – 24.10.2).

size range-3 . . . enter **3** to plot G-density against system size n. (section 24.11).

parameter graph : . . . *plots between parameters* for a predefined rule sample – enter **L** . . .

Zpara-Lda . . . to plot Z against λ_{ratio} (sections 24.12 – 24.12.3 .

Zleft-Zright . . . to plot Z_{left} against Z_{right} (sections 24.12 – 24.12.3) The Z-parameter is the greater of the two.

(C-P data) . . . to see the proportions of rules in rule-space – λ_{ratio}, P-parameter, and canalizing inputs (section 24.13). This can be activated once the $Z - \lambda_{ratio}$ or Z_{left} against Z_{right} plots (above) are complete.

24.10 G-density against Z and/or λ_{ratio}

Enter **1** in section 24.9 for a graph of G-density against Z, or **2** for an additional graph of G-density against λ_{ratio} drawn simultaneously.

Enter **d** at the next prompt (optional) to skip further options. The Z graph will be drawn in the lower right of the screen, the λ_{ratio} graph beside it on the left. A subtree, basin or field for each of the set of rules specified (in reverse decimal order) will be drawn (at a small scale by default) in a window in the upper part of the screen, as in figure 24.12. When each attractor basin is complete the G-density (y-axis) is plotted against the rule's Z and/or λ_{ratio} on the x-axis.

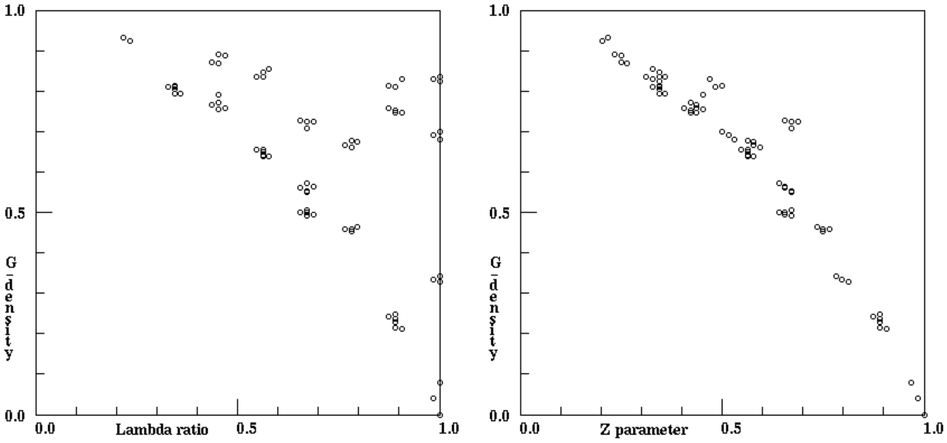


Figure 24.11: G-density plotted against λ_{ratio} (left) and the Z parameter (right). n=12, for the set of all k=7 totalistic rules. The complete basin of attraction field was generated for each rule and the garden-of-Eden states counted. Z provides a tighter prediction of G-density than λ_{ratio}.

24.10.1 G-density for tcode, or rcode $k \leq 3$

For tcode (totalistic rules), or rcode if $k \leq 3$, the rules included in the plot consist of a countdown from the decimal rule number or totalistic code originaly selected in section 16. To include all rules the maximum decimal rule number or totalistic code should have been selected, or all 1s entered if setting bits.

However, the range of Z may be restricted, with a lower and upper limit. Note that low values of Z (especially Z=0) should be avoided except for small network sizes, because in-degree (and G-density) will be very high. Very high in-degrees can exhaust computer memory or take a long time to compute. The following prompts are presented, firstly to set the lower limit, then the upper.

min Z (def 0.2): **set max Z (def 1):**

24.10.2 G-density for rcode $k > 3$ rules

If the rules specified are not totalistic rules and if $k > 3$, the following prompt is presented,

G-density plot: number of rules (def 1250):

Enter the number of rules to be plotted or accept the default. The rules will be selected at random, but with a bias to include a representative distribution over values of Z.

24.11 G-density plotted against system size

The rate of increase of G-density with system size n is another order-chaos indicator in CA, with ordered dynamics showing the highest increase[27]. Chaotic rules approximate a zero rate – only the maximally chaotic chain-rules show G-density decreasing with increasing n (figure 24.13.

Enter **3** in section 24.9 to plot G-density against a range of n. If required, this can be done for a number of selected rules on the same plot. The following prompt is presented,

G-density - size range graph
start size (def=5): **end size (max=31):**

Enter the start and end values of n. Note that if the end size is set high, computation time may be inordinately long.

Enter **d** at the next prompt (optional) which normally skips all further options, but in this case skips to mutation (see chapter 28). The plot will be drawn in the lower right of the screen. The basin field will be draw (at a small scale by default) in a window in the upper part of the screen. When each attractor basin is complete the G-density (y-axis) is plotted against the system (or network) size n, and the plot will continue automatically for the next value of n. Once complete, the following prompt appears in the top-right data window,

quit-q rule-r (mut-def) keep+k:

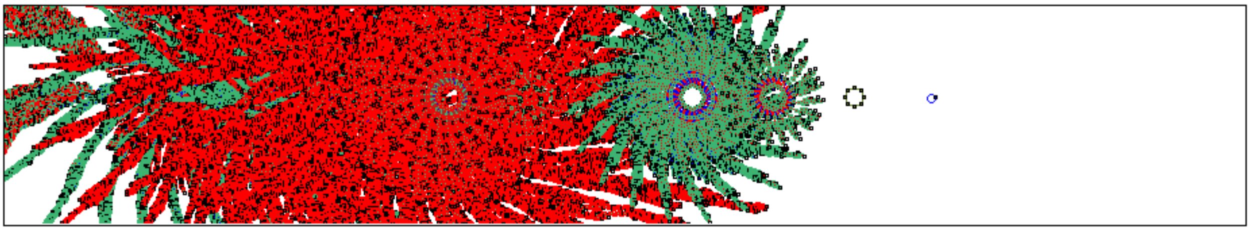


Figure 24.12: As the G-density - system size plot is in progress (and also plotes in section 24.9), the attractor basins are drawn in an upper window at a default scale and layout (which can be changed in chapter 25). This example shows the CA basin of attraction field for rule 9569a999, n=20, to plot the final point in figure 24.13.

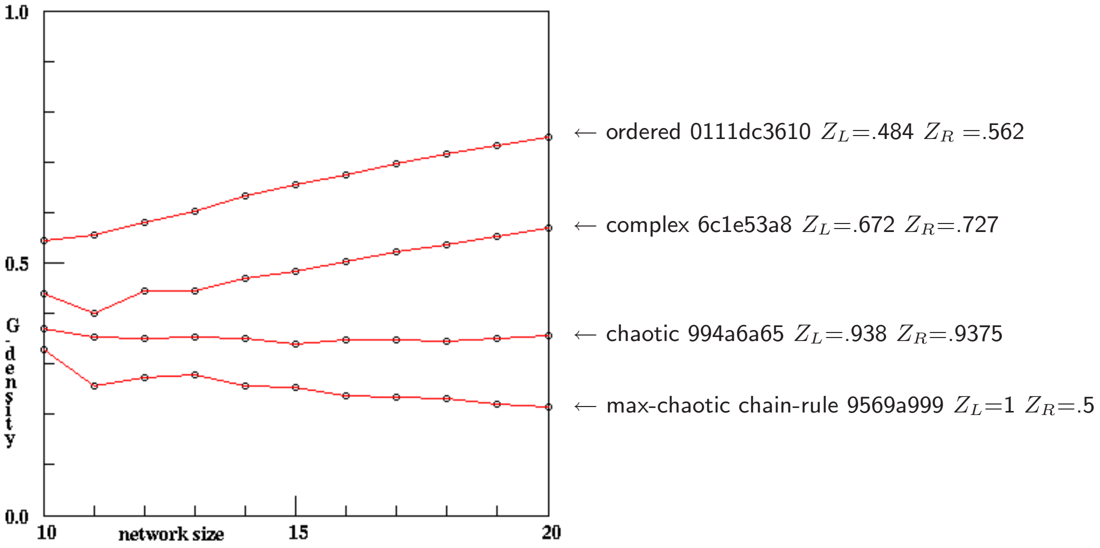


Figure 24.13: Plotting G-density against a range of n for ordered, complex and chaotic k=5 CA rules (indicated), where the ordered rule has the highest G-density rate of increase. Chaotic rules approximate a zero rate – only chain-rules show G-density decreasing

Enter **r** to select a new rule (set in a lower right window, chapter 16). To superimpose the new plot on an old plot enter or add **k** – i.e. for a enter **rk** for a new rule, or just **k** for a mutant. Otherwise enter **return** for the next mutant – the rule will change according to the type of mutation set in section 28.1. Enter **q** to quit the G-density - n plot, and backtrack.

24.12 Z - λ_{ratio} plot

Enter **L** in section 24.9 for a plot of a Z (x-axis) against λ_{ratio}, or Z_{left} against Z_{right}, for a predefined set of rules. The plot is located in the lower right part of the screen. In addition, the proportions of rules in the rule sample with different levels of λ_{ratio}, the P parameter, and canalizing inputs may be displayed once the graph is complete (section 24.13). The measures are computed directly from rule-tables.

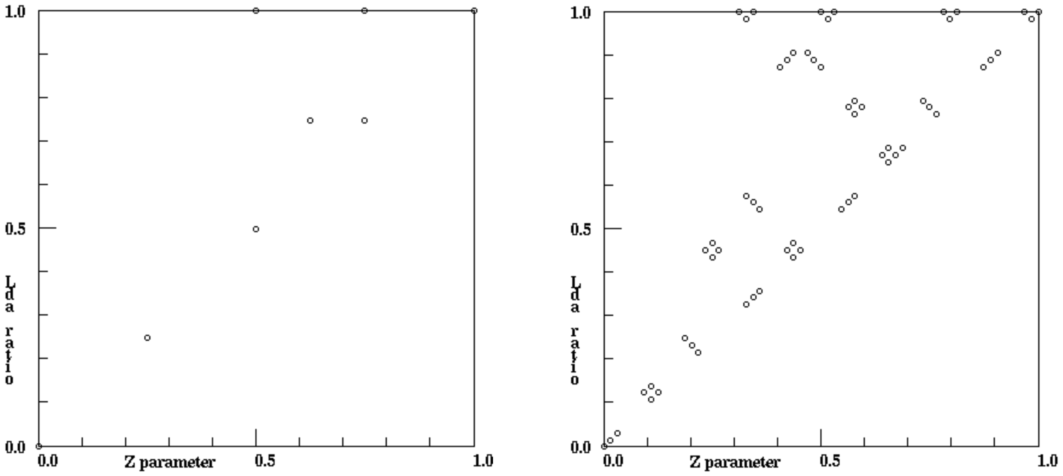


Figure 24.14: λ_{ratio} plotted against Z for Left: all 256 $v2k3$ rcodes, and Right: all 256 $v2k7$ tcodes in 72 clusters.

The following prompt is presented,

plot: Zleft-Zright-1, Zpara - Lda ratio and C-P data-(def):

Z_{left} and Z_{right} are components of the Z parameter. Z itself is greater of the two[19]. Whichever plot is selected, the prompts that follow are similar.

Enter **return** for a Z - λ_{ratio} plot, or **1** for a Z_{left} - Z_{right} plot.

24.12.1 Z - λ_{ratio} or Z_{left} - Z_{right}

There are two methods of predefining the rule set,

countdown ... from the (start) rule selected in section 16. If the start rule is all 1s (or the maximum decimal rule number 2^k-1) the entire rule-space will be included, but computation can be lengthy for large k. For tcode[4] and rcode $k \leq 3$ this is the only method. For rcode $k \leq 3$ the plot will proceed without further options. Countdown is also available for k=4 and k=5 rcode.

sample ... a random sample of rules which can be biased (or not) to cover a representative spread of λ_{ratio} and Z – for rcodes $k \geq 4$.

24.12.2 Z - λ_{ratio} or Z_{left} - Z_{right} for $k \geq 4$ rcode

If the rules specified are k=4 or k=5 rcode, the following prompt allows the rule sample size to be specified,

lambda-Z graph: countdown-C, number of rules (def 2200):
or
Zleft-Zright graph: countdown-C, number of rules (def 2200):

[4] For binary rules, v=2, tcode and kcode are identical, but for these options to work use tcode.

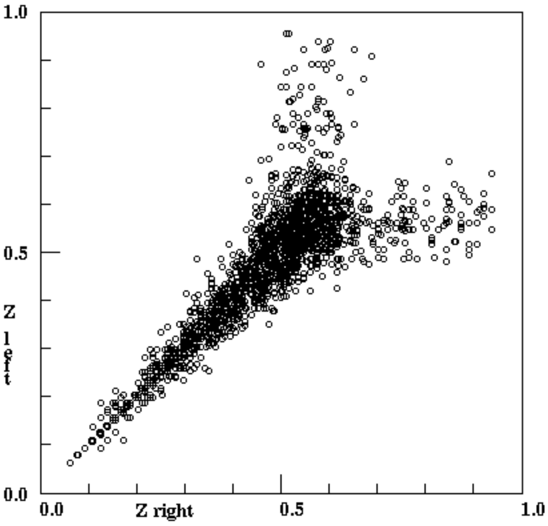


Figure 24.15: Z_{left} plotted against Z_{right} for 2200 $k = 7$ rules, selected at random but with a bias to cover a representative spread of Z.

Enter **C** to countdown from the start rule entered in section 16. If the start rule is all 1s the entire state-space is included – this may take some time for k=5.

For $k \geq 6$ the countdown option is omitted because rule-space becomes too big. Enter a number to set the sample size. By default the rules will be selected at random, but with a bias to include a representative distribution over values of λ_{ratio} and Z. The following further option allows an unbiased random sample.

random rule sample will have lambda bias, dont bias-n:

An unbiased sample is useful in estimating the proportion of rule-space with different measure of the λ_{ratio}, P-parameter and canalizing inputs. Once the plot is complete, a lower window prompts for these details, described in section 24.13.

24.12.3 Z - λ_{ratio} or Z_{left} - Z_{right} for tcode

For tcode only, The following prompt is presented, allowing "equivalent" tcode to be skipped,

show only equivs-e all-a (def clusters): *(tcode only)*

All these alternatives perform a decimal countdown of tcodes from the start tcode selected in section 16. Because each tcode must be transformed to rcode, computation can be lengthy for large k. These options allow skipping (and to seeing) equivalent rules.

Enter **a** to compute all rules (without skipping). Enter **e** to show one representative tcode from each set of equivalent tcodes. The default is one tcode from each cluster[5]. A further prompt allows a pause to note the equivalents of members of the tcode cluster,

pause to show codes-p:

[5] There are at most 2 equivalent tcodes, the start tcode and its negative (the reflection of a tcode = identity), and at most 4 tcodes in a cluster which includes the compliments of both equivalents[19] Rules in a rule cluster have the same measures of Z, λ_{ratio} (and also G-density) so the resulting plots will be identical.

If **p** is entered above, after each point is plotted, data about equivlent rules and clusters will be displayed in a top-right window, for example for $k = 7$ tcodes,

7-code:233=104 22=151 equiv=2 clust=4 tot=22 cont-ret:

This signifies that codes 233 and 104 are equivalent, as are their compliments, codes 22 and 151. These 4 codes make up a cluster. In this example, the number clusters plotted so far is 22.

Once the plot is complete, for example for the rcodes in figure 24.14, a top-right window gives information about canalizing frequency, for example,

c-frequency:136 78 24 18 total rules=256 last=256

This shows the frequency of the 4 possible inputs (0, 1, 2, 3) – because k=3. A lower window prompts for more detailed parameter frequency information, described in section 24.13.

24.13 Proportions of canalizing inputs and λ_{ratio}-P

Once the Z - λ_{ratio} or Z_{left} - Z_{right} plot in section 24.12.1 – 24.12.3 is complete, data on the proportions of rules in the rule sample with different frequencies of "canalizing inputs", and the equivalent measures λ_{ratio} and P, can be displayed. The following prompt is presented in a small window to the left of the plot,

tot plotted=20000 *(for example)*
C-P data-d, +print-p: *(***+print-p** *not for DOS)*

Enter **d** to see the data table displayed across the top of the screen, or **p** to also print the data to the xterm window (not for DOS) as in tables 24.1 and 24.2.

```
Canalyzing-lambda ratio and P frequency count, total rules=256
C0=136=53.125% C1=78=30.469% C2=24=9.375% C3=18=7.031%  Cr=120=46.875%, Cin=180=23.438%
ldr  0/4       1/4       2/4       3/4       4/4
ldr  0         0.25      0.5       0.75      1
P    1         0.875     0.75      0.625     0.5
freq 1=0.391%  17=6.641% 56=21.875% 112=43.750%70=27.344%
C=0  0         0         8=3.125%  64=25.000% 64=25.000%
C=1  0         0         24=9.375% 48=18.750% 6=2.344%
C=2  0         0         24=9.375% 0         0
C=3  1=0.391%  17=6.641% 0         0         0
```

Table 24.1: Canalizing, λ_{ratio} and P data for k=3 rule-space, 256 rules.

```
Canalyzing-lambda ratio and P frequency count, total rules=65536
C0=94.638% C1=3080=4.700% C2=336=0.513% C3=64=0.098% C4=34=0.052%  Cr=3514=5.362%, Cin=4080=1.556%
ldr  0/8       1/8       2/8       3/8       4/8       5/8       6/8       7/8       8/8
ldr  0         0.125     0.25      0.375     0.5       0.625     0.75      0.875     1
P    1         0.9375    0.875     0.8125    0.75      0.6875    0.625     0.5625    0.5
freq 1=0.002%  33=0.050% 240=0.366% 1.709%   5.554%    13.330%   24.438%   34.912%   19.638%
C=0  0         0         16=0.024% 416=0.635% 3.918%   11.963%   23.755%   34.717%   19.626%
C=1  0         0         64=0.098% 512=0.781% 1.562%   896=1.367% 448=0.684% 128=0.195% 8=0.012%
C=2  0         0         96=0.146% 192=0.293% 48=0.073% 0        0         0         0
C=3  0         0         64=0.098% 0         0         0         0         0         0
C=4  1=0.002%  33=0.050% 0         0         0         0         0         0         0
```

Table 24.2: Canalizing, λ_{ratio} and P data for k=4 rule-space of 65536 rules.

The key to data in tables 24.1 and 24.2 is as follows,

data heading . . . *what it means*

C0=, C1=, C3=, ... the number and percentage of the k+1 possible canalizing levels found – numbers shown if less than 9999.

Cr . . . the number and percentage of rules with at least 1 canalizing input.

Cin . . . the number and percentage of canalizing inputs in the rule sample.

ldr (first line) . . . the 2^{k-1}+1 possible values of λ_{ratio} as a fraction.

ldr (second line) . . . as above shown in decimal. λ_{ratio}= the fraction 0s or 1s (whichever is less) relative to half rule-table size, λ_{ratio} varies from 0 (order) to 1 (chaos).

P . . . as above showing the equivalent values of the P-parameter. P is the fraction of 0s or 1s in the rule-table (whichever is more), P varies from 1 (order) to .5 (chaos), and is related to λ_{ratio} as follows, $P = 1 - (\lambda_{ratio}/2)$

freq . . . the number and percentage of rules that were found in each P or λ_{ratio} category above.

C=0, C=1, C=2, ... the number and percentage of each canalizing level that was found in each P or λ_{ratio} category above – numbers are only shown if less than 999.

24.14 "Backwards" space-time patterns

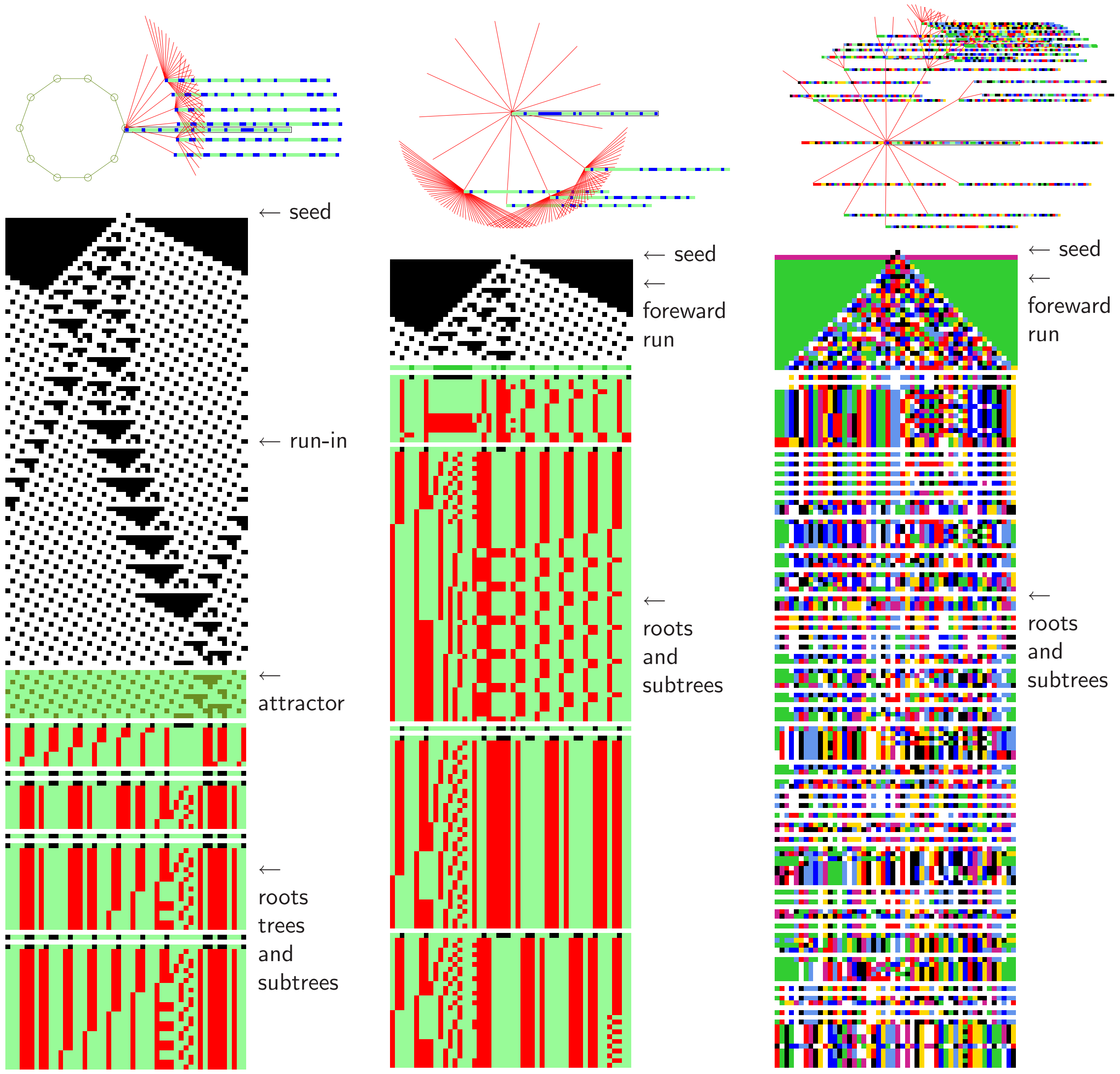
At the same time that attractor basins (state transition graphs) are being drawn, "backwards" space-time patterns may be displayed on the left of the screen, if selected in section 24.4 – figure 24.16 shows examples.. The sequence of events reflect the way basins and subtrees are computed. For basins of attraction, there will be three distinct phases in the space-time patterns,

1. The run-in from the initial seed state, which includes the transient and attractor cycle, disclosed from the first repeat.

2. The attractor cycle itself is redrawn (as it is being recorded).

3. Then the space-time patterns for drawing each transient tree, starting from the "root" and back through successive levels of the tree. After a gap in the pattern, the root of each tree (or subtree) is shown, followed by its immediate pre-images. Trees are computed from each attractor root state in turn (omitting the pre-image on the attractor), cycling anti-clockwise. The set of pre-images of attractor states must exclude the pre-image on the attractor itself to avoid infinite regress.

For a subtree phase 1 and 3 are similar, but here the run-in phase is replaced by an optional forward-run phase (section 29.2) and there is no attractor. A forward-run (for a given number of time-steps) is usually necessary (other than for maximally chaotic "chain" rules – section 16.11) because most of state-space consists of leafs (garden-of Eden) – a leaf state has no pre-images thus no subtree.



(a) basin of attraction $v2k5$ (b) subtree $v2k5$ (c) subtree $v8k3$

Figure 24.16: "Backwards" space-time patterns (n=50) from a singleton seed, also showing (at top) the basin or subtree state transition graph with the bit/value patterns at nodes. (a,b) $v2k5$ rule (hex)e0897801. (a) shows a basin of attraction, (b) a subtree from a state 22 time-steps forward from the original seed. (c) $v8k3$ modified chain rule, showing a subtree from a state 25 time-steps forward from the original seed. For v=2 only, colors distinguish distinct phases: run-in or forward run, attractor, subtree root, and its preimages. The figures show just the start of the backward run, which may continue for an indeterminate length of time.

The initial default is not to show the backward space-time patterns – the following prompt is presented,

backwards space-time patterns (now OFF) tog-s: *(or* **now ON***)*

Enter **s** to toggle the current setting.

24.14.1 Scroll space-time patterns

By default, space-time patterns (both forward and backward) will scroll vertically. The pattern can also be presented in successive vertical sweeps, restarting at the top when the screen is full, which is somewhat faster than scrolling. If space-time patterns were set in section 24.14, the following prompt is presented,

sweep backwards space-time pattern-s, scroll-def:

Enter **s** to sweep, or **return** to scroll. Scrolling can also be toggled on/off on-the-fly (section 24.16).

24.15 Basin speed

Drawing drawing basins and backwards space-time patterns may generate too fast for some purposes, such as the key hit on-the-fly options in section 24.16 where slowing down is done with the < key – so slowing down may need to be preset.

The "basin speed" option in section 24.4 has the following top-right prompt,

basin speed: max-(def), or slowmotion (try 300):

Enter a number to slow down, the higher the slower. Maximum speed can be restored on-th-fly with the > key. The same slowdown option can be reached while interrupting (section 30.2), and from the "attractor basin complete" prompt (section 30.4). An addition/alternative to slowmotion is to pause backward itteration after each tree or each pre-image fan (section 27.3).

24.16 Basin on-the-fly options

While attractor basins and backwards space-time patterns are being drawn, a number of settings can be changed on-th-fly. The following reminder will appear in the bottom title bar,

matrix-m STP:tog/scroll/exp/contr-s/#/e/c slow/max-</>

These changes take effect as soon as the key is pressed, without **return**, and apply whether or not preset in the relevant sections in this chapter.

on-the-fly key ...	*what it means*
matrix-m ...	to toggle the state-space matrix on/off (section 24.5).

STP:tog-s . . . to show/hide backwards space-time patterns (section 24.14).

STP:scroll-# . . . to toggle between scroll and sweep of backwards space-time patterns, (section 24.14.1).

STP:exp/cont-e/c . . . press **e** to expand and **c** the scale of backwards space-time patterns and basin nodes if shown as a bit/value pattern (section 26.2) – the scale can be set in section 26.2 and the scale can be preset in section 26.2.4

slow/max-</> . . . each time < the drawing speed is halved – both basins and backwards and space-time patterns. Enter > to restore maximum speed.

Chapter 25

Layout of attractor basins

Basins of attraction are drawn either singly, or in groups – for the basin of attraction field, for a range of network size n, or for mutant single basins. Groups of basins are drawn in successive rows starting from a top left position. Before drawing starts, the following layout parameters may be altered (or defaults accepted): their scale, position, angle, horizontal and vertical spacing, rows sraggered, spacing increase (for a range of network size n), and the right border before the next row starts. A basin of attraction field may be displayed as successive single basins as well as in a group. Initial default layout values depend on the type of attractor basin selected. Once revised, the new values generally become defaults, but may be reset to the original values.

If **l** is selected at the first "output parameter" prompt in section 24.1 (or if the output parameter prompts are viewed in sequence) a layout preview for attractor basins will appear, together with the first layout prompt described below (section 25.2). Some layout parameters may also be varied on-the-fly, after each subtree or basin is drawn (section 25.3).

The basin of attraction field can also be redraw within the attractor jump-graph window, according to the jump-graph layout (section 20.7) providing many additional layout possibilities to those described in this chapter.

25.1 The layout preview

The layout preview shows the spacing and scale of attractor basins as they would appear on the screen according to the current default settings, and appears in the top left quadrant of the screen, in a window representing the screen at 1/2 size. Polar coordinates are used to scale and position basins, and these are indicated. x and y axes divide the screen with coordinates 0,0 at the center. At the left of the screen x= -100, at the right x= +100. At the top of the screen y= +100, at the bottom y= -100.

Basins are represented by concentric circles, where the inner circle is the maximum attractor diameter. Successive circles indicate successive levels. The first basin representation shows the first 90 levels where the distance between successive levels decreases asymptotically (thereafter the distance between levels remains constant). In the first basin every 20th circle is highlighted, as well the attractor circle itself and the first 3 levels. Other basin representations show the attractor circle and the first 3 levels only. If a single basin or subtree was selected (without a range of network size n) only one preview basin is shown. Otherwise a representative number of basins are shown to anticipate the layout (figures 25.1 – 25.2).

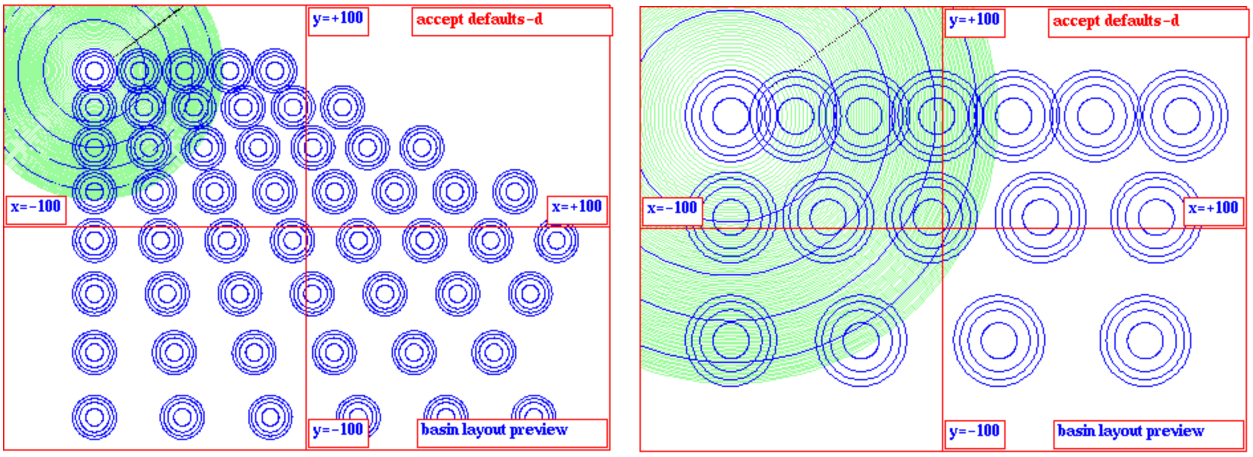


Figure 25.1: The default layout preview for a single basin or subtree (left), and for a range of single basins (right).

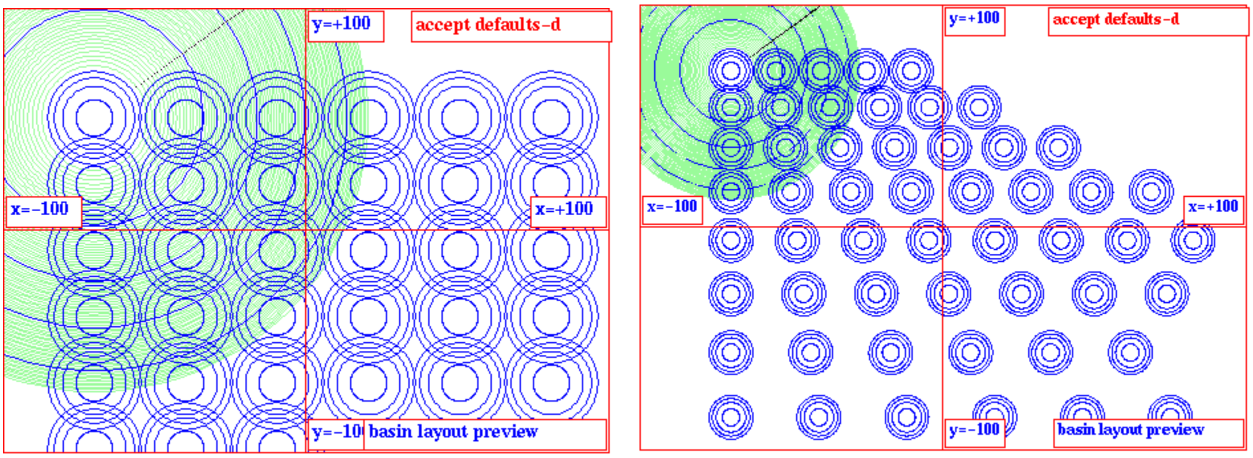


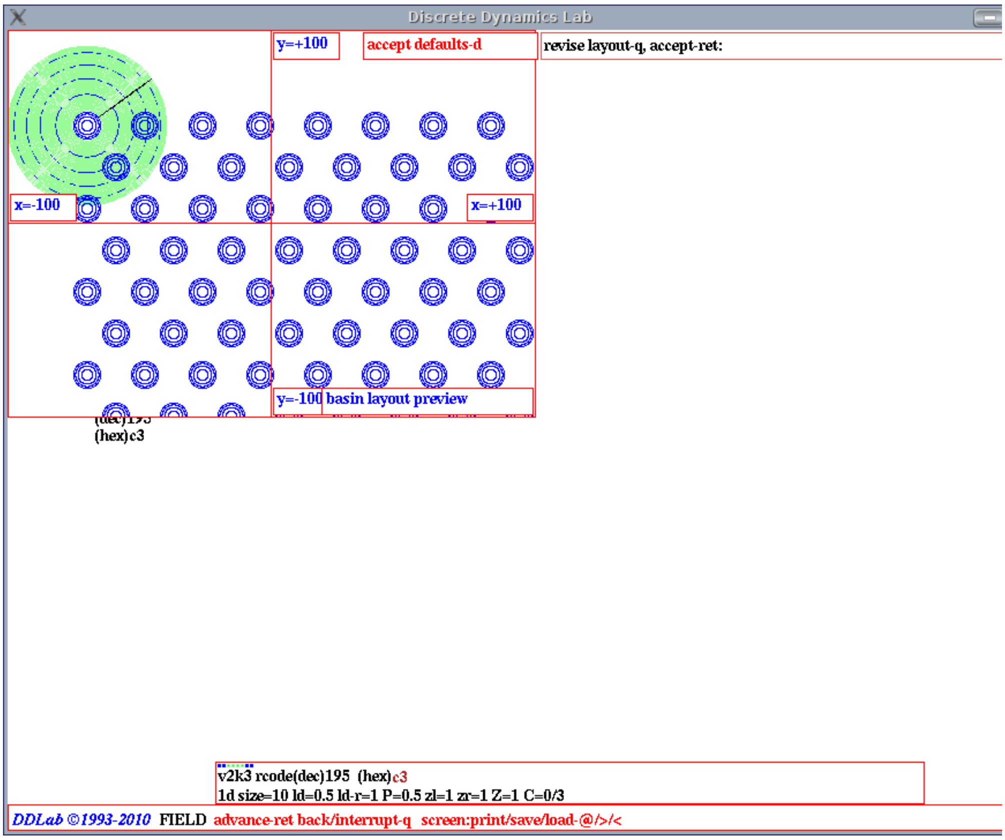
Figure 25.2: The default layout preview for a basin of attraction field (left), and for a range of basin of attraction fields (right).

25.2 The first layout prompt

Enter **l** at the first output parameter prompt in section 24.1 to skip directly to the preview for the layout of attractor basins (section 25.1). Alternatively these options can be viewed in sequence.

The layout preview for attractor basins will appear (described above, section 25.1), together with the following (top-right) first layout prompt ,

> **layout: reset defaults: all-r mutants-M** *(***M** *for single basins only)*
> **select att rad, % of 1/2 screen, x-axis (min 0.24 def 6):**
> *(min depends on screen resolution, the default depends on previous selections)*



accept defaults-d

revise layout-q, accept-ret:

(dec)195
(hex)c3

v2k3 rcode(dec)195 (hex)c3
1d size=10 ld=0.5 ld-r=1 P=0.5 zl=1 zr=1 Z=1 C=0/3

DDLab ©1993-2010 FIELD advance-ret back/interrupt-q screen:print/save/load-@/>/<

Figure 25.3: The layout preview as it appears on the DDLab screen. This examle is for a basin of attraction field, rad=2, start position: x=-70, y=50, spacing: x, y, and right border=22, with staggered rows.

25.2.1 Reset all layout defaults

Enter **r** in section 25.2 to restore all layout prompts to their original settings. These default settings depend on the type of attractor basin selected. A revised layout preview will be displayed.

25.2.2 Mutants for single basins on one screen

For single basins only, enter **M** in section 25.2 for a special default layout appropriate for displaying successive mutants on one screen – for example to display the set of one bit mutants of a rule, with the original basin shown by itself in the top row (section 28.3.1 and figure 28.3).

25.2.3 Basin scale, attractor radius

The first layout prompt (section 25.2) gives the current attractor radius, scaled according to the x-axis. In the example, **(def 6)** means 6% of 1/2 of the screen width. To alter the default basin radius controlling the scale of attractor basins, enter a new radius – allowing up to two places after the decimal point. The minimum radius allowed depends on the screen resolution and will correspond to a scale of one pixel. A small scale is somtimes required for basins with very long transients.

25.2.4 Basin start position

To alter the default position of the first basin, enter **return** in section 25.2. The following top-right prompt is presented,

> **start position, % offset from screen centre**
> **x (def -70): y (def 50):** *(values shown are examples)*

Enter the new polar coordinates for the first basin, first x, then y. Remember that the screen center is at 0,0 and the edges of the screen are as follows, left x= -100, right x= +100, top y= +100, bottom y= -100. A position outside the screen limits is permitted[1], allowing, for example, the relevant part of a large basin to be displayed.

25.2.5 Show the field as successive basins

FIELD-mode only

In FIELD-mode (but not for a *range* of n in section 8.1), a further option is presented (in the same window as section 25.2.4) to display each basin in the basin of attraction field one by one – successive single basins at the same start position.

> **show field as successive basins-s, and pause-p:**

Enter **s** to show successive basins without pausing. Enter **p** to pause after each basin – data on each basin (and other options) is will be displayed (section 27.5.3) – enter **return** will generate the next basin.

25.2.6 Basin spacing and stagger rows

FIELD-mode only

To alter the default spacing between basins in a basin of attraction field (see figure 25.2 left), the following prompts are presented in sequence:

> **basin spacing, % of 1/2 screen** *(defaults depend on the radius set in section 25.2.3)*
> **x spacing (def 30): y spacing(def 30) tog stagger rows (now off)-s:**

Enter the new spacing according to the polar coordinate scale, first x, then y. Remember that the screen is 200 units wide and 200 units high. Lastly, enter **s** to toggle the staggering of alternate rows, as in figure 25.3

25.2.7 Select minimum right border width

FIELD-mode only

To alter the default minimum right border width, enter the new border width as a percentage of the current x-spacing. If the distance between the center of the next basin and the right border is less than this setting, the basin will be drawn at the start of the next row. Negative values for the right border are permitted. The following top-right prompt is presented,

[1] The centre of attractors can be located up to one screen height below the bottom screen edge. Any attractors that would be located below this limit are repositiond on the limit, and their basin transients are not drawn.

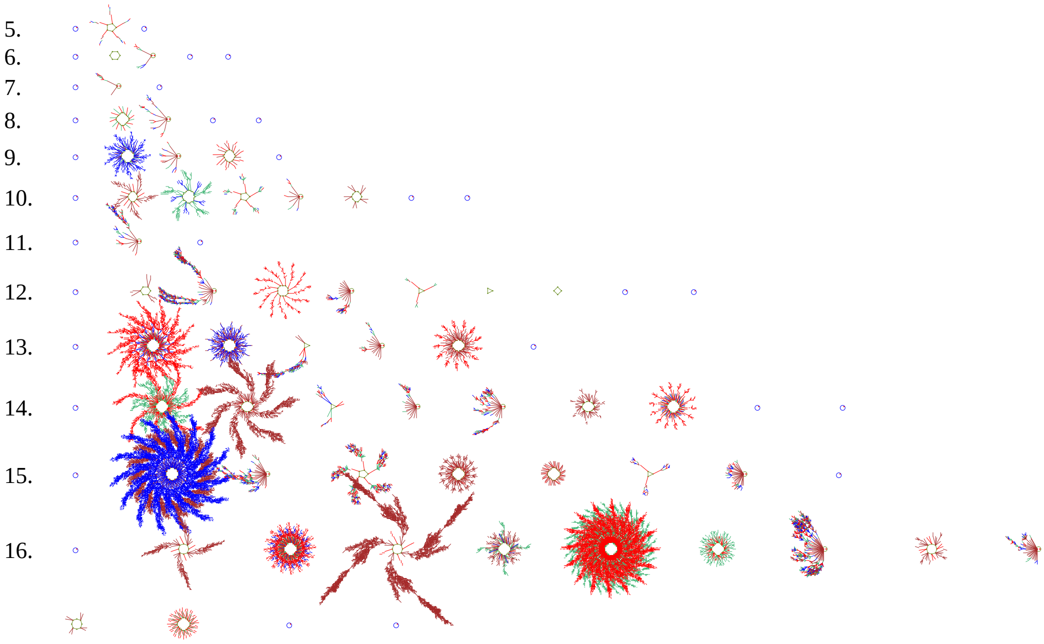


Figure 25.4: The basin of attracion field for a range of network size, n=5–16, shown on the left, $v2k4$ rcode (hex)8b26. The spacing between basins increases at a set rate as larger basins are expected for greater n. In this example, the basin scale is 1.2 (see section 25.2.3), the x, y start position is -90,70 (see section 25.2.4), the initial x, y spacing belween basins is 6,6 (see section 25.2.6), the right border width is 45% (see section 25.2.7), and the x, y spacing increase is set at 12%.

> **select right border as % of x spacing (default 45.0%):**
> *(the default right border depends on the spacing set in section 25.2.6)*

25.2.8 Amend the spacing increase for a range of n

If a range of network size n was selected in 8.1, the spacing between successive basins (and fields) is set to increases by a default percentage which may be amended (see figure 25.2 right). The increase is usually required because the size of basins tends to increase with greater n. The following prompt is presented,

> **% increase in spacing for successive network size,**
> **x increace (def 10%): y increase (def 10%):** *(values shown are examples)*

Enter the new percentage increase, first for x (for the horizontal spacing), followed by y (for the vertical spacing). This becomes the new default. An example of a range of basin of attraction fields is shown in figure 25.4 where the default setting were slightly amended, and in figure 4.5.

25.2.9 Accept or revise layout parameters

Finally, the layout preview showing the current settings (as amended) is displayed, with the following prompt to revise or accept,

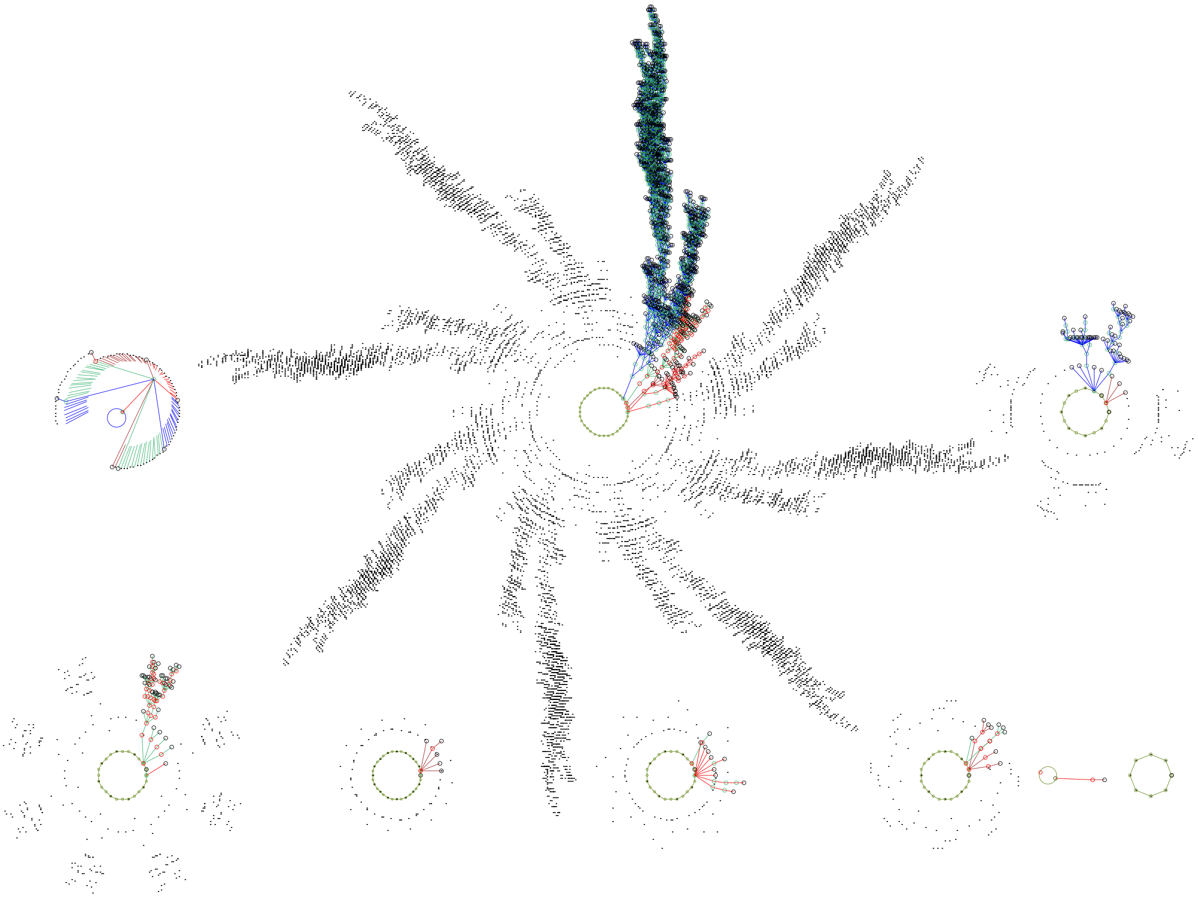


Figure 25.5: Amending the layout of a basin of attraction field on-the-fly, for a CA $v2k3$, rcode (dec)110, n=16. The field is shown with copies of equivalent trees suppressed exept for garden-of-Eden nodes shown as dots (section 26.1.3). Layout parameters were set as follows: basin scale 3.8 (section 25.2.3), x, y start position -75,13 (section 25.2.4), initial x, y spacing belween basins is 65,73 (section 25.2.6), right border width 15%, (section 25.2.7). A pause after each basin was set (section 27.3) allowing the x spacing to be changed – after the 4th basin to 40, after the 7th basin to 15.

layout: revise-q, accept-ret:

Enter **q** to revert to the first layout prompt (section 25.2) and re-adjust settings, or **return** to accept the layout and continue.

25.3 Amend the layout during pause

If one of various "pause" options is selected in section 27.3, for a basin of attraction field or range of fields, layout options become available after each basins (or field) is draw, including the angular orientation, subtree fan angle, and the spacing and position of each successive basin. At the pause, data will be displayed in a top-right window (chapter 27), and one of the following prompts will be presented in a small top-centre window (section 27.4) as follows,

next field-ret nopause-q ops-o layout-a: *(for a field-pause*
next basin-ret nopause-q ops-o layout-a: *(for a basin-pause*

Enter **a** for the layout options (for others see section 27.4) – the following top-right prompt,

amend angles-a next pos-p just spacing-ret:

These options are described in sections 25.3.1 to 25.3.4 below.

25.3.1 Amend the orientation and fan angle during pause

Enter **a** in section 25.3 to amend the basin orientation, and the "fan angle" – the spread of in-degree edges at nodes. Further prompts will be presented in turn,

change basin orientation (now 0), enter angle:
change pre-image fan angle(now 1.00), enter factor: *(values shown are examples)*

See section 26.3 for further details about these settings.

25.3.2 Amend the spacing and right border during pause

If **return** is entered in section 25.3 to amend the spacing (and right border). Prompts are presented as described in section 25.2.6 and 25.2.7 above. Figures 25.5 and 27.1 give examples.

25.3.3 Amend the next position during pause

Enter **p** in section 25.3 to amend the next position (as well as the spacing). Prompts are presented similar to those described in section 25.2.4 above,

(start= -80,50) amend next pos, % offset from screen centre
x (def -50.2): y (def 50.1): *(values shown are examples)*

The last start position is shown as a reference. Enter the next position's polar co-ordinates, first x, then y. Remember that the screen centre is at 0,0 and the edges of the screen are as follows, left x= -100, right x= +100, top y= +100, bottom y= -100. A position outside the screen limits is permitted. See section 25.2.4 for further details about these settings. Figures 25.5 and 27.1 give examples.

After the new position is set the new x co-ordinate becomes the new default x start position for future basins, the default y co-ordinate is not affected.

When this option is complete, the spacing options 25.2.6 and 25.2.7 are presented.

25.3.4 Amend the spacing increase during pause

If a range of network size n was selected in 8.1, the spacing increase between successive basins can be amended as described in section 25.2.8.

Intentionally Blank

Chapter 26

Display of attractor basins

This chapter describes various options to change the default display of attractor basins. Enter **p** at the first "output parameter" prompt in section 25.2 to jump directly to the display prompts, or reach them by viewing the output parameter in sequence. For regular 1d or 2d CA, attractor basins can be "compressed" (section 26.1). In this case the first prompt is in section 26.1.1. Otherwise the first prompt is in section 26.2.

26.1 Compression of equivalent CA dynamics

If the network is a regular 1d or 2d CA (i.e. CA wiring and just one rule), rotational symmetries of the periodic array result in equivalent dynamics [19]. A given network state (periodic pattern) A is embedded in a particular past and future made up of other connected network states. If A is translated around its circle (1d) or torus (2d) by an arbitrary number of moves, and transformed to state B, the states in B's past and future must be identical transformations of the states in A's past and future. The two sets of states are rotational equivalents, and their connection topology is identical.

A complication arises because some states are made up of repeating pattern segments, for example 011011011011 has 4 repeating segments 011. If the repeating segment size=s, there will be only s rotational equivalent states irrespective of the network size n, whereas with no repeating segments (always the case if n is prime), the number of rotational equivalents is n. This becomes more complicated for a 2d torus were repeating segments exist in two directions, and errors may occur in the algorithm in this case. Compression in 3d has not been implemented.

Note also that states with a given repeating segment size s, cannot be upstream of a state with greater s. In an attractor cycle the value of s for all the states must be equal. The *uniform states* (all 0s and all 1s) with the shortest repeating segment size (s=1) are often powerful attractors.

For regular 1d and 2d CA, "compression" algorithms automatically come into play (unless disabled in section 26.1.1) to compute equivalent basins, transient trees, and subtrees from the uniform states, from each prototype. This considerably speeds up computation. For equivalent basins, only the prototype is shown. The display of equivalent trees and subtrees may also be suppressed to varying degrees.

26.1.1 Suppress compression

The first prompt for regular 1d or 2d CA allows compression (section 26.1) to be suppressed (or reinstated if turned off). The top-right prompt gives a reminder of the v, k and n (or $i \times j$ for 2d), and is presented as follows,

> **display: regular 1d CA, v2k3 size=10** *(for example)*
> **suppress compression in basins/fields is ON, off-s:** *(or . . .* **is OFF, on-s***)*

Enter **s** to suppress (or reinstate) the compression algorithms.

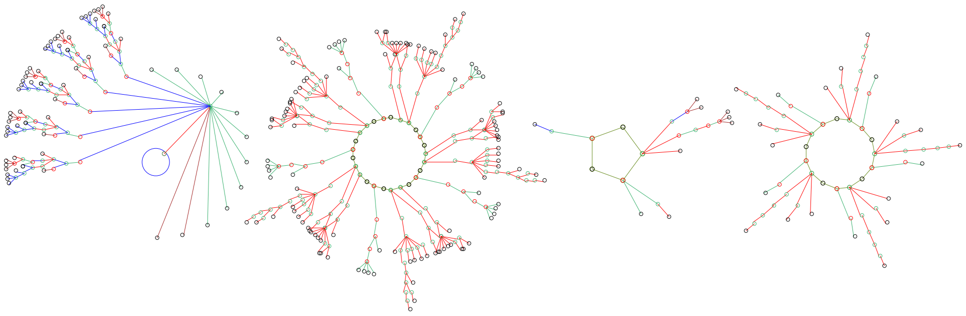


Figure 26.1: A basin of attraction field with compression of rotational equivalent dynamics, $v2k3$ rule (dec)110, n=10. Only one prototype of each set of equivalent basins is drawn. In addition, equivalent trees on attractor cycles (2nd basin), and subtrees from a uniform states (1st basin) are copied and rotated, thus further speeding up computation as a reverse algorithm need not be re-applied.

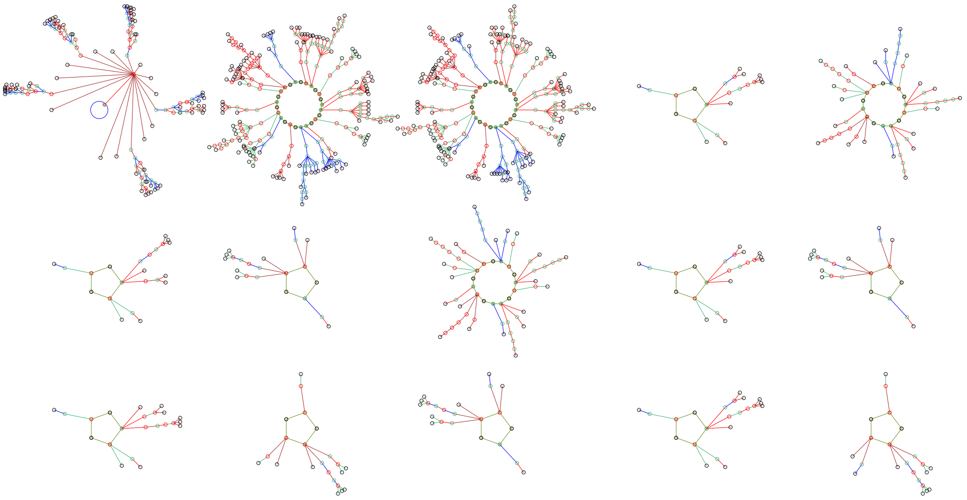


Figure 26.2: The same basin of attraction field as in figure 26.1 without compression of equivalent dynamics, so that all components of the state transition graph are computed from scratch with a reverse algorithm, not copied and rotated as in figure 26.1. All states in state space are represented.

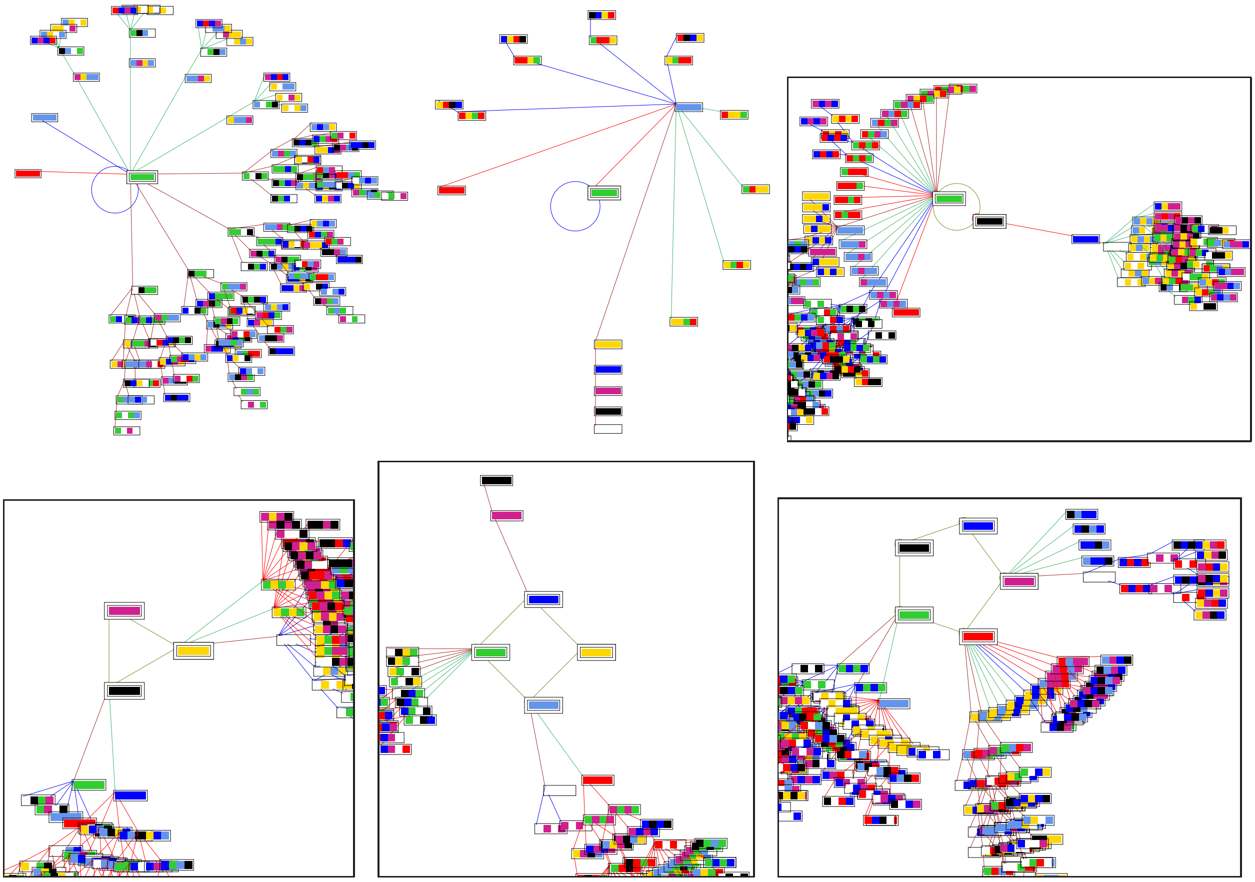


Figure 26.3: The special properties of CA uniform states, which lie on or very close to attractors, allow speeding up computation of their pre-image fan and and attached trees. Each (sub)tree type only needs to be computed once, the equivalent (sub)trees simply have their states rotated. The pre-image fan is organised into equivalent groups. This is illustrated here by some 1d CA $v8k2$ rules n=4, but the method applies equally to binary, v=2, or any value-range v. States are shown as v=8 value patterns – attractor states are highlighted by default inside a double outline. Attractor period range from 1 to 5. The framed basins are fragments centred on the attractor.

26.1.2 Pre-images of uniform states

The uniform states, where all cell values are equal, enjoy a privileged status in the dynamics of CA because they have maximum rotational symmetry (RS) so lie on or very close to attractors. Uniform states can only be pre-images of each other[1]. States with higher RS must be downstream of states with lower RS – all states in an attractor must have equal RS[19]. Take any pre-image of a uniform state – all its rotations must also be valid pre-images, so this property allows a method to speed up computation (if "compression" is on, section 26.1) as well as organising the pre-image fan and attached trees into equivalent groups. At present the method applies to uniform states on the attractor, or to a uniform state that is the unique pre-image of a point attractor (figures 26.3 and 26.8). In principle this could be extended to any uniform state, and also to states with repeating segments in general, but this generalisation is not implemented in DDLab at present.

[1] The property that uniform states can only be pre-images of each other applies to any system with a homogeneous rule – RBN and DDN as well as CA. However, the uniform state equivalent tree method only applies to CA.

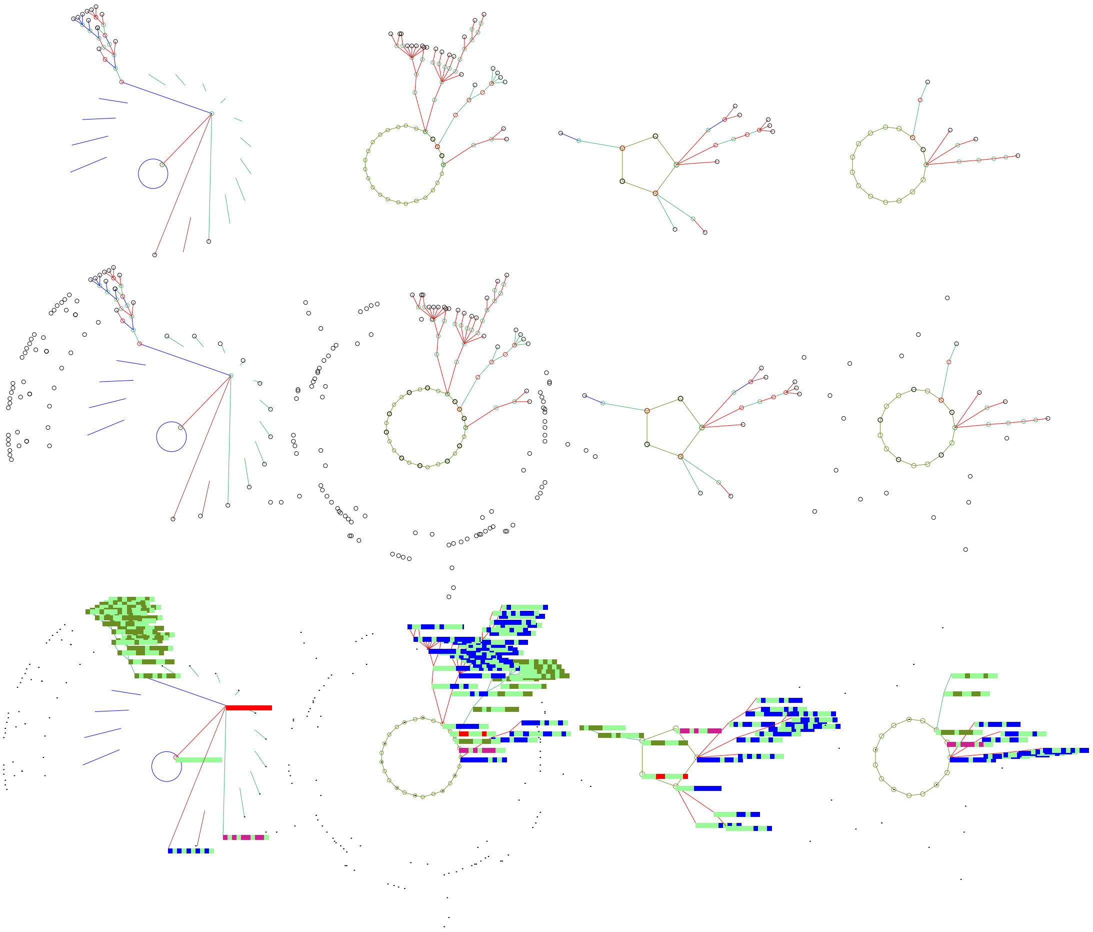


Figure 26.4: Suppressing equivalent subtrees in a a basin of attraction field, $v2k3$ rule (dec)110, n=10, (as in figure 26.1). *Top*: all equivalent subtrees suppressed, however, short edges are still shown from a uniform state (1st basin). *Centre*: leaf states in the suppressed subtrees shown as spots, and *Bottom:* as one pixel dots. Nodes in the prototype trees are shown as selected in section 26.2 – in the latter case as a 1d bit pattern.

26.1.3 Suppress copies of trees (and subtrees)

If compression remains set, a further option is offered to suppress copies of transient trees from attractor cycles, and subtrees from uniform states. However, the garden-of-Eden (g-of-E) or leaf nodes can be retained as spots/dots to indicate the footprints of the suppressed trasients (figures 25.5 and 26.4). The prototype tree/subtree is drawn in full with whatever node display set in section 26.2. The following top-right prompt is presented,

> **copies of trees (& subtrees from uniform states)**
> **suppress: all-3, show only g-of-E nodes: normal-2 small-1:**

Enter **3** to suppress all copies of equivalent trees or subtrees. This results in a clearer picture of crowded basins. Alternalively, to retain just the leaf nodes in the equivalent trees or subtrees, showing just their footprints, enter **2** for bold footprints, or **1** for subdued footprints where each leaf node is shown as one pixel.

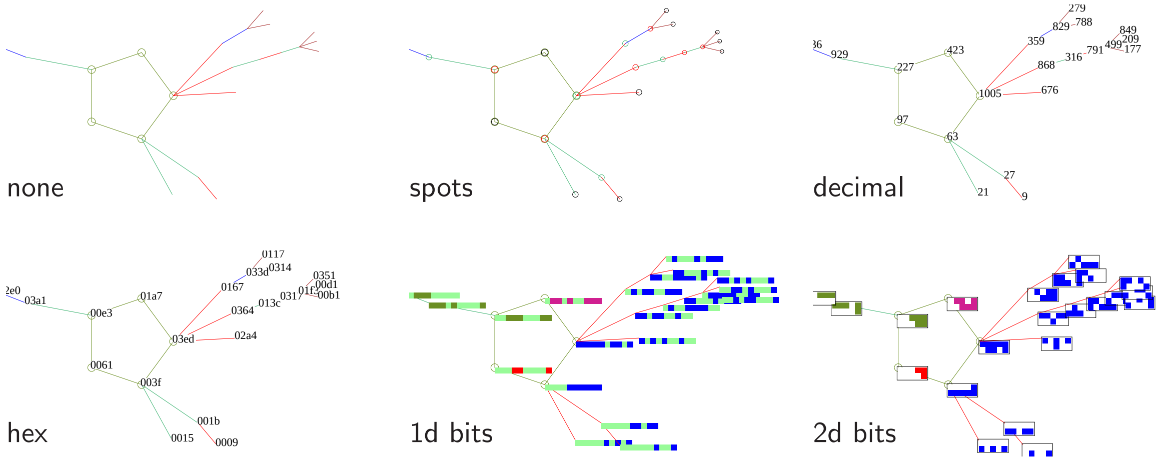


Figure 26.5: Examples of alternative node display. *from the top left*: none (except on the attractor): spots, decimal, hex, 1d bit pattern, and 2d bit pattern, where the $i \times j$ dimensions can be preset. This is the small basin in figures 26.1-26.4, $v2k3$ rule (dec)110, n=10.

26.2 Node display

The nodes in attractor basins may be displayed in a variety of ways (summarised below) as small circles or spots, as scalable bit/value patterns in 1d or 2d, as the hex or decimal value of the pattern, or nodes need not be shown. Whatever the selection, node display can be restricted to just rotationally symmetric states (figure 26.6), just leaf states, or states other than leaf states.

As well as the primary method of node display, attractor nodes can be independently highlighted as bit/value patterns. Pre-defined sets of nodes can also be highlighted in the "learning" routines described in chapter 34. A top-right prompt to set the node display (similar to the following) is presented, with options summarised below. The default depends on previous selections,

> **nodes: 2d-B 1d-b hex-h dec-c spot-s** *(***dec-c** *if applicable,* **2d-b** *for a 2d network)*
> **just Sym-nodes +S, just g-nodes +g, no g-nodes +G, none-n (def-s):**
> *(***def-** *shows the current setting)*

For networks other than regular 1d or 2d CA, this is the first prompt in the display sequence, preceded by the title **display:** – the prompts in sections 26.1.1 and 26.1.3 are omitted.

options ...	*type of node*
2d-B ...	as a 2d bit/value patterns for 1d (or 3d) networks).
2d-b ...	as a 2d bit/value patterns for 2d networks.
1d-b ...	as 1d bit/value patterns for for (1d or 3d networks).
hex-h ...	in hex (hexadecimal).
dec-c ...	in decimal (*(if applicable – section 21.6)*.
spot-s ...	as small circles or spots, cycling through 4 colors to contrast with transient edges colors, except garden-of-Eden states which are colored white.

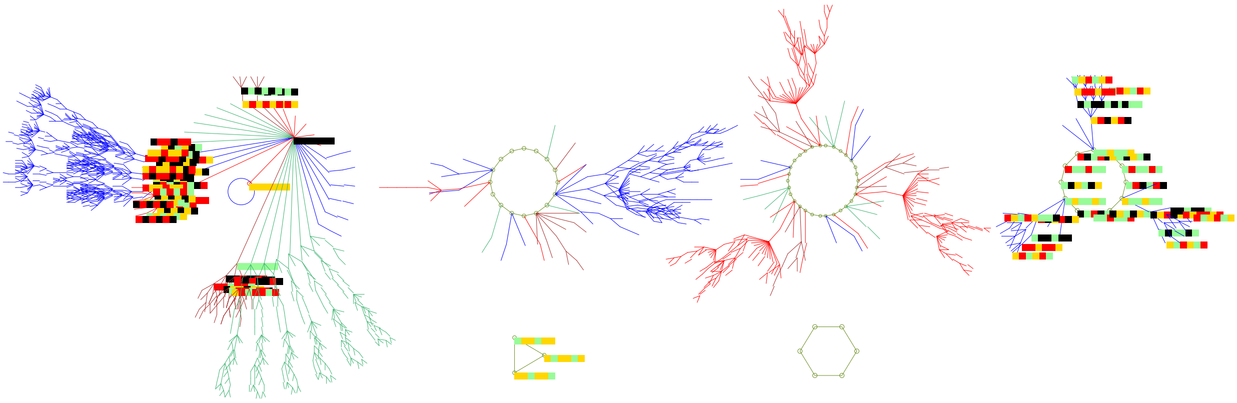


Figure 26.6: Symmetrical states only, those made up of repeating segments (including the uniform states) are shown as value-pattern nodes in a basin of attraction field – other nodes are suppressed. Symmetric states occur in basins 1, 4 and 5. [v4k3n6] rule (hex)4d3f8c86143ffca77a0ed4b5d172e7c3. This example has nodes as 1d bit patterns.

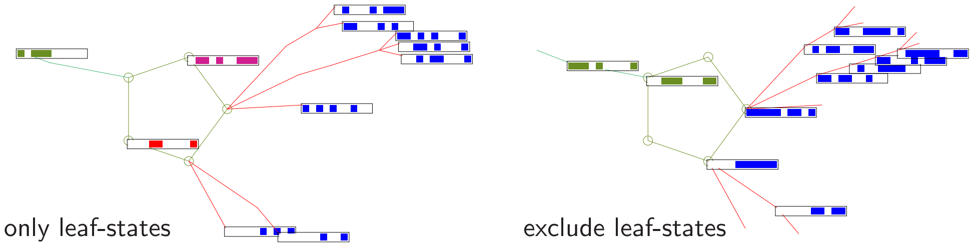


Figure 26.7: Showing just leaf-states or excluding leaf-states, $v2k3$ rule (dec)110, n=10, (as in figure 26.4). *Left*: only leaf-states and attractor states without transients are shown. *Right:* leaf-states and attractor states without transients, are excluded. This example has nodes as 2d bit patterns 10×1.

+options . . .	*to restrict node display*
just Sym-node +S . . .	add **S** for just rotationally symmetric states consisting of repeating segments, which include the uniform states. For example enter **bS** for figure 26.6.
just g-nodes +g . . .	add **g** for just leaf states (and attractor states without transients). For example enter **Bg** for figure 26.7*Left*.
no g-nodes +G . . .	add **G** to exclude leaf states (and attractor states without transients). For example enter **BG** for figure 26.7*Right*.
none-n . . .	to omit nodes altogether.

The options for node display can be combined with any other node options – compression (sections 26.1.1 – 26.1.3) and highlighting attractor and subtree root nodes (section 26.2.2).

The colors of spots cycle through four colours to contrast with transient edge colors (section 23.3.1). Decimal and hex numbers are shown black. For $v \geq 3$ nodes displayed as 1d and 2d value-patterns follow the default value colors (chapter 7), but for binary systems the colors of 1s vary to distinguish trees – described below (section 26.2.1).

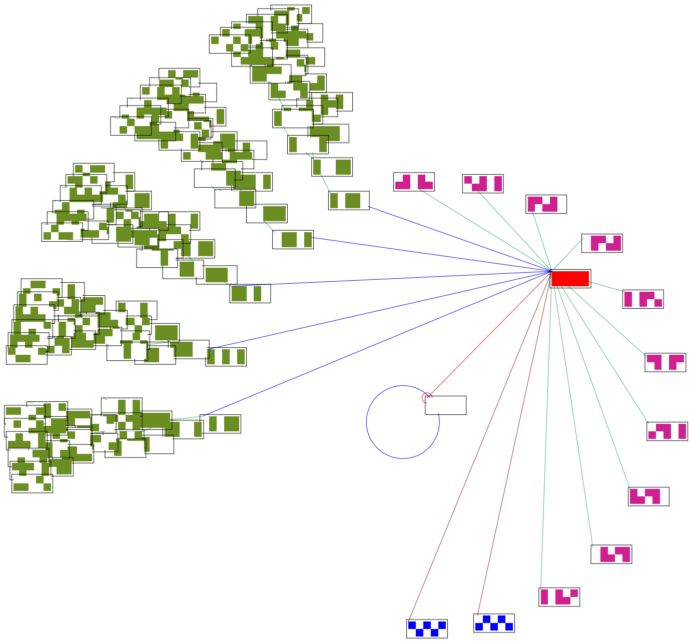


Figure 26.8: The pre-images of a uniform state's tree are organized into groups of equivalents. For binary systems, the bit patterns of these states, and states in their subtrees are shown in the same color, cycling through four colors. This example, with nodes as 2d bit patterns 5×2, is the first basin in figure 26.1, $v2k3$ rule (dec)110, n=10.

26.2.1 Node colors

When nodes are displayed as spots (the default), the colors of transient nodes are set to one of 4 colors contrasting with the colors of transition edges (section 23.3.1) where a cycle of four colors distinguishes transient trees for attractor periods of 6 or more. This is also the case for the bit patterns color of 1s in binary systems, whereas for $v \geq 3$ the default value colors apply (chapter 7). Leaf (garden-of-Eden) nodes as spots are black. 2d bit patterns have an outer border, 1d bit patterns do not.

In binary systems, nodes in the same pre-image fan are drawn in the same color (except for the uniform states, figure 26.8). If the attractor period is 5 or less, successive fans are assigned a different color so that a given transient tree may have a mix of colors. For attractor periods of 6 or more, all nodes in the same tree are assigned the same color, and the color is changed for the next non-equivalent tree. Note that with compression on (section 26.1) equivalent trees will be colored identically, and the pre-images of a uniform state's tree are organized into groups of equivalents. These states, and states in their subtrees, are shown in the same color, cycling through four colors.

26.2.2 Highlight attractor, or subtree root, in 2d

One or all attractor states, or the root state in a subtree, can be highlighted as a bit/value pattern in 2d, irrespective of the overall node display set in section 26.2. The following top-right prompt is presented,

highlight attractor (or subtree root) in 2d: one-1 all-a:

Enter **1** to highlight one state – knowing this state would allow that single basin to be regenerated. The state highlighted is the last attractor state reached in the initial forward run, and is positioned due east on the attractor cycle (figure 26.13).

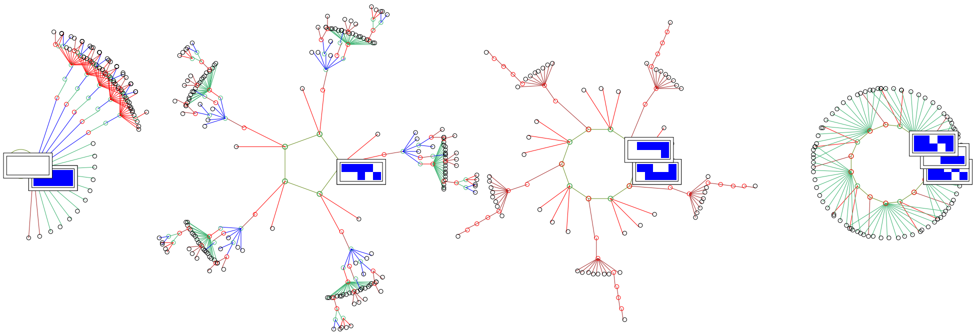


Figure 26.9: Highlighting all non-equivalent attractor states as a 2d bit pattern for the basin of attraction field of a 1d CA, $v2k3$ rule(dec) 111, n=10. Other node are shown as spots.

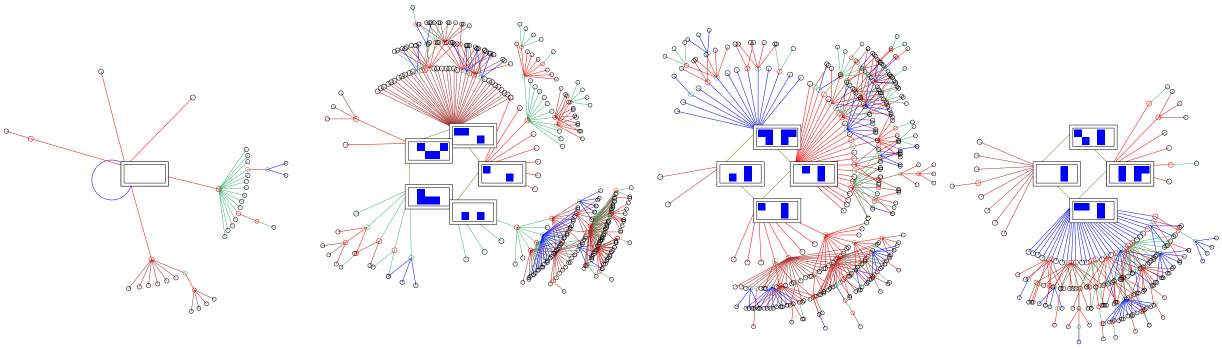


Figure 26.10: Highlighting all attractor states in a RBN basin of attraction field, $v2k4$, n=10. Other node are shown as spots.

Enter **a** to highlight all non-equivalent attractor states for regular 1d or 2d CA (figure 26.9) – if compression is set in section 26.1.1. Otherwise if **a** is entered all attractor states will be highlighted, as in figure 26.10 for a RBN. If either **1** or **a** is entered, and running backwards for a subtree is subsequently selected in section 29.1, the subtree root will be highlighted as a bit pattern as in figure 26.12

26.2.3 Change the 2d node i, j

2d bit/value patterns at all or some nodes are selected by entering,

2d-B . . . for 2d bit/value patterns for 1d (or 3d) networks (section 26.2).
2d-b . . . for 2d bit/value patterns for 2d networks (section 26.2).
1 or **a** . . . to highlight all attractor states, just one, or a subtree root (section 26.2.2).

A 2d network retains its i, j (a 3d network its i) as the default, set in sections 11.6.1 or 12.3. For 1d networks a default i, j is chosen automatically, where j (the number of rows) is the highest whole factor of n, where $j \leq \sqrt{n}$ (see also section 21.4.6). For any dimention, the following top-right prompt allows the default i to be revised,

2d node ij now 7x2: reset i (max 14): *(example for 1d n=14)*

The network is broken up into successive rows starting with the maximum cell index in the top left hand corner. If the i selected is not a whole factor of n, some cells will be missing from the bottom row, which is evident if dots are activated in section 26.2.4.

26.2.4 Alter scale, divisions and dots – node as bits/values

If nodes are set as bits/values either in 1d or 2d (section 26.2) or if the attractor or subtree root nodes are highlighted in 2d (section 26.2.2), divisions between cells, dots at zero values, or both, can be added to the basic presentation, illustrated in table 26.1 below,

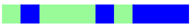
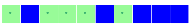
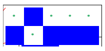
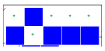
	basic	divs+dots
1d		
2d 5×2		

Table 26.1: Alternative presentations for bit/value nodes, v=2, n=10

For a cell scale in pixels $p \leq 5$, the basic presentation, without divisions or dots, is the default, otherwise dots and divisions are added. This can be changed – the following top-right prompts are presented in sequence, for example,

alter cell scale, now 5 pixels: tog divs (now OFF)-i: tog dots (now OFF)-t:

If required, enter the new cell scale in pixels, then toggle the divisions, then the dots, from their present settings (ON or OFF). These new setting become the new defaults. Note that divisions and dots will not apply if the cell scale in pixels $p \leq 3$.

The cell scale in "backwards" space-time patterns (section 24.14) and the cell scale of basin nodes as bit/vale patterns described in this section, will be the same. Both can be changed on-th-fly as basin are being drawn (section 30.3).

26.2.5 Alter decimal or hex node scale

If nodes were set as decimal or hexadecimal in section 26.2 the default node label size is set automatically depending on the basin scale (see section 25.2.3). This default can be altered by a given factor. The following top-right prompt is presented, for example,,

dec/hex node scale (now 10), enter factor (now 1.0), original-o:

26.3 Change orientation, fan angle, edge color

The next three prompts allow the default orientation of attractor basins (section 23.2), the default pre-image fan angle, and the default edge colors, to the changed. The following prompts top-right are presented in turn,

change basin orientation (now 0), enter angle:
change pre-image fan angle (now 1.00), enter factor:
change start edge color (now 0) —-0 —-1 —-2 —-3:

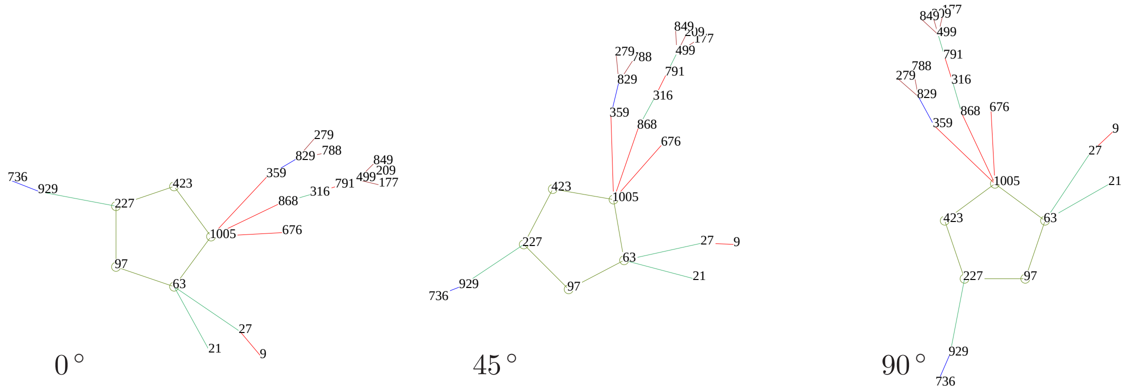


Figure 26.11: Changing the orientation of a single basin of attraction, starting with 0 °, then setting 45 °, and finaly 90 °. Labels are in decimal with a seed of 63, $v2k3$ rule (dec)110, n=10.

26.3.1 Orientation

Changing the orientation allows attractor basins to be rotated anti-clockwise by the angle that is entered at the prompt – **change basin orientation (now 0), enter angle:** – in section 26.3. The dedault is 0 ° due east. Figure 26.11 gives examples. Note that for a single basin of attraction, with the orientation at 0 °, the seed state is positioned one step clockwise from east. Changing the orientation works for all types of attractor basins. The orientation of individual basins in a field can also be changed during a pause as fields are drawn (section 30.2).

26.3.2 Pre-image fan angle

The pre-image fan-angle is the angle containing all the pre-image edges converging on a node – set automatically depending on the number of edges. It is sometimes useful to decrease the angle for denser branching occurring in large networks or ordered rules (figure 26.12), or to increase the angle for chaotic rules with low branching (figure 26.13). This is done by entering a multiplication factor at the prompt – **change pre-image fan angle (now 1.00), enter factor:** – in section 26.3. The fan-angle can also be changed during a pause as fields are drawn (section 30.2).

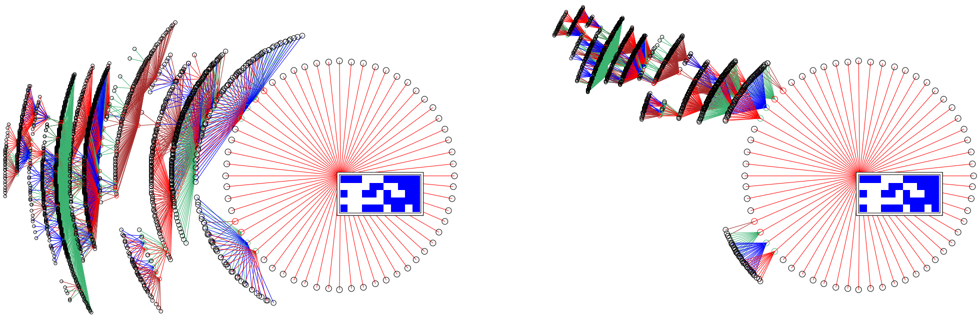


Figure 26.12: Decreasing the pre-image fan-angle of a subtree of a 1d CA $v2k3$ rule (dec)110, n=55. The subtree root, (hex)71f0cf34c0fced, is highlighted as a bit pattern. Other nodes are show as spots. *left*: default fan-angle (factor=1). *right*: reduced fan-angle (factor=0.3). Note that the fan converging on the root of a subtree is always 360 ° 360, and is not affected.

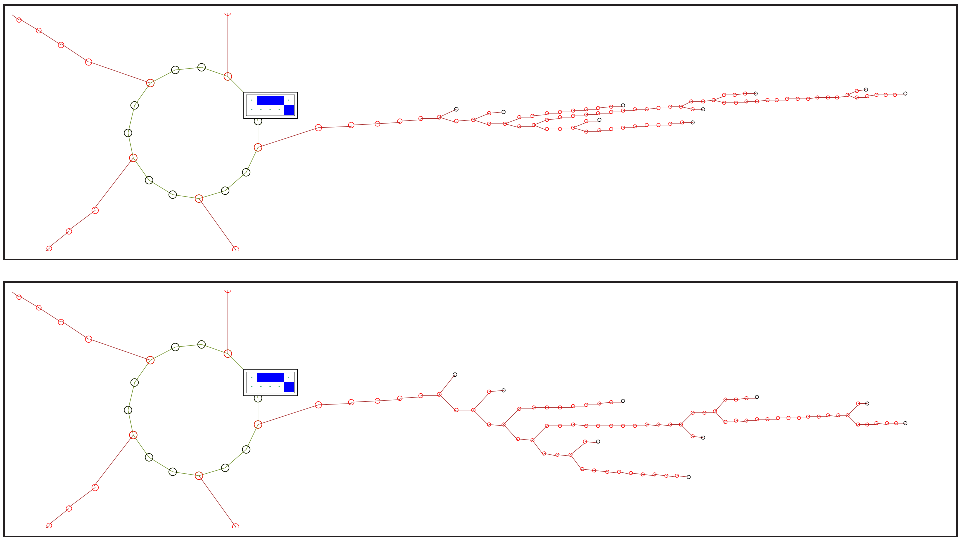


Figure 26.13: Increasing the pre-image fan-angle of a single basin of attraction of a 1d CA, $v2k3$ rule (dec)30 n=10. This is a chain-rule with very low in-degree, characteristic of chaos. In each case, part of the basin including the attractor and one complete tree is shown. One attractor state is highlighted as a bit pattern. Other nodes are show as spots. *Top*: default fan-angle (factor=1). *Bottom*: increased fan-angle (factor=3). The basin orientation was also changed to 34° (section 26.3.1).

26.3.3 Edge color

As described in section 23.3.1, a cycle of four colors is used to draw transition edges, and this also determines node colors (section 26.2.1). The resultant color scheme of the attractor basin depends on the start edge color – red is the default. This can be changed to a different start color resulting in a color scheme for both edges and nodes, by selecting a new start color at the prompt – **change start edge color (now 0) —-0 —-1 —-2 —-3:** – in section 26.3. The prompt colors differ between a DDLab screen with a white or black background (section 6.3.3). The default edge color sequence for a white screen is red-brown-green-blue, for a black screen red-blue-green-yellow.

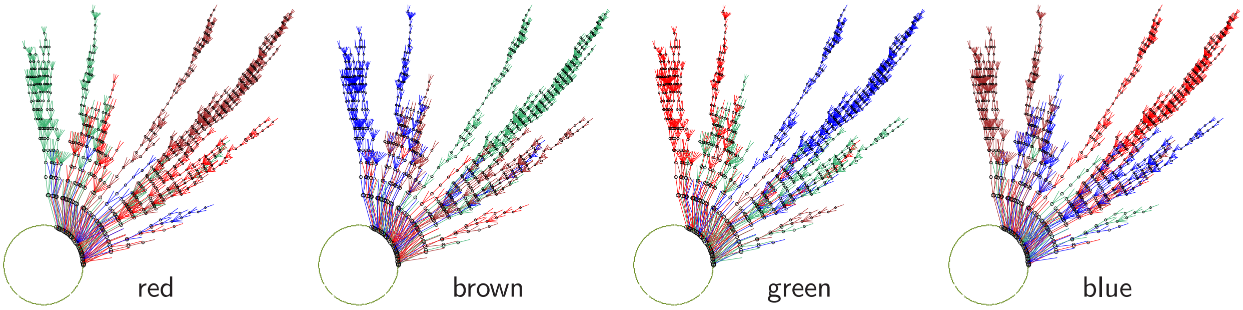


Figure 26.14: Alternative edge start colors (indicated) result in different basin color schemes. This is a single basin with copies of trees suppressed (section 26.1.3), the fan angle reduced by 0.5 (section 26.3.2) and leaf nodes excluded (section 26.2). Transient nodes are shown as spots. $v2k3$ rule (dec)110, n=15, seed(hex) 660d.

Intentionally Blank

Chapter 27

Pausing attractor basins, and data

This chapter describes methods for pausing attractor basins at a hierarchy of stages during construction, to show/print/save data at each stage, and other options. To jump directly to the **pause/data** category of prompts, enter **t** at the first "output parameter" prompt in section 25.2, or arrive there by viewing the output parameters in sequence.

When drawing attractor basins – a basin of attraction field, a single basin or a subtree, or a range n of the above – a data summery is displayed once complete. However, a pause can be set during the drawing itself to show intermediate, more detailed, data. The data can also be saved and/or printed to the terminal at any level of detail, including a sorted list of states.

27.1 Pause stages hierarchy

The top down hierarchy of stages of pause/data, from least to most detail, is as follows,

1. pause after each basin of attraction field in a range of network size n (section 8.1).
2. pause after each basin of attraction in a basin of attraction field, or after each single basin or each single subtree for a range of network size n.
3. pause after each tree (or subtree from the uniform states) in a basin of attraction.
4. pause after each fan – the set of pre-images of each state in a tree.

From the selected pause/data stage, relevant upper stages will also apply. For example, if stage 3 is selected for a basin of attraction field, a data pause will occur at stages 2 and 1. Each stage gives a specific top-centre pause prompt described in section 27.4.

27.2 Pause after each field for a range of fields

If a basin of attraction field for range of n was specified in section 8.1, the following prompt is displayed,

pause/data: field range 5-12, pause after each field-f: *(for example)*

Enter **f** to pause after each successive field. A top-center prompt is also shown, described in section 27.4. If **f** was entered, the prompt in section 27.3 is presented for further stages of detail.

27.3 Pause after each basin, tree, or pre-image fan

A pause can be selected for a basin of attraction field, a single basin, or a subtree (including a range n set in section 8.1) at various stages of detail. The following context dependent top-right prompt is presented,

(for a basin of attraction field, including a range n set in section 8.1)
pause/data: pause for data: fan-3 tree-2, basin-1, none-def:

(for a a single basin, or a subtree)
pause/data: pause for data: fan-3 tree-2, none-def:

Enter **1**, **2** or **3** as required, or **ret** not to pause.

27.4 The pause prompt

At at each pause, as well as basin data in a top-right window, a prompt is presented in a top-centre window depending on the stage reached in the pause hierarchy (section 27.1) as follows,

stage . . . *pause prompt*
1. **next field-ret nopause-q ops-o layout-a:**
2. **next basin-ret nopause-q ops-o layout-a:**
3. **next tree-ret basin-q:**
4. **next fan-ret tree-q:**

option . . . *what it means*
-ret . . . for the next field, basin, tree, or fan.
-q . . . to disable pausing at the current stage – pause at the next higher stage.
-o . . . for a top-right prompt with various options (section 30.4).
-a . . . for the layout options, to amend the angular orientation, subtree fan angle, and the spacing and/or position of each successive basin, described in section 25.3.

27.5 Examples of data

The data are displayed in a top-right window. Examples and decoding are shown below for a CA with compression on (section 26.1). More detailed data, including a sorted list of states, may also also printed or saved to a file (sections 27.6 - 27.8).

27.5.1 Data on basin of attraction fields

By default, when a basin of attraction field is complete, unless a range of fields was specified (see sections 8.1 and 27.2), data on the field is presented in a top-right window, An example of data on a field (for $v2k3$ rule (dec)110, n=12) as in figure 27.1 is shown below,

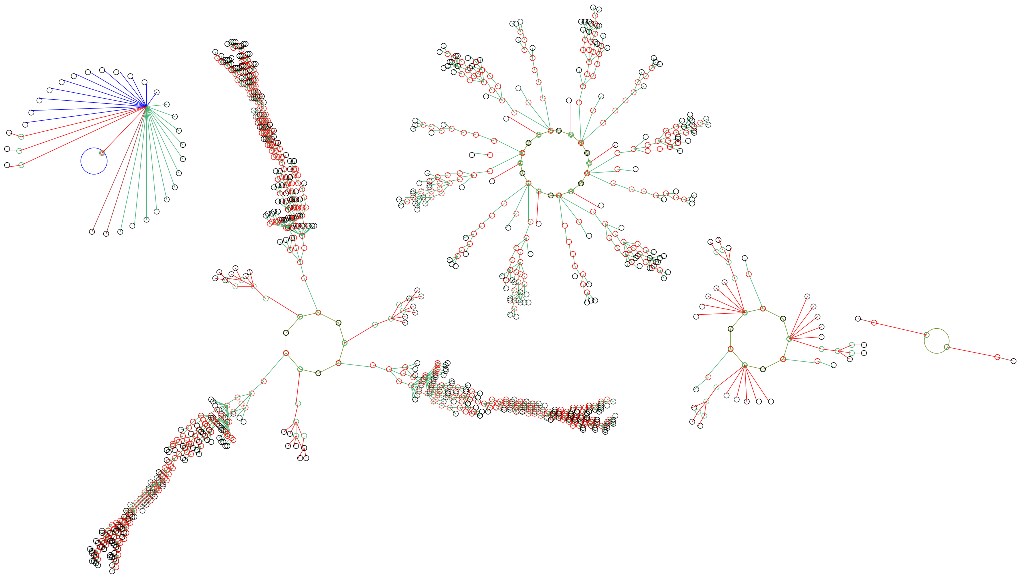


Figure 27.1: A basin of attraction field of a CA, $v2k3$ rule (dec)110, n=12, relating to various data outputs in section 27.5, and saved in sections 27.7.1 - 27.7.4. In this example the default layout was amended in the jump-graph with no edges (section 20.3).

for a single field, or if pausing in a range of fields
basin types=5 total basins=11
n=12 field=4096 g=1971=0.481 0.109sec

If the basin of attraction field is interrupted[1] (by entering **q** twice, see section 30.2), the data on the incomplete field appears as in the example below, depending on point of interuption,

EARLY EXIT basin types=3 total basins=5
n=12 sspace=4096 field=3358 g=1637=0.487 *(computation slowed)*

data type . . . *decode of data*

EARLY EXIT . . . indicates that the field was interrupted.

basin types= . . . the number of basin prototypes displayed.

total basins= . . . the total number of basins in the field.

n= . . . the network size.

field= . . . the total number of states in the field, which should equal the size of state-space, v^n, unless interrupted. If the computed field $\neq v^n$, the size of state-spaces (**sspace=**) will be displayed as well as the incorrect size of the field (**field=**). If the field was not interrupted this indicated an error (section 27.5.2).

g=x**=**y . . . the number and density of garden-of-Eden states in the field.

x**min** y**sec** . . . the time taken to draw the field.

[1] In this and other examples, the computation was slowed (section 24.15) to give enough time to interrupt.

27.5.2 Errors in basin of attraction fields

Inconsistencies and errors in computing attractor basins can be caused by parameter changes while the basin is in the process of being drawn. One such change is abandoning a tree and continuing with the next tree described in section 30.2.1. Another is to change some aspect of the network itself, which is possible while pausing or interrupting (section 30.2).

If such an error occurs in a basin of attraction field, the progress bar (section 30.1) is likely to go off its scale, the size of state-space as well as the field will be indicated in the data window, for example **sspace=1024 field=1486**. A message will also appear below the progress bar, for example, **ERROR excess states=462** indicating that more states were computed than the size of state-space. Section 30.2.2 gives further details.

Note that similar errors can occur in single basins or subtrees, but these will not be indicated.

27.5.3 Data on basins

An example of data on a basin (for $v2k3$ rule (dec)110, n=12) is shown below,

for a single basin

```
equiv basins=4 att(hex)=0c 1c
period=9 size=831=81.2% g=402=0.484 ml=35 mp=13 0.014sec
```

for basin 2 if pausing a field

```
basin 2, equiv basins=4 att(hex)=00 73
period=9 size=831 81.2% g=402=0.484 ml=35 mp=13
```

If the single basin is interrupted (by entering **q** twice, see section 30.2), the data on the incomplete basin appears as in the example below,

```
EARLY EXIT equiv basins=4 att(hex)=0c 1c
period=9 size=42=4.1% g=21=0.5 ml=5 mp=4 4.084sec
```
(computation slowed)

data type ...	*decode of data*
EARLY EXIT ...	indicates that the basin was interrupted.
basin x ...	the basin in question is the prototype of the 2nd type.
equiv basins= ...	number of equivalent basins of this type.
att(hex)= ...	the "first" attractor state, shown in hex.
period= ...	attractor period.
size=x**=**y**%** ...	size of the basin, and the percentage of state-space made up by the basin and its equivalents.
g=x**=**y ...	the number and density of garden-of-Eden states in the basin.
ml= ...	the maximum number of levels in the basin outside the attractor – the length of the longest transient.
mp= ...	the maximum in-degree found in the basin.
x**min** y**sec** ...	the time taken to draw the basin.

27.5.4 Data on trees

An example of data on a tree in basin 2 (as shown in figure 27.1),

```
for tree 3 if pausing basin 2
tree=3, no=3 size=263 (tree types=3 att=9)
root(hex)=01 cc    g=127=0.483 ml=35 mp=13
```

data type	...	*decode of data*
tree=	...	the tree type index.
no=	...	number of trees of this type.
size=	...	size of the tree (including the root).
tree types=	...	total number of non-equivalent tree types.
att=	...	attractor period.
root(hex)=	...	the state at the root of the tree, in hex.
g=x**=**y	...	the number and density of garden-of-Eden states in the tree.
ml=	...	the lenght of the tree, excluding the attractor root.
mp=	...	maximum in-degree found in the tree.

27.5.5 Data on pre-image fans

An example of data on a pre-image fan, in the 2nd tree in basin 2 (as shown in figure 27.1),

```
fansize=2 nextindex=12 level=4 fan-root(hex)=00 07
```

data type	...	*decode of data*
fansize=	...	the number of pre-images in the fan (at least 1).
nextindex=	...	the current accumulated number of states at the next level in the tree – for diagnostic purposes.
level=	...	the current level away from the attractor – the transient length from the fan-root in time-steps.
fan-root(hex)=	...	the state at the root of the fan, in hex.

27.5.6 Data on subtrees

If generating just one sub-tree from an arbitrary state, on completion, data will be displayed in a top-right window.

The first example (figure 27.2) is for a large CA, n=150. A seed state chosen at random was iterated forward by 3 steps and the subtree was generated from the state reached – the subtree root. Running forwards by some steps before running backward (section 29.2) is necessary because a random state is very likely to be a garden-of-Eden state.

For such a large network the root is not shown in the data window – root states are shown if $n \leq 56$. This limit does not apply when data is printed or saved (section 27.7.7).

```
subtree=4335
g=4030=0.93 ml=11 mp=1253 5.292sec
```

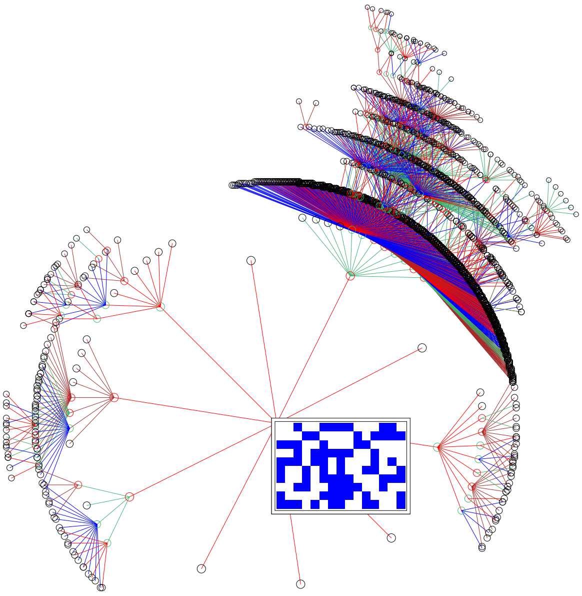


Figure 27.2: A subtree for a CA, n=150, $v2k5$ rule (hex)aa5566a1, from a root state shown as a 2d bit pattern (15 × 10). The root was reached by iterated 3 steps forward from the seed state chosen at random (12b8ffe223639495865e3e60c45f7cce42b251 in hex).

If the subtree is interrupted (by entering **q** twice, see section 30.2), the data on the incomplete subtree appears as in the example below,

EARLY EXIT part subtree=946
g=7=0.0074 ml=2 mp=850 2.500sec

The second example (figure 27.3) is for a small CA n=14 with the same rule, where a random seed was iterated forward by 33 steps before generating the subtree. However, the state reached (the root of the subtree) turned out to be on the attractor, so all the states in the basin were generated. This is indicated by **subtree=basin**. DDLab keeps track of a repeat to prevent the subtree running backwards for ever.

subtree=basin=6895 root(hex)=10 f1
g=2505=0.363 ml=84 mp=6 26.014sec

data type	...	*decode of data*
EARLY EXIT	...	indicates that the subtree was interrupted.
subtree=	...	size of the subtree.
part subtree=	...	size of an interrupted subtree.
subtree=basin=	...	indicates the root state is on the attractor, the whole basin is generated, so this is the size of the basin.
root(hex)=	...	the root of the sub-tree, in hex, if $n \leq 56$.
g=x**=**y	...	the number and density of garden-of-Eden states in the subtree
ml=	...	the maximum number of levels in the subtree, excluding the root.
mp=	...	the maximum in-degree found in the subtree.
x**min** y**sec**	...	the time taken to generate the subtree.

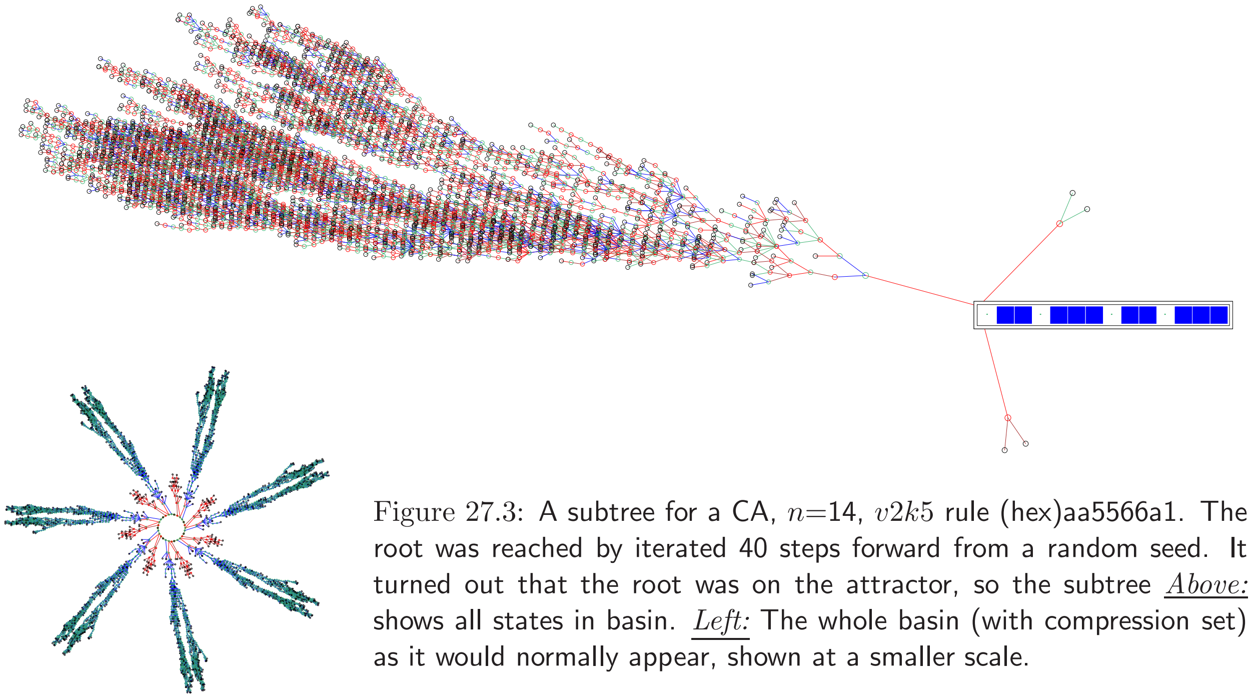


Figure 27.3: A subtree for a CA, n=14, $v2k5$ rule (hex)aa5566a1. The root was reached by iterated 40 steps forward from a random seed. It turned out that the root was on the attractor, so the subtree *Above:* shows all states in basin. *Left:* The whole basin (with compression set) as it would normally appear, shown at a smaller scale.

27.5.7 Data on subtrees from a uniform state

If a subtree pause is selected for a CA in section 27.3 when generating the tree from a uniform state (where all values are the same), with compression set the pause will occur after each equivalent set of subtrees has been generated, and data will be displayed on each subtree prototype.

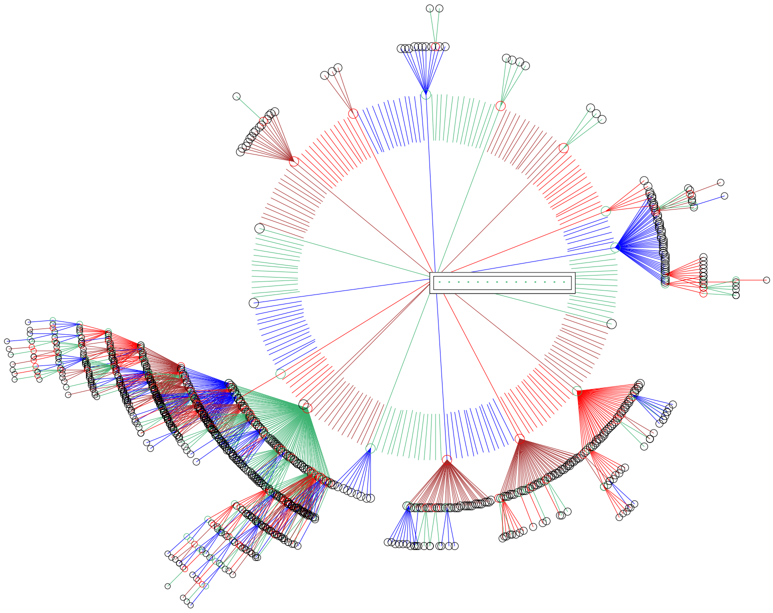


Figure 27.4: Prototype subtrees from a uniform state, all 0s, for a CA, $v2k3$ rcode (dec)96, n=14. Copies of the prototypes have been suppressed in section 26.1.3, and the default fan angle reduced by a factor of 0.3 in section 26.3.2.

An example of the data on the 4th prototype subtree in figure 27.5.7is shown below,

```
seg-fan=211 (17 segs) seg-tree=4 no=14 size=59
seg-root(hex)=00 42 g=51=0.864 ml=8 mp=49
```

data type . . . *decode of data*

seg-fan= . . . the total number of pre-images of the uniform state.

(x seg) . . . made up of x segments or groups of rotation equivalent states.

seg-tree= . . . the subtree prototype number.

no= . . . number of subtrees of this type.

size= . . . size of the subtree including its root.

seg-root(hex)= . . . the state at the root of the sub-tree, in hex.

g=x**=**y . . . the number and density of garden-of-Eden states in the subtree.

ml= . . . maximum number of levels in the subtree including its root.

mp= . . . maximum in-degree found in the subtree.

Once all prototype subtrees have been generated, data on the complete tree is presented as follows (decode as in section 27.5.6),

```
subtree=basin=16333 root(hex)=00 00
g=13363=0.818 ml=9 mp=212 9.266sec
```

27.6 Print or save data

printing data in the terminal is for Linux-like systems only

As attractor basins are drawn, data similar to those described above (sections 27.5.1 – 27.5.7), and also a list of states in different formats (section 27.8) may be printed in the terminal from which DDLab was launched (for Linux-like systems only) and/or output to a `.dat` file.

All the states or just the attractor states can be listed, with or without extra details (section 27.8). States are shown as value strings. The extra details are the state's basin number, level, and the number of its pre-images.

The following prompts, which have the same options, are presented in turn, firstly to print, then to save the data,

for printing data in the terminal – Linux-like systems only

```
print data: basin data only-1, and tree data-2
include states: details-s/+s no-details-S/+S att-only=a/+a:
```

for saving data as an ascii file

```
save data: basin data only-1, and tree data-2
include states: details-s/+s no-details-S/+S att-only-a/+a:
```

Enter a combination of the following,

option ...	*what it means*
basin data only-1 ...	for basin data (section 27.5.3) or subtree data (section 27.7.7).
and tree data-2 ...	for basin and tree data (section 27.5.6).
include states: ...	*show just the states, or add states to selections above (section 27.8.*
details-s/+s ...	list all states including details.
no-details-S/+S ...	list all states without details.
att-only-a/+a ...	list only attractor states states including details.

For example, enter,

1a ...	for basin data, combined with a list of attractor state including details.
2s ...	for basin and tree data, combined with a list of all states including details.
S ...	for a list all states without details.

If the **save data** options were selected, further prompts will appear to set the file name as a `.dat` file (see Filing, chapter 35) – the default filename is `my_data.dat`. The format of the data printed in the terminal, or in the ascii data file, is shown in the examples in section 27.7, together with the decoding of the data.

27.7 Data format

Examples of the data format, printed in the terminal (Linux-like systems only) or saved to a `.dat` ascii file (section 27.6) are described and decoded below.

27.7.1 Network parameters data

If **1** or **2** is selected in section 27.6, the following network parameters data are given first, including the rule, network dimension and size, and various rule parameters.

This example relates to the figure 27.1.

```
v2k3 rcode(dec)110 (hex)6e
1d size=12 ld=0.625 ld-r=0.75 P=0.625
zl=0.75 zr=0.625 Z=0.75 C=0/3
```

data type ...	*what it means*
`v2k3` ...	value-range v, neighborhood size k.
`rcode(dec)110 (hex)6e` ...	the rcode in decimal and hex.
`1d size=12` ...	network dimension and size.
`ld=0.625 ld-r=0.75 P=0.625` ...	λ, λ_{ratio}, P-parameter.
`zl=0.75 zr=0.625 Z=0.75` ...	Z_{left}, Z_{right} and the Z-parameter.
`C=0/3` ...	canalizing inputs.

This initial network parameters data are only given for single rule networks. For mixed rule (and mixed k) networks, the complete network parameters can be printed and saved (also to a `.dat` ascii file) as described in section 19.6.

27.7.2 Basin field data

An example of the complete data for the basin of attraction field in figure 27.1 (enter **1** in section 27.6),

```
data                                  ... decode
v2k3 rcode(dec)110 (hex)6e
1d size=12 ld=0.625 ld-r=0.75 P=0.625
zl=0.75 zr=0.625 Z=0.75 C=0/3    ... network parameters, described in section 27.7.1 above

ty.  at no (p) s % g gd ml mp    ... reminder of data order, key in section 27.7.3 below
1.  0000 1 (1) 34 0.83% 29 0.853 3 29    ... basin 1 data
2.  0073 4 (9) 831 81.2% 402 0.484 35 13    ... basin 2 data
3.  07c7 2 (18) 312 15.2% 138 0.442 10 6    ... basin 3 data
4.  01c7 2 (9) 51 2.49% 27 0.529 4 7    ... basin 4 data
5.  0777 2 (2) 6 0.293% 2 0.333 2 2    ... basin 5 data
basin types=5 total basins=11    ... basin summary
n=12 field=4096    ... network size, field size
g=1971=0.481    ... total garden-of-Eden states, and density
```

27.7.3 Key to basin data order

The key to data on basins is set out below, where the labels are shown as a reminder.

```
ty.  at no (p) s % g gd ml mp
```

reminder labels ... *what they mean*

`ty.` ... basin type numbered in the order drawn.
`at` ... the "last" attractor state in hex.
`no` ... number of equivalent basins of this type – always 1 if no compression.
`(p)` ... attractor period.
`s` ... total states in basin.
`%` ... percentage of state-space made up of this basin type.
`g` ... number of garden-of-Eden states in the basin.
`gd` ... the density of garden-of-Eden states in the basin.
`ml` ... the length of the longest tree in the basin, excluding the attractor root.
`mp` ... the maximum in-degree found in the basin.

If CA compression is suppressed, (see section 26.1), or for a non-CA network, there are no equivalent basins – data on every basin will be shown.

27.7.4 Tree data

An example of the same data as for the basin of attraction field in section 27.7.2 above (see figure 27.1), but also including tree (and subtree) data, which precedes the basin data summary is given below (enter **2** in section 27.6). For CA, subtree data is given for each subtree type rooted on the pre-images of a uniform attractor state (all values equal) (see section 27.5.7).

```
data                              ... decode
v2k3 rcode(dec)110 (hex)6e
1d size=12 ld=0.625 ld-r=0.75 P=0.625
zl=0.75 zr=0.625 Z=0.75 C=0/3     ... network parameters, described in section 27.7.1

ty.  at no (p) s % g gd ml mp     ... reminder of data order, key in section 27.7.3
   tree.  root no s g gd ml mp ... reminder of tree data order, key in section 27.7.5 below
      seg-fan=29 (4 segs)     ... uniform state in-degree, and number of sub-tree types
      1.  0aaa 2 1 1 1 2 0     ... data for each subtree type, in basin 1
      2.  0ab6 12 1 1 1 2 0
      3.  0ad6 12 1 1 1 2 0
      4.  0db6 3 2 1 0.5 3 1
1.  0000 1 (1) 34 0.83% 29 0.853 3 29     ... data, basin type 1
   1.  0fd1 3 13 7 0.538 5 4     ... data for each tree type in basin 2
   2.  0d70 3 1 0 0 0 0
   3.  0730 3 263 127 0.483 35 13
2.  0073 4 (9) 831 81.2% 402 0.484 35 13     ... data, basin type 2
   1.  037d 6 2 1 0.5 1 2     ... data for each tree type in basin 3
   2.  0137 6 1 0 0 0 0
   3.  0f1d 6 49 22 0.449 10 6
3.  07c7 2 (18) 312 15.2% 138 0.442 10 6     ... data, basin type 3
   1.  0f7d 3 13 8 0.615 4 7     ... data for each tree type in basin 4
   2.  0d34 3 1 0 0 0 0
   3.  071c 3 3 1 0.333 2 2
4.  01c7 2 (9) 51 2.49% 27 0.529 4 7     ... data, basin type 4
   1.  0ddd 2 3 1 0.333 2 2     ... data for each tree type in basin 5
5.  0777 2 (2) 6 0.293% 2 0.333 2 2     ... data, basin type 5
basin types=5 total basins=11     ... basin summary
n=12 field=4096     ... network size, field size
g=1971=0.481     ... total garden-of-Eden states, and density
```

27.7.5 Key to tree data order

The key to data on trees is set out below, where the labels are shown as a reminder.

```
tree.  root no s g gd ml mp
```

reminder labels ...	*what they mean*
tree. ...	tree (or subtree) type numbered in the order drawn.
root ...	the state at the root of the tree, in hex.
no ...	number of equivalent trees of this type (always 1 if compression suppressed).
s ...	the size of the tree including the root.
g ...	number of garden-of Eden states in the tree.
gd ...	the density of garden-of-Eden states in the tree.
ml ...	the maximum number of levels in the tree, excluding its root.
mp ...	the maximum in-degree found in the tree.

Note that for CA with compression active (section 26.1), data is also given on subtrees from a uniform state. If CA compression is suppressed, or for a non-CA network, there are no equivalent trees – data on every tree will be shown.

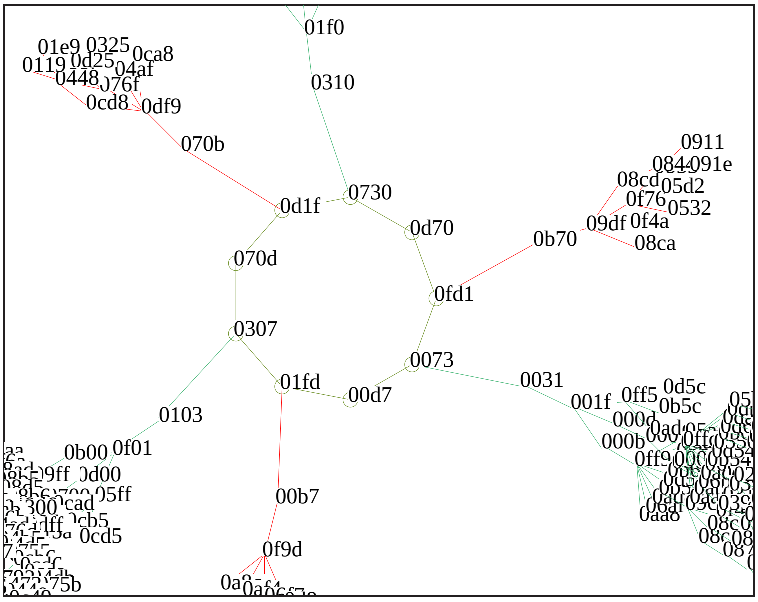


Figure 27.5: A detail of the second basin in figure 27.1 with nodes shown in hex. CA $v2k3$ rcode (dec)110, n=12. To reproduce this basin and the data in section 27.7.6. generate this single basin with the seed (hex)0073.

27.7.6 Single basin data

An example of the data for a single basin, with tree data, is given below (enter **2** in section 27.6). This is the second basin in figure 27.1. A detail with nodes in hex (figure 27.7.5) shows how this data corresponds to the states on the attractor.

```
data                                . . . decode

v2k3 rcode(dec)110 (hex)6e k3
1d size=12 ld=0.625 ld-r=0.75 P=0.625
zl=0.75 zr=0.625 Z=0.75 C=0/3    . . . network parameters, described in section 27.7.1
   1.  0fd1 3 13 7 0.538 5 4    . . . data for each tree type
   2.  0d70 3 1 0 0 0 0
   3.  0730 3 263 127 0.483 35 13
single basin.  0073 4 (9) 831 81.2% 402 0.484 35 13    . . . basin data
```

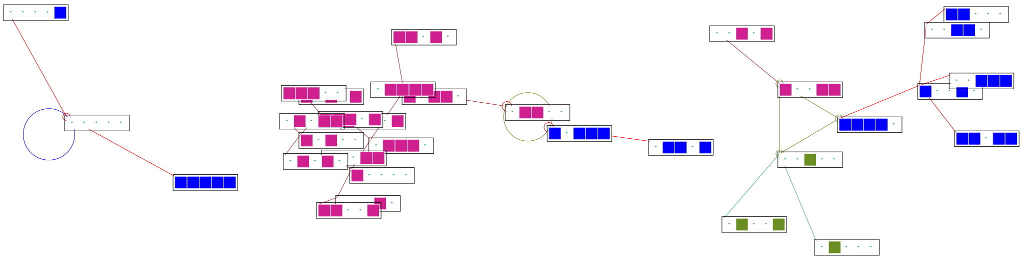
The tree data order is as in section 27.7.5, and the basin data order is in section 27.7.3, but the reminders of the data orders are not shown.

27.7.7 Subtree data

An example of the data for the subtree (n=150) shown in section 27.2 and figure 27.2, is given below (enter **2** in section 27.6).

```
data                                . . . decode

v2k5 rcode(dec)2857723553 (hex)aa 55 66 a1
1d size=150 ld=0.469 ld-r=0.938 P=0.531
zl=0.938 zr=0.672 Z=0.9375 C=0/5    . . . network parameters, described in section 27.7.1
subtree=4335 root(hex)=09e30c2fe4605f23b454a99978e67921e8f599    . . . subtree size, root in hex
g=4030=0.93 ml=11 mp=1253    . . . more data according to the key in section 27.5.4
```



	2.	1.	0.
4:	4	1	2
3:	3	4	4
2:	0	3	4
1:	2	2	4
0:	1	2	2

Figure 27.6: The basin of attraction field of an RBN, with states listed in section 27.8. The RBN has a homogenious rule rule $v2k3$ rcode (dec)110, n=5. Nodes are displayed as a 2d bit pattern (section 26.2). *Right:* The wiring matrix (section 17.2).

27.8 List of states

Examples of the list of states and data (with explanatory notes) are shown in sections 27.8.1 to 27.8.3 below, for a basin of attraction field of a small RBN[2] defined in figure 27.6. The list of states can be very lenghty so a small netork, n=5, is applied in these examples.

As well as showing just the list of states and data, the list can also be broken up into sections giving basin data, or in addition tree data, by entering **1s** or **2s** in section 27.6. Both of these options show network parameters, a reminder of the basin data (and tree data) order at the top, and a summary of data for the basin of attraction field at the bottom, as in section 27.7. Compare these lists with the basin of attraction field and its nodes shown in figure 27.6.

27.8.1 Just the list of states

The example below shows just the list of states and three items of extra data about the states. Enter **s** in section 27.6, or **S** to exclude the extra data. The first column is the state bit-string (or value-string for multi-value). The next three columms give extra information about each string as follows,

the basin number . . . as drawn in sequence. For a CA, if compression has not been suppressed (section 26.1) this is the basin prototype number. For a subtree or single basin the number is always 1.

level . . . the number of steps (levels) of the state away from the attractor for a basin of attraction, or from the subtree root for a subtree. If this is 0 then the state is on the attractor, or is itself the subtree root.

pre-images . . . the number of the state's immediate predecessors (pre-images). If this is 0 then the state is a garden-of-Eden (leaf) state.

[2] Note that in an RBN compression does not apply (see section 26.1).

```
data details        attractors-only          decode

00000 1 0 3          00000 1 0 3               \...
11111 1 1 0          10111 2 0 2
00001 1 1 0          01100 2 0 2
10111 2 0 2          11110 3 0 2
01101 2 1 0          10011 3 0 2
01100 2 0 2          00100 3 0 3
10110 2 1 3                   \
11010 2 2 0                    \...level=0 indicates attractor states
01111 2 2 0
01110 2 2 2
10001 2 3 1
10000 2 3 2
11101 2 4 2
00011 2 4 0
00010 2 4 1
10101 2 5 1
10100 2 5 2
11001 2 5 0
11100 2 6 0
01011 2 6 0
01010 2 6 0
11110 3 0 2
10010 3 1 3
11011 3 2 0
00111 3 2 0
00110 3 2 1
11000 3 3 0
10011 3 0 2
00101 3 1 0
00100 3 0 3
01001 3 1 0
01000 3 1 0
   \   \ \ \...the number of pre-images of the state, garden-of-Eden=0
    \   \ \...level, attractor state or subtree root=0
     \   \...basin number, subtree or single basin=1
      \.....state shown as a bitstring (or value-string)
```

27.8.2 The list of states with basin data

This example shows the same list as in section 27.8.1 together with basin data. Enter **1s** in section 27.6,

```
data                                ... decode
v2k3 rcode(dec)110 (hex)6e
1d size=5 ld=0.625 ld-r=0.75 P=0.625
zl=0.75 zr=0.625 Z=0.75 C=0/3    ... network parameters, described in section 27.7.1 above

ty.  at no (p) s % g gd ml mp    ... reminder of data order, key in section 27.7.3
00000 1 0 3    ... list of states and details for basin 1
11111 1 1 0
00001 1 1 0
1.  00 1 (1) 3 9.38% 2 0.667 1 3    ... basin 1 data
10111 2 0 2    ... list of states and details for basin 2
...            ... just the first and last sates are listed
01010 2 6 0
2.  0c 1 (2) 18 56.2% 8 0.444 6 3    ... basin 2 data
11110 3 0 2    ... list of states and details for basin 3
...            ... just the first and last sates are listed
01000 3 1 0
3.  04 1 (3) 11 34.4% 6 0.545 3 3    ... basin 3 data
basin types=3 total basins=3   ... basin summary
n=5 field=32   ... network size, field size
g=16=0.5   ... total garden-of-Eden states, and density
```

27.8.3 The list of states with basin *and* tree data

This example shows the same (complete) list as in sections 27.8.1 and 27.8.2 together with basin data *and* tree data (decode notes are omitted). Enter **2s** in section 27.6.

```
v2k3 rcode(dec)110 (hex)6e
1d size=5 ld=0.625 ld-r=0.75 P=0.625
zl=0.75 zr=0.625 Z=0.75 C=0/3

ty.  at  no  (p)  s  %  g  gd  ml  mp
   tree. root  no  s  g  gd  ml  mp
00000 1 0 3
11111 1 1 0
00001 1 1 0
   1. 00 1 3  2 0.667 1 3
1. 00 1 (1) 3 9.38% 2 0.667 1 3
10111 2 0 2
01101 2 1 0
   1. 17 1 2  1 0.5 1 2
01100 2 0 2
10110 2 1 3
11010 2 2 0
01111 2 2 0
01110 2 2 2
10001 2 3 1
10000 2 3 2
11101 2 4 2
00011 2 4 0
00010 2 4 1
10101 2 5 1
10100 2 5 2
11001 2 5 0
11100 2 6 0
01011 2 6 0
01010 2 6 0
   2. 0c 1 16  7 0.438 6 3
2. 0c 1 (2) 18 56.2% 8 0.444 6 3
11110 3 0 2
10010 3 1 3
11011 3 2 0
00111 3 2 0
00110 3 2 1
11000 3 3 0
   1. 1e 1 6  3 0.5 3 3
10011 3 0 2
00101 3 1 0
   2. 13 1 2  1 0.5 1 2
00100 3 0 3
01001 3 1 0
01000 3 1 0
   3. 04 1 3  2 0.667 1 3
3. 04 1 (3) 11 34.4% 6 0.545 3 3
basin types=3 total basins=3
n=5 field=32
g=16=0.5
```

The format and keys of network parameters, basin data and tree data are described in sections 27.7.1 - 27.7.5.

Intentionally Blank

Chapter 28

Mutation of attractor basins

When the attractor basin has been drawn for a given network, a new attractor basin may be immediately generated with the same presentation settings, but with a change, or mutation, to some aspect of the network. This process can be repeated indefinitely and automatically. Mutations can be cumulative or relate back to the starting network, and may comprise just one bit or value in a rule table or just one wire move in a RBN. A whole spectrum of mutations can be preset from small to large, from all possible one bit/value mutants, a countdown of the decimal rule number, up to fully random rules and/or wiring. This chapter describes the various mutation options. The methods apply to RBN as well as CA, rule-mix and k-mix, and to all the rule types, rcode, kcode and tcode.

Mutants of single basins can be grouped on one screen (section 25.2.2) – especially useful to show the set of one bit/value mutants of a rule (section 28.3.1).

28.1 The first mutation prompt

If **m** is selected at the first "output parameter" prompt (section 24.1), or if the output parameter prompts are viewed in sequence, the first mutation prompt is presented in a top-right window as follows,

> **mutation: (for next) chain-c Post-P tcode-t kcode-k rcode-r none-n** *(***Post-P** *v=2 only)*
> **wiring-w/+w, rule: special-s, set bits/values-b, single bit/value-1-def:**

28.1.1 The first mutation prompt – options summary

The options are summarised below, some are expanded in greater detail in later sections.

options ...	*what they mean*
chain-c ...	for a random chain-rules (section 16.11). The rules results in attractor basins with very low in-degree, suitable for encryption[35].
Post-P ...	*(value-range v=2 only)* for a random Post-functions (section 14.12). If $k \leq 5$ a Post-function is found at random from rcode-space. For $k \geq 6$ a Post-function if found at random from kcode-space where Post-function are more abundant – otherwise the search would take too long.

tcode-t . . . for a random tcode.

kcode-k . . . for a random kcode.

kcode-r . . . for a random rcode.

none-n . . . for no change – keeping the same rule and wiring – required when trying different layouts, display, and other options in section 24.1).

wiring-w/+w . . . enter **w** to mutate the wiring by moving a selected number of wires randomly within network (section 28.2) without changing the rule. Enter **+w**, for example **cw**, **sw**, **bw** or **1w**, to first set the wiring mutation, then the wiring and rule will mutate simultaneously.

special-s . . . for special rule mutations (section 28.3), including bit/value-flips in sequence (section 28.3.1 – not for k-mix), and single basins grouped on one screen (section 25.2.2).

set bits/values-b . . . to set the number of bits/values to flip in rcode (section 28.4).

single bit/value-1 . . . to flip one bit/value at random, the default, section 28.4 give more information.

For a rule-mix (including a k-mix), these option generally apply at all cell locations simultaniouslly unless noted otherwise.

28.2 Mutate wiring

Enter **w** at the first mutation prompt (section 28.1), or another option followed by **w**, to move one or more wires in the network to random positions for each new attractor basin. A top-right prompt similar to the following is presented,

> **wires to move=1/30=3.33%(def): all-a number-n %-p:** *(values shown are examples)*
> **local-L, from original wiring-w/+w:**

In this example, n=10 and k=3, so the total number of wires is 30. In a mixed k network the total number of wires equals the sum of the n neighborhoods.

The default is to move just one wire. The wiring mutations are either cumulative, or can restart from the original wiring with **wiring-w/+w** below. The wire moves are made by picking a random wire of a random cell for each wire move. Note that a regular CA will be re-assigned as a randomly wired (non-local) network if wiring mutation is chosen, so compression in basins (section 26.1) will be deactivated.

The options below activate the wiring change at each mutation,

options . . . *what they mean*

return . . . to move one random wire in the network, the default.

all-a . . . for new random wiring for the whole network.

number-n . . . to specify the number of wires to move, selected at random, with the following subsequent prompt, . . . **number of wires:**

%-p . . . to specify the percentage of total wires to move, selected at random, with the following subsequent prompt, . . . **% of total wires:**

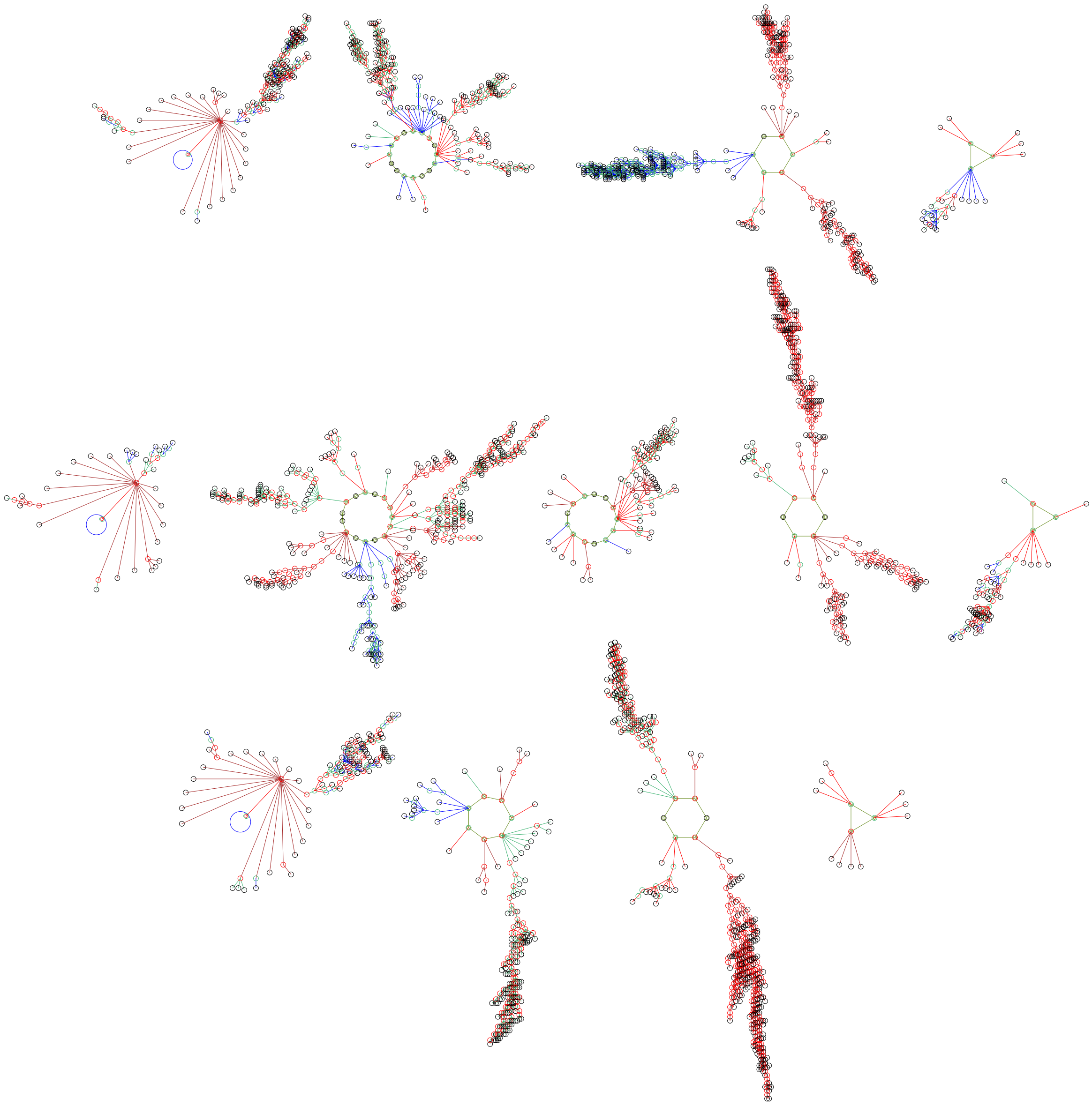


Figure 28.1: Starting with CA $v2k3$ rule (dec)110, n=10, as shown in figure 26.2, three mutant basin fields are shown where one randomly chosen wire was moved at random to a new postion. The actual moves were as follows: *Top:* cell 9, 098 → 398. *Centre:* cell 3, 432 → 452. *Bottom:* cell 5, 654 → 634.

local-L . . . to restore local wiring (for 1d, 2d or 3d). For 1d and a single rule compression in basins (section 26.1) will be restored, and the program will revert to the the first mutation prompt (section 28.1).

wiring-w/+w . . . enter **w** for the default single wire move to start from the original wiring at each mutation, otherwise mutations are cumulative. For non-cumulative mutations of several wires, enter the option followed by **w**, for example **aw** or **pw**.

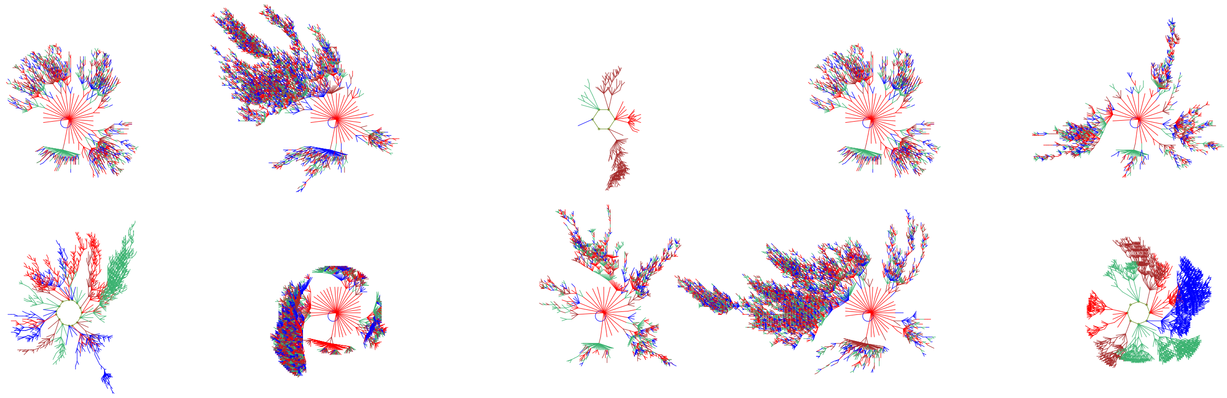


Figure 28.2: Mutant basins of attraction shown on one screen – selected with *mutants*-**M** in section 25.2.2. The original basin *Top left:* is a CA $v4k2$, n=7, kcode (hex)027e60, seed (hex)cdca (as in figure 28.5 but without compression). The other basins are one wire move mutants from the original. As there are many possible single wire moves, this exact set of mutants would be hard to reproduce. In this example, both the nodes (section 26.2, and equivalent trees (and subtrees) (section 26.1.3), are suppressed.

28.3 Special mutation options

Special mutation options take into account the different rule types (rcode, kcode or tcode) selected in section 13.1.1. For single basins or subtrees, sucessive mutants can be generated automatically and grouped on one screen (pausing at each full screen) if *mutants*-**M** is selected in section 25.2.2. otherwise the “attractor basin complete” prompt (section 30.4 occurs after each mutant is complete.

The following kinds of changes/mutations are included in special mutations,

1. A countdown by decimal rule number, provided the rule-table is within the decimal limits (table 16.1). For a rule-mix or k-mix, the rule-number at each cell is decreased by one.
2. A succession of random single bit/value flips. For a rule-mix or k-mix one cell is first selected at random for the flip.
3. Flipping successive bits or values in the rule-table from left to right, either cumulatively or singly. For a rule-mix this applies to all rule-tables simultaneously. This option does not apply for a k-mix.

Enter **s** at the first mutation prompt (section 28.1) for the special rule mutation options, which depend on whether kcode, tcode, or neither, are active, and whether the rule-table is within the decimal limits. The following is an example of a special mutation prompt in a top-right window,

(kcode active, and both kcode and rcode within decimal limits)
specify rcode or kcode mutation
kcodeno-5 kcodeflip-4 rcodeno-3 rcodeflip-2 (def-4)
(Mutant Layout Set) bit/value flip from left: cumulative-1 single-0:

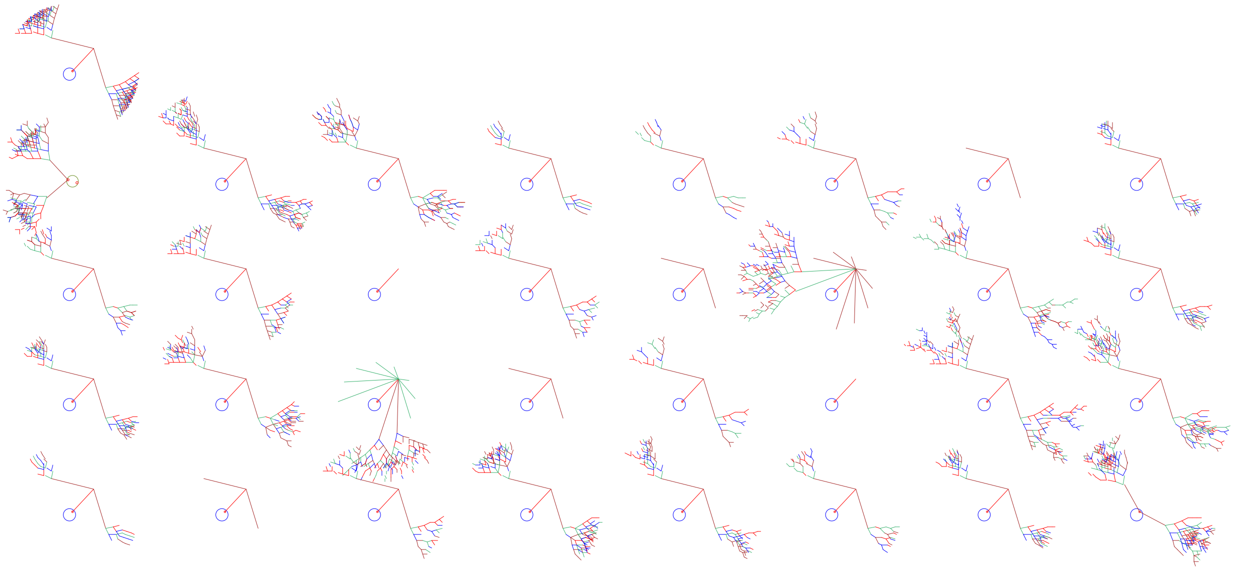


Figure 28.3: 32 mutant one-bit basins of attraction shown on one screen selected with *mutants*-**M** in section 25.2.2.. The original rule *Top left:* is a CA$v3k3$, n=8, rcode 60, seed all 0s. where all states belong to one very regular basin. The rule was first transformed to its equivalent k=5 rule (f00ff00f in hex), with 32 bits in its rule table. *Below:* All 32 one-bit (non-cumulative) mutant basins are shown. If the rule is the genotype, the basin of attraction can be seen as the phenotype.

The options, and the wording subject to the context, are described below,

info and options . . . *what they mean*

specify rcode. . . . . . always appears – the rcode options apply even if kcode or tcode is active.

. . . **or kcode**. . . . . . *(if kcode is active)* otherwise . . . **or tcode**. . . , or neither, appears.

kcodeno-5 . . . *(if kcode is active and within decimal limits)* for a countdown from the initial decimal kcode number.

tcodeno-5 . . . *(if tcode is active and within decimal limits)* for a countdown from the initial decimal tcode number.

kcodeflip-4 . . . *(if kcode is active)* for a random bit/value flip in kcode.

tcodeflip-4 . . . *(if tcode is active)* for a random bit/value flip in tcode.

rcodeno-3 . . . *(if rcode is within decimal limits)* for a countdown from the initial decimal rcode number.

rcodeflip-2 . . . for a random bit/value flip in the rcode.

(def-4) . . . enter **return** for the default – **kcodeflip-4** or **tcodeflip-4** is the default if either is active, otherwise the default is **rcodeflip-2**.

(Mutant Layout Set) . . . *(for single basins, and if mutants-***M***) is selected in section 25.2.2)*.

bit/value flip from left: . . . *(does not apply for a k-mix)* Successive flips from left to right (cumulative or single – below). Section 28.3.1 provides further details.

cumulative-1 . . . keep previous flips.

single-0 . . . restore previous flips – so single bit/value flip to the original rule.

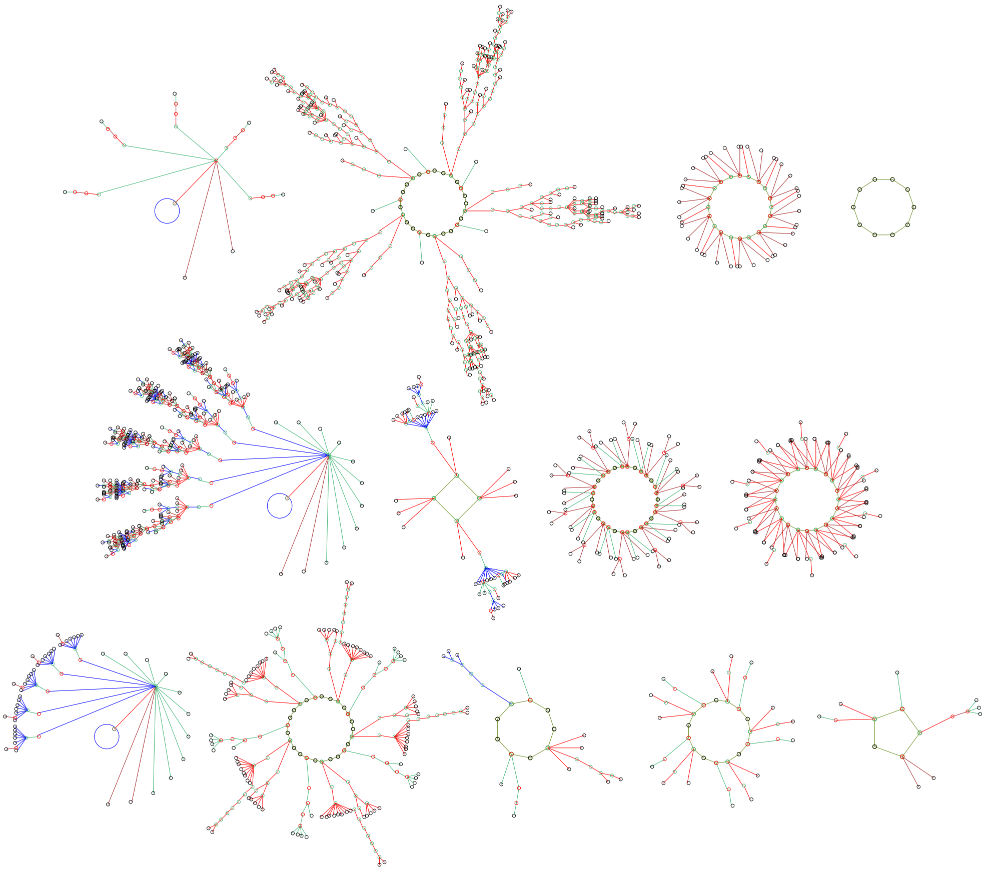


Figure 28.4: Mutation of the basin of attraction field by one bit-flips. The $v2k3$ rcode (dec)110 was first tranfomed to the equivalent $k5$ rcode (hex)3cfc3cfc (section 18.7). The CA, n=10 (basin field shown in figure 26.1), was then mutated in sequence by a single bit from the left (non-cumulatively). Three mutant basin fields are shown (from the top) for the 5th, 8th and 14th bit-flip The tree rules (hex) are 34fc3cfc, 3dfc3cfc, 3cf83cfc.

28.3.1 Bit-flip or value-flip in sequence

not for mixed-k

If **1** or **0** is entered in section 28.3, at each mutation, successive bits in the rule-table (tcode, kcode or rcode) will be flipped in sequence from left to right. The difference is that for **cumulative-1** the previous flip remains unchanged, whereas for **single-0** the previous flip will be restored to its former state. The single-flip results in a set of rules 1 Hamming distance from the start rule – the cumulative-flip results in a steady increase in the Hamming distance.

The start and end position of flips can be altered from the default at the extreme ends of the rule-table. The following top-right prompt is presented, this eample for $v2k3$ rcode,

rcode: specify start - end position of bit/value flip (countdown)
start position (def 31): end position (def 0):

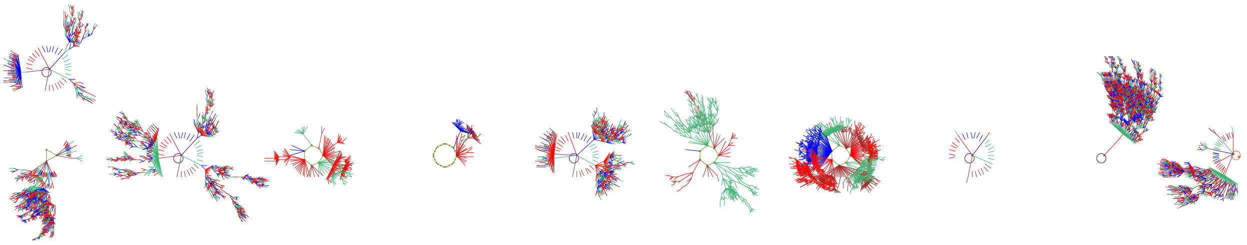


Figure 28.5: Mutant basins of attraction shown on one screen – selected with *mutants*-**M** in section 25.2.2. The original basin *Top left:* is a CA $v4k2$, n=7, kcode (hex)027e60, seed (hex)cdca (as in figure 28.2 but without CA compression active). *Below:* 10 one-value (non-cumulative) mutant basins. As v=4 and the value-flip is random, this exact set of mutants would be hard to reproduce. In this example, both the nodes (section 26.2, and equivalent trees (and subtrees) (section 26.1.3), are suppressed.

Enter the new start position, followed by the new end position. Once the end position is reached, the next mutation will revert to the start.

For a rule-mix the mutation applies to the rule at each cell simultaneously.

Note that finer mutations can be made to rcode if it is first transformed to an equivalent rcode with greater k, as described in section 18.7. For single basins only, the mutant basins can be grouped on one screen with the original basin by itself in the top row – by selecting *mutants*-**M** in the layout (section 25.2.2). For a bit/value-flip in sequence, if the set of mutants fits on one screen the program will pause once the set is complete (or when the screen is full). Figures 28.5 and 28.3 gives examples.

28.4 Flip bits or values

Enter **return** at the first mutation prompt (section 28.1) for the default one bit or value flip, or enter **b** to first alter the number of bits or values to be flipped at random in kcode or tcode if active, or in rcode. The total number of bits/values T, the maximum that may be flipped, depends on the network – if S is the rule-table size (section 13.2.1), n the network size, and i the network index, T is given as follows,

$$\begin{aligned} \text{single rule} \ldots \ & T{=}S \\ \text{rule-mix} \ldots \ & T{=}S \times n \\ k\text{-mix} \ldots \ & T{=}\sum_{i=0}^{n-1} S_i \end{aligned}$$

T appears in the prompt and is the basis for the percentage setting. Examples of the the top-right prompt are as follows,

(single rcode v4k3, T=v^k=64)
rcode: bits/values to flip=1/64=1.56%: all-a number-n %-p

(homogeneous k, mixed rcode v4k3 n=10, T=$n(v^k)$=640)
rcode: bits/values to flip=1/640=0.156%: all-a number-n %-p

Mutations are made by picking a random bit or value of a random cell's rule-table for each bit/value flip. The options are described below,

options . . . *what they mean*

return . . . to flip one bit or value, the default.

all-a . . . for a new random rule or rules.

number-n . . . to specify the number of random flips, with the following subsequent prompt, . . . **number of bits/values:**

%-p . . . to specify the percentage of total bits/values to randomly flip, with the following subsequent prompt, . . . **% of total bits/values:**

28.5 No pause before next mutant

When an attractor basin is complete, the program will pause[1] and wait for **return** to be entered for the next mutant attractor basin. To suppress the pause and produce a continuous display of mutant attractor basins, the following top-right prompt is presented,

dont pause after mutant graphics-y:

Enter **y** to suppress the pause. There will be a short delay to see the complete attractor basin before the next mutant is generated. The default delay can be changed – the following subsequent prompt is presented if **y** was selected above,

change delay, enter factor (now 1.00 max 10):

Enter a factor to change the delay. For example to make it half as long enter .5, twice as long enter 2. To restore the default setting enter 1. This option works for all types of attractor basins, basin fields, single basins, and subtrees, including a range of sizes set in section 8.1. To interrupt the continuous display enter **q**.

[1] The "attractor basin complete" prompt (section 30.4) provides many other options, including methods to examine the current network.

Chapter 29

Final options for attractor basins

This chapter describes a number of final options before attractor basins are drawn, which depend on the parameters set previously in chapters 24 to 28.

If **s** for a "single basin/subtree" was selected at the first DDLab prompt in section 6.2 (and a seed in chapter 21) then a choice will need to be made between a subtree and a single basin of attraction, or to opt for *space-time patterns only* (as in section 24.1).

For a subtree, its possible (and usually advisable) to run forward by a given number of time-steps from the selected seed, and generate the subtree from the state reached. The number of backward steps in the subtree, or in trees in a single basin, can be limited to see just the core behaviour of large networks.

One of two different reverse algorithms that compute pre-images directly are applied automatically by default for either local or non-local wiring. However, the non-local algorithm can be forced on local wiring as a reality check. The inner workings of both algorithms can be observed.

As a further reality check, an exhaustive algorithm can be selected. There are further options for attractor basins of networks with sequential instead of synchronous updating, and for (biased) random maps, which rely on the exhaustive algorithm. Finally, there is an option to generate neutral order components utilising DDLab's subtree methods, showing how the space of sequential orders is partitioned into sets of orders with identical dynamics.

29.1 Subtree or single basin

If a single basin or subtree was selected at the first DDLab prompt in section 6.2 (and a seed in chapter 21) a top-right prompt allows either running backward to generate a subtree, or running forward to determine the attractor cycle and generate a single basin of attraction,

> **backward for subtree-b, single basin of attraction-(def):**
> **space-time patterns only -S, tog backwards stp (now OFF)-s/+s:** *(or* **now ON***)*

Enter **return** or **s** for a single basin of attraction. Enter **b** or **bs** for a subtree[1]. **s/+s** toggles the current setting for showing backward space-time patterns (section 24.14).

[1] Select a subtree if "neutral order components" are to be generated (section 29.10).

Enter **S** to abandon further options in this chapter and proceed with the options for just space-time patterns (chapter 31).

29.2 Subtree: Run forward before running backwards

If **b** is entered for a subtree in section 29.1, a further top-right prompt is presented to first run the network forward from the seed state by a specified number of time-steps before running backwards,

> **forwards before backwards?**
> **how many steps (def 0):**

The state reached on the forward run becomes the root of the subtree. Running forwards before running backwards is usually necessary[2], other than for very chaotic networks such as CA with "chain" rules (section 16.11) – for chain rules large network sizes (n_{Lim}=65025, section 8.3) are permissible. For most other networks, a seed selected at random is very likely to be a garden-of-Eden state which has no pre-images thus no subtree, and a much smaller size is required (section 8.3).

Enter the number of forward steps, or accept the default, 0. A final prompt will be presented (section 29.4) before the subtree is drawn.

29.3 Single basin of attraction

If a single basin of attraction was selected in section 29.1 a final prompt will be presented (section 29.4) before the basin is drawn. For a single basin of attraction (or subtree) although n_{Lim}=65025 (section 8.3) large network sizes should be applied with extreme caution.

29.3.1 Single basin history limit

To generate a single basin of attraction, the network must first be run forward from the initial state (seed) to find the attractor. States at each successive time-step are added to a history array. The state at each new time-step is checked against this history looking for a repeat which would define the attractor cycle. The maximum number of steps in the history array (the history limit) is 65535 (the upper limit of an unsigned short int). If the number of iterations exceeds the history limit, no more time-steps will be added to the history array, but the checks of new time-steps against the history array will continue – for a large state-space and chaotic rules (especially chain-rules section 16.11) the attractor may therefore not be found.

29.3.2 Interrupting while looking for attractor

If the dynamics is interrupted (with **q**) while the network is still iterating forward looking for the attractor cycle, the following top-right message is presented,

> **still looking for attractor cycle, 1247 iterations** *(for example)*
> **exit-q, cont-ret:**

[2] However, enter **return** for zero forward steps if "neutral order components" are to be generated (section 29.10).

Enter **return** to continue looking, or **q** to give up – the top-right attractor basin complete prompt will appear (section 30.4).

29.4 Final attractor basin prompt

Before drawing attractor basins, a final top-right prompt is presented as a reminder of the type of wiring (local or non-local) and the type of attractor basin (subtree, single basin, or basin field), and giving further special options. Enter **return** to skip these options and generate the attractor basin. The wording of the prompt depends on the network parameters, and for example,

for a sub-tree with non-local wiring
sub-tree (non-local) n=8, local-x exh-e rmap-r seq-s nto-o
view ppstack-1 +pause-2, reorder(on) tog-R/+R, limit backsteps-b/+b:

for a single basin with non-local wiring
single basin (local) n=14, non-local-x exh-e rmap-r seq-s
view ppstack-1, limit backsteps+b:

for basin of attraction field with local wiring
basin field (local) n=10, non-local-x exh-e rmap-r seq-s nto-o
view ppstack-1:

29.4.1 Final attractor basin prompt – conditions and reminders

The special options must meet the following conditions to be included in the final attractor basin prompt (section 29.4).

exh-e rmap-r seq-s . . . state-space, v^n, must be within the limits in table 29.1.

nto-o . . . for v=2 and $n \leq 12$ only, and a basin field, or a subtree with zero forward steps (section 29.2).

The first line of the final attractor basin prompt includes the following reminders,

subtree, **single basin**, **basin field** . . . the type of attractor basin.

n=10, **n=5-12** . . . the network size n, or the start and finish for a range of n (section 8.1).

(local), **(non-local)** . . . the wiring; whether local (as in 1d CA) or non-local (as in RBN or DDN) but including regular 2d and 3d. This can be toggled.

29.4.2 Final attractor basin prompt – options summary

Enter **return** to generate the attractor basin with the default reverse algorithm for local or non-local wiring. Other options are summarised below, and described in detail in the rest of this chapter,

options . . . *what they mean*

non-local-x . . . for 1d local wiring, enter **x** to redefine the wiring as non-local and use the non-local algorithn.

local-x . . . for non-local wiring (except mixed-k), enter **x** to restore 1d local wiring (if changed to non-local above), or to rewire any non-locally wired network with 1d local wiring – the non-local wiring will be forgotten, but it could be saved and reloaded (chapter 19).

exh-e . . . *(v^n within the limits in table 29.1 only)* for attractor basins of the current network using the exhaustive algorithm. Optionally save/print the exhaustive pairs. (section 29.7).

rmap-r . . . *(v^n within the limits in table 29.1 only)* for attractor basins of a random map (uses the exhaustive algorithm) consisting of exhaustive pairs (optionally with Hamming bias). Optionally load/save/print the exhaustive pairs. (section 29.8)

seq-s . . . *(v^n within the limits in table 29.1 only)* for attractor basins of the current network, but with sequential updating (uses the exhaustive algorithm). (section 29.9)

nto-o . . . (*subject to the conditions in section 29.4.1*) to find the neutral order components in sequential updating and show the components as if they were subtrees. (section 29.10).

view ppstack-1 . . . to view the changing partial pre-image stack as a histogram, indicating the inner workings of the reverse algorithms (section 29.6) for either local wiring (section 29.6.1), or non-local wiring section 29.6.2.

+pause-2 . . . *(for non-local wiring only)* to view the partial pre-image histogram, as above, and also pause after each set of valid pre-images (section 29.6.2).

reorder(on)-R/+R . . . *(for non-local wiring only)* the reverse algorithm reorders the cells for more efficient computation. The reordering can be toggled off/on by entering **R**, or another entry followed by **R**, for example **1R** (section 29.6.3).

limit backsteps-b/+b . . . *(for a single basin or sub-tree only)* to restrict the number of backward steps, enter **b** or another entry followed by **b** (section 29.5).

29.5 Limit backward steps

Its possible to limit the number of backward steps (or levels) in subtrees (from the root state), or in the trees of single basins (from the attractor), to see just the core behavior. If **b**, or another entry followed by **b**, is entered in section 29.4, the following top-right prompt is presented,

enter max backward steps (def no limit):

Enter the number of backward steps. Trees and sub-tree will stop generating further pre-images when the number is reached. Enter **return** not to impose a limit.

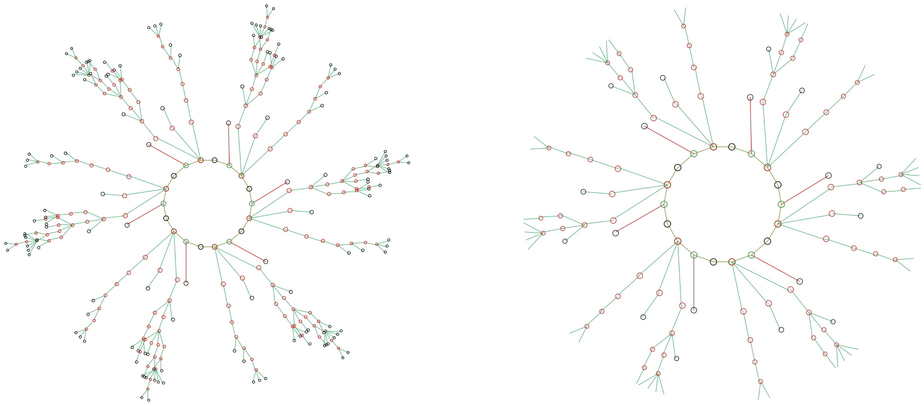
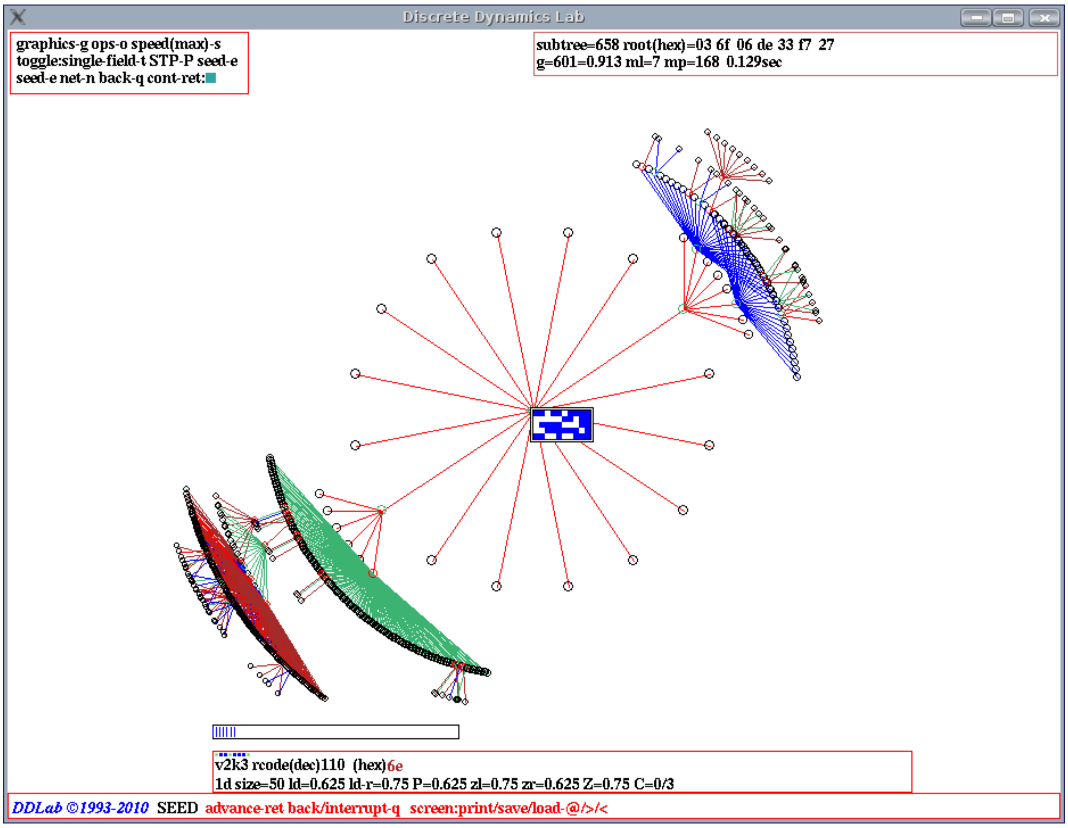


Figure 29.1: Limiting the number of backward steps from the attractor in the trees of a single basins. *Left:* the complete basin. *Right:* trees limited to 5 backward steps. $v2k3$rcode 110, n=12, seed (hex)08e0.



Discrete Dynamics Lab

graphics-g ops-o speed(max)-s
toggle:single-field-t STP-P seed-e
seed-e net-n back-q cont-ret:■

subtree=658 root(hex)=03 6f 06 de 33 f7 27
g=601=0.913 ml=7 mp=168 0.129sec

v2k3 rcode(dec)110 (hex)6e
1d size=50 ld=0.625 ld-r=0.75 P=0.625 zl=0.75 zr=0.625 Z=0.75 C=0/3

DDLab ©1993-2010 SEED advance-ret back/interrupt-q screen:print/save/load-@/>/<

Figure 29.2: A 1d local CA subtree, showing the partial pre-image stack, $v2k6$ rule (dec)110, $n = 50$. The partial pre-image stack is represented by a horizontal bar extending from left to right within a rectangle near the bottom of the screen – its length (in pixels) shows the varying size as the partial pre-image stack grows and shrinks, leaving a trace of its maximum extent (shown in this ecample). It may be necessary to slow computation (sections 24.15 and 30.4) to see the bar.

29.6 Viewing the partial pre-image stack

For networks with synchronous updating (the default in DDLab) two distinct reverse algorithms apply to generate pre-images[19, 20, 26] – for either local or non-local wiring. The local algorithm is for homogeneous-k networks with 1d CA (nearest neighbor) wiring, but with possibly mixed rules. The non-local algorithm is for any other type of wiring (RBN, DDN, mixed-k, 2d or 3d networks). The non-local algorithm in general is considerably slower. Both algorithms work by generating a set of partly computed pre-images, "partial pre-images", held in a stack until either completed or rejected as invalid. Each partial pre-image can give rise to more partial pre-images.

Some inner workings of these algorithm may be observed by viewing the partial pre-image stack as it initially grows, then shrinks, both for the local wiring algorithm (section 29.6.1), and for non-local wiring algorithm (section 29.6.2).

29.6.1 Partial pre-image stack, local wiring

For local 1d wiring, enter **1** in section 29.4, to see how the partial pre-image stack varies in size for the local 1d reverse algorithm. This is shown as a horizontal bar near the bottom of the screen, whose length (in pixels) corresponds to the varying size of the partial pre-image stack (figure 29.2).

29.6.2 Partial pre-image stack, non-local wiring

For non-local wiring, enter **1** or **2** in section 29.4 for a view of the innner workings of the non-local reverse algorithm. A histogram will appear in the lower half of the screen with n columns (figure 29.2) showing the changing partial pre-image stack for each successive cell in the network starting from the left. When the attractor basin is complete (or abandoned), a final histogram gives the average over all histograms to date.

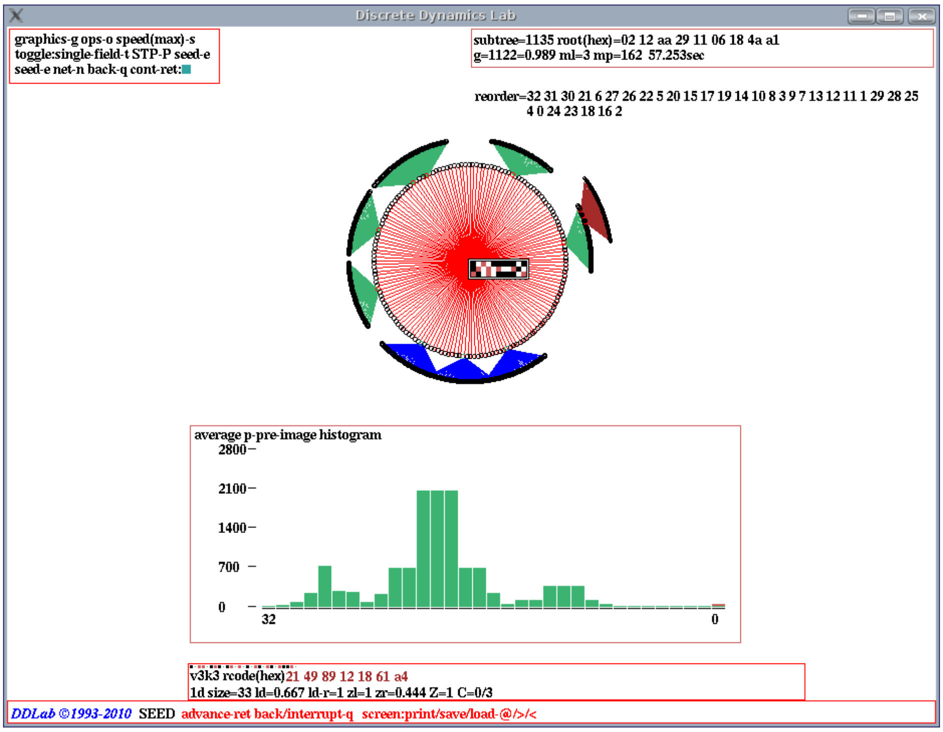
If **1** is entered in section 29.4, the histogram changes continually, and the following top-right prompt is predented,

partial pre-image histogram: start pause-p:

If **2** is entered in section 29.4, or if **p** is entered above, there will be a pause just before each pre-image fan is drawn – but no pause for garden-of Eden states. The following top-right prompt is presented, explained below,

pre=81 (wild=9) abandon pause-p, early exit-q, see next-def:

info or option . . .	*what it means*
pre=81 . . .	the size of the pre-image fan.
(wild=9) . . .	*(if wildcards exist)* the number of pre-images resulting from wildcards.
abandon pause-p . . .	to abondon pausing.
early exit-q . . .	to abandon the single basin or subtree (section 30.2.3), giving the "Attractor basin complete" prompt (section 30.4).
see next-def . . .	for the next pre-image fan and pause.



graphics-g ops-o speed(max)-s
toggle:single-field-t STP-P seed-e
seed-e net-n back-q cont-ret:

subtree=1135 root(hex)=02 12 aa 29 11 06 18 4a a1
g=1122=0.989 ml=3 mp=162 57.253sec

reorder=32 31 30 21 6 27 26 22 5 20 15 17 19 14 10 8 3 9 7 13 12 11 1 29 28 25
4 0 24 23 18 16 2

v3k3 rcode(hex)21 49 89 12 18 61 a4
1d size=33 ld=0.667 ld-r=1 zl=1 zr=0.444 Z=1 C=0/3

DDLab ©1993-2010 SEED advance-ret back/interrupt-q screen:print/save/load-@/>/<

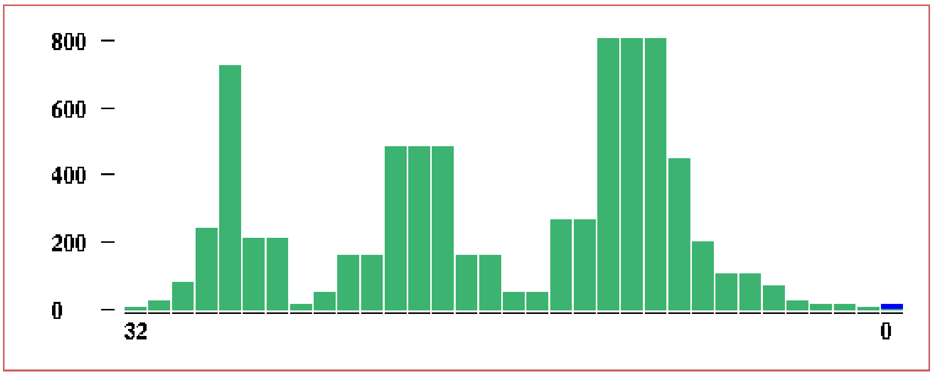


Figure 29.3: A 1d non-local CA subtree ($v3k3$, n=33 – defined below). The changing partial histogram shows the varying size of the partial pre-image stack for each *next* cell. The final bar height represents the number of pre-images of a particular state. The numbers below the top-right prompt window show how cells were reordered to optimize the non-local 1d reverse algorithm (section 29.6.3).
Top: The DDLab screen showing the complete subtree and the final histogram giving the average over all histograms to date. *Above:* The histogram of the first subtree at level 1. The network: n=33, $v3k3$ rcode (hex)214989121861a4 (chain rule) and random wiring defined in file `v3n33alg.wso`, 2 steps forward from the initial state (hex)010a154814556a4201 before running backwards.

Successive values in the histogram tend to increase then decrease – if the value reduces to zero at any time the state is a garden-of-Eden state. The value of the last green column represents the final set of pre-images. However these pre-images may contain "wild-cards" cells, which have no outwires. An additional blue column (red for the final histogram) on top of the last green column shows additional pre-images that result from filling in these wild-cards.

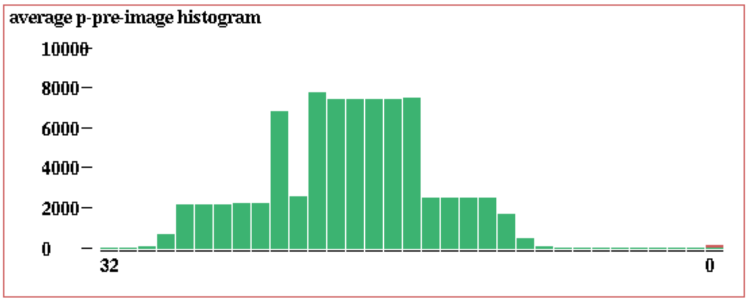


Figure 29.4: The final partial pre-image histogram, with the reordering algorithm suppressed, for the same network as in figure 29.3 *Top*. An approximate comparison of the time t, and the maximum average stack height h, with the reordering algorithm active is as follows, active: t=57sec, h=2000, inactive: t=5min 51sec, h=8000.

29.6.3 Reorder to optimize the non-local reverse algorithm

The algorithm to reorder cells for more efficient computation is implimented by default, however it can be toggled off/on by entering **R**, or another entry followed by **R**, for example **1R** in section 29.4. Toggling reordering *off* impliments the reverse algorithm from left to right.

The no-local reverse algorithm[20, 24] builds a stack of partial (i.e. incomplete) pre-images by taking each cell in turn and filling in valid entries taken from the rule-table of that cell according to the cell's wiring. Any conflict between successive entries disqualifies the partial pre-image. The partial pre-image stack will grow, then shrink as conflicts mount (figure 29.3).

The algorithm works for cells taken in any order. However, to reduce the growth of the partial pre-image stack, the cells are reordered to ensure there is wiring overlap of succesive cells (if possible), to allow conflics to happen early. For fully random wiring, this reduces the growth of the partial pre-image stack considerably compared to a left to right (one arbitrary) order, which is of course very efficient for 1d CA wiring. Memory load can be reduced and computation speed increased. The new order (if applied) is shown below the top-right prompt window (figure 29.3). The reordering reduced the time (and memory) to complete this subtree by a factor of about 6 (figure 29.4.

The new order is local within the reverse algorithm and makes a small difference to the presentation of attractor basins. The partial pre-image histogram in section 29.6.2 (if active) will appear according to the new order.

29.7 Exhaustive algorithm

v^n within the limits for the exhaustive algorithm in table 29.1

Enter **e** in section 29.4 to use the exhaustive algorithm, instead of either of the default reverse algorithms (sections 29.6 and 2.18). The following exhaustive pairs prompt (below the top-right) is presented, and options summarised below.

exhaustive pairs: compute-ret, and save-s, and prt-p/+p: (**prt-p/+p** *not for DOS)*

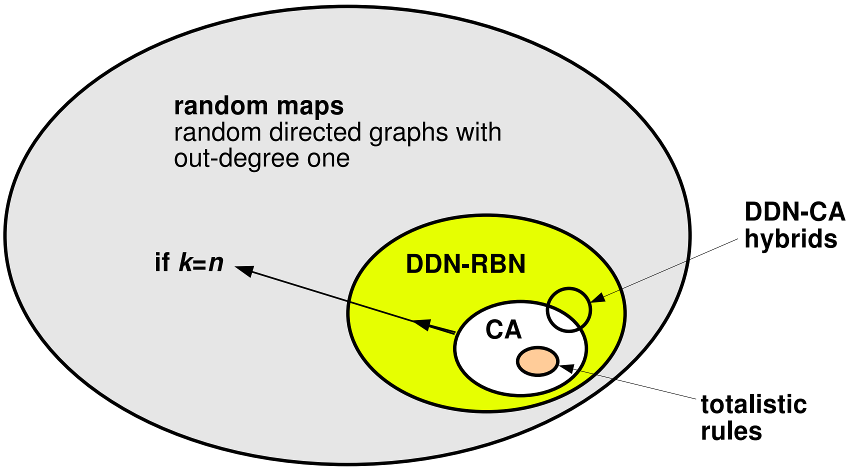


Figure 29.5: CA, DDN/RBN, and random maps, are systems that form a hierarchy of subsets , with random maps the most general. This is reflected by the reverse algorithms, for (a) 1d local wiring, (b) non-local wiring but including (a), and (c) exhaustive testing which includes (a) and (b) as well as random maps and asynchronous updating.

After the attractor basin is drawn, the exhaustive algorithms remains active and the prompt reappears before the next mutant (chapter 28) – enter **q** to exit.

options . . . *what they mean*

return . . . to compute the list according to the current network architecture.

and save-s . . . to compute the list as above, and save it to a `.exh` file, which can be loaded as a random map (sections 29.8 and 29.8.1).

and prt-p/+p . . . *(not for DOS)* enter **p** to compute the list as above, and print to the terminal (section 29.7.3), or or **lp** or **sp** to load or save the list and print it at the same time.

Although the exhaustive algorithm is limited to small networks the method applies to CA, DDN/RBN, and random maps, systems that form a hierarchy of subsets with random maps as the most general (figure 29.5). The exhaustive algorithm also applies to asynchronous updating (section 29.9).

The exhaustive algorithm is efficient for basin of attraction fields, or large single basins/subtrees that use up a good proportion of state-space, and additionally is not sensitive to neighborhood size, k. However the method is highly sensitive to increasing network size n. Every state in state-space must be iterated forward by one step according to the network architecture, and the list of v^n "exhaustive pairs", each state and its successor (a non-random map) held in an array. The maximum network size n_{exhL} is listed in table 29.1 for each value-range v. In practice n is limited to much smaller networks to compute the complete list in a reasonable time. However, even with the maximum size it should be possible to see the start of the list being printed in the terminal (with **prt-p/+p**).

Once complete, attractor basins are generated by scanning the list of successors for pre-images of a given state – if the given state is not found it is a garden-of-Eden (leaf) state.

v	n_{exhL}	state-space v^n ≈ millions
2	28	268435455 ≈ 268.4
3	18	387420488 ≈ 387.4
4	14	268435455 ≈ 268.4
5	12	244140624 ≈ 244.1
6	11	362797055 ≈ 262.8
7	10	282475248 ≈ 282.5
8	9	134217727 ≈ 134.2

Table 29.1: Maximum network size for the exhaustive testing algorithm for different value-ranges v, and the corresponding maximum state-space v^n.

29.7.1 Generating the exhaustive list

Enter **return** or **p** in section 29.7 to compute the exhaustive list. A horizontal red progress bar (shown) near the top-right of the screen will monitor the proportion of state-space completed so far, together with the following message,

setting up exhaustive pairs, interrupt-q:

If **q** is entered to interrupt, which may be necessary if you lose patience with a large network, the following prompt is presented,

exhaustive pairs 25% complete; stop-q options-o cont-ret:

Enter **q** to abandon the exhaustive pairs computation, enter **o** for the "options" prompt in section 30.4, or **return** to continue.

29.7.2 Saving the exhaustive pairs

Enter **s** in section 29.7 to compute and then save the exhaustive pairs. Once the list is generated the following prompt is presented below the red progress bar,

SAVE exhaustive pairs-s:

Enter **s** to save the list as an `.exh` file (chapter 35). The list is essentially a random map (section 29.8) but is not random, and the file can only be loaded from the random map prompt (section 29.8 and 29.8.1).

29.7.3 Printing the exhaustive pairs in the terminal

Enter **p** to compute and print the list of exhaustive pairs to the terminal, or **sp** to save the list and print it at the same time. The list is presented in four columns. The left two are the consecutive states in state-space ordered by the decimal equivalents of the rcode, first the state in decimal, then its bit/value string. The right two columns show the corresponding successor states, first state bit/value string, then in decimal – tables 29.2 and 29.3 give examples.

```
consecutive states       successor states
     dec       bits          bits       dec
     .... ............ ............ ....
     4090 111111111010 100000001111 2063
     4091 111111111011 000000001110 14
     4092 111111111100 100000000101 2053
     4093 111111111101 000000000111 7
     4094 111111111110 100000000011 2051
     4095 111111111111 000000000000 0
```

Table 29.2: Printing exhaustive pairs, the final six items of the list, for the 1d CA $v2k3$ rule (dec)110, n=12, as in figure 27.1.

```
consecutive states            successor states
     dec       bits              bits       dec
   ...... ............ ............ .......
   728060 002301233330 101210302211 4607141
   728061 002301233331 001210302220 412840
   728062 002301233332 301210302233 12995759
   728063 002301233333 101210302221 4607145
   728064 002301300000 101211311111 4611413
   728065 002301300001 001211311101 417105
   ...... ............ ............ .......
```

Table 29.3: Printing exhaustive pairs, for a $v4k3$ rule selected at random, n=12. A six item sample from the list after 60 seconds, The complete list of 16777215 (16.7 million) items took about 3 minutes (without printing).

29.8 Random map

v^n within the limits for the exhaustive algorithm in table 29.1

Enter **r** in section 29.4 to generate a random map, which also consists of exhaustive pairs but created at random (but possibly with a Hamming distance bias), instead of by some dynamical rule. The following prompt below the top-right is presented, with some options similar to "exhaustive pairs" (section 29.7) – the same size constraints apply (table 29.1),

random map: load-l, rnd-(def), set Ham-h, and save-s/+s, and prt-p/+p:
(**prt-p/+p** *not for DOS)*

After the attractor basin is drawn, the random map function remains active and the prompt reappears before the next (new) random map – enter **q** to exit. However, if Hamming bias was set, the prompt changes as follows,

random map: load-l, Ham1-(def), reset Ham-h, and save-s/+s, and prt-p/+p:
(**Ham**x *shows the current Hamming bias)*

options ... *what they mean*

load-l ... to load an `.exh` file of a random map (section 29.8.1).

rnd-(def) ... enter **return** to create a random map, which is a list of random states just like exhaustive pairs. While the (random) exhaustive pairs are being assigned, a horizontal red progress bar near the top-right of the screen will monitor the proportion of state-space completed so far, together with prompts to interrupt etc. as described in section 29.7.1.

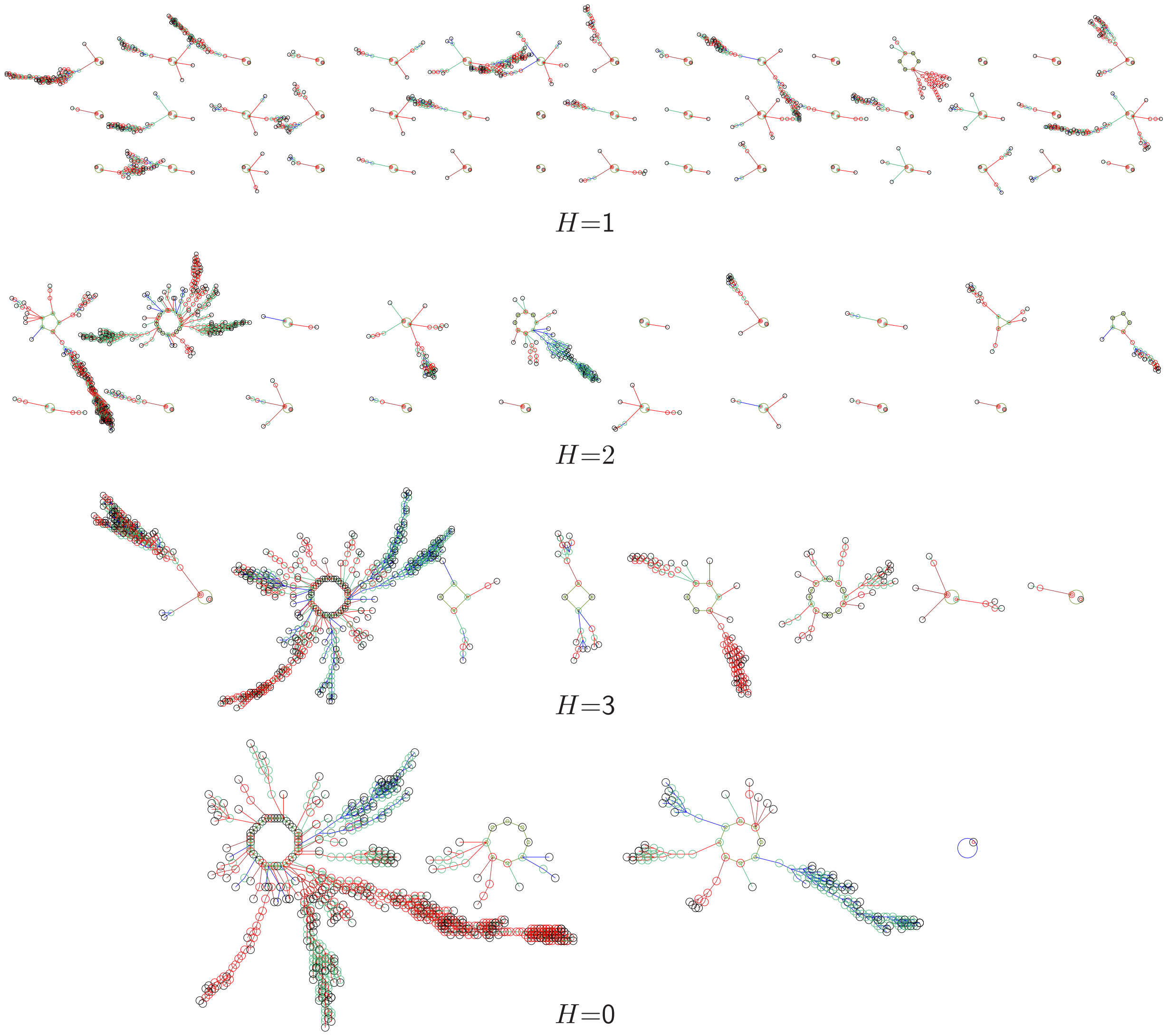


Figure 29.6: Typical random map graphs (basin of attraction fields) with different Hamming bias H. From the top, H=1, H=2 ,H=3, and without bias H=0, so fully random. H=1 is the most fragnented, H=0 tends to have a characteristic giant component[24, ch.3].

set Ham1-(def) ... enter **return** to create a random map with the current Hamming distance bias, shown as **Ham**x, i.e. **Ham2**, **Ham3** etc.

set Ham-h ... (or **reset Ham-h**) to create a random map, but biased according to Hamming distance (section 29.8.2).

and save-s/+s ... enter **s**, to create a random map as above, and save it to a `.exh` file, or **hs** to reset the Hamming bias and save. A prompt will appear to **SAVE exhaustive pairs-s:**, the same as in section 29.7.2.

and prt-p/+p ... *(not for DOS)* enter **p** to create a random map as above,and print to the terminal (section 29.7.3), or **lp** or **sp**, to load or save the list and print it at the same time, or **hsp** to reset the Hamming bias, save, and also print.

In DDLab, a random map is a directed graph with out-degree one, representing an exhaustive list of transitions, and is arguably the most general discrete dynamical system. The set of all random maps contains the subset of all DDN/RBN, which in turn contains the subset of all CA.

Provided that the size of the map is a power of the value range v, the function to set up a random map is implicit in the exhaustive testing algorithm (section 29.7). This algorithm creates a list of v^n pairs of states, each state and its successor, by iterating each state in state-space forward by one step according to the network dynamics. A random map, on the other hand, assigns the successors at random (or with some bias). Assigning successors according to the dynamics of a particular class of network is just a special case of such a random assignment.

Some randomly assigned successors states are likely to occur more than once, so other states will not occur at all (there will be no room left in the list). These are the states without pre-images, "garden-of-Eden" states. A random map has all the properties of a dynamical system, trajectories, attractors, etc. though this may be irreducible to a sparsely connected network. A fully connected network, however, where $k = n$, where each cell receives inputs from all cells (including from itself), can implement any random map. The space of all such fully connected networks, and all random maps (of the Boolean hypercube of length n) is $(2^n)^{2^n}$, which is also the upper limit for the space of all possible binary basin of attraction fields.

29.8.1 Loading the random map or exhaustive pairs

Enter **l** in section 29.8 to load a previously saved list of exhaustive pairs from an `.exh` file (chapter 35), and draw the attractor basins. Note that the current values of v, n must match the file value of v, n. If not, a top-right warning[3] such as the following will be displayed,

file n=7 != network n=8 (wrong file) cont-ret: *(values shown are examples)*

The default file name for a random map (or exhaustive pairs) file is `myexh_v`x`.exh` where x is the current value range v. The (binary) encoding is described in section 35.1 (EXHAUSTIVE).

29.8.2 Biasing the random map by Hamming distance

If **h** was entered in section 29.8 the random map can be biased (or the current bias reset) according to Hamming distance. The following prompt near the top-right is presented,

set Hamming distance (0-10): *(for n=10)*

Enter the Hamming distance H (0-n). 0 signifies no bias, a fully random map. If biased, the exhaustive pairs will differ by H bits or values, i.e. between each bit/value string and its assigned successor. Figure 29.6 shows random map graphs (basin of attraction fields) with a range of Hamming bias H from H=1 to no bias, a fully random map. H=1 is the most fragnented, H=0 tends to have a characteristic giant component[24, ch.3].

While the exhaustive pairs are being assigned, a horizontal red progress bar near the top-right of the screen will monitor the proportion of state-space completed so far, together with prompts to interrupt etc. as described in section 29.7.1.

[3] A mismatch in v will usually be detected because the file size will be incorrect, but this will be reported as a mismatch in n.

29.8.3 Random map data

In addition to the normal data shown in a top-right window when a subtree, basin, or the basin of attraction field is complete (chapter 27), for a complete random map (basin of attraction field), additional data shows the frequency of different cycle periods in a bottom center window, which is of some interest in graph theory[24, ch.3]. This example (from figure 29.6 H=2) shows that there are 14 cycles with period 2, 1 with period 3, etc. The last column (10+) gives the frequency of period 10 or more.

```
Random Map, v2 n=10  cycle period:    1   2   3   4   5   6   7   8   9   10+
Ham bias=2           cycle frequency: 0   14  1   0   2   0   0   1   0   1
```

For a single basin or subtree, the window just gives the message,

Random Map, v2 n=10 *(for example)*

29.9 Sequential updating

v^n within the limits for the exhaustive algorithm in table 29.1

By default, network updating is synchronous, in parallel. Alternatively, the updating can be sequential. The way that cell indexes are listed from left to right is the sequential order, and corresponds to an update-tag (from 0 to n-1), for example,

0	1	2	3	4	5	6	7	8	9	10	11	-	the update-tag for n=12
2	8	10	5	6	9	7	4	11	0	3	1	-	a possible sequential order

There are n! (factorial n) possible sequential orders in a network of size n. DDLab provides simple default orders, left to right (e.g. 9876543210 for n=10), right to left (e.g. 0123456789 for n=10), and a random order. A specific order can also be set by hand. The order can be saved and loaded from a `.ord` file. For $n \leq 12$, all possible orders can be listed – identified by an "order-index" from 0 (left to right) to n!-1 (right to left), and an order can be chosen from this list.

Enter **s** in section 29.4 to select sequential updating. The following top-right prompt is presented,

sequential update: >-(def) <-b rnd-r set-s all-S: *(***all-S** *if $n \leq 12$)*

At the same time a top center reminder **backtrack sequential-q** remains in place while sequential selection is in progress. The options are summarised below.

options . . . *what they mean*

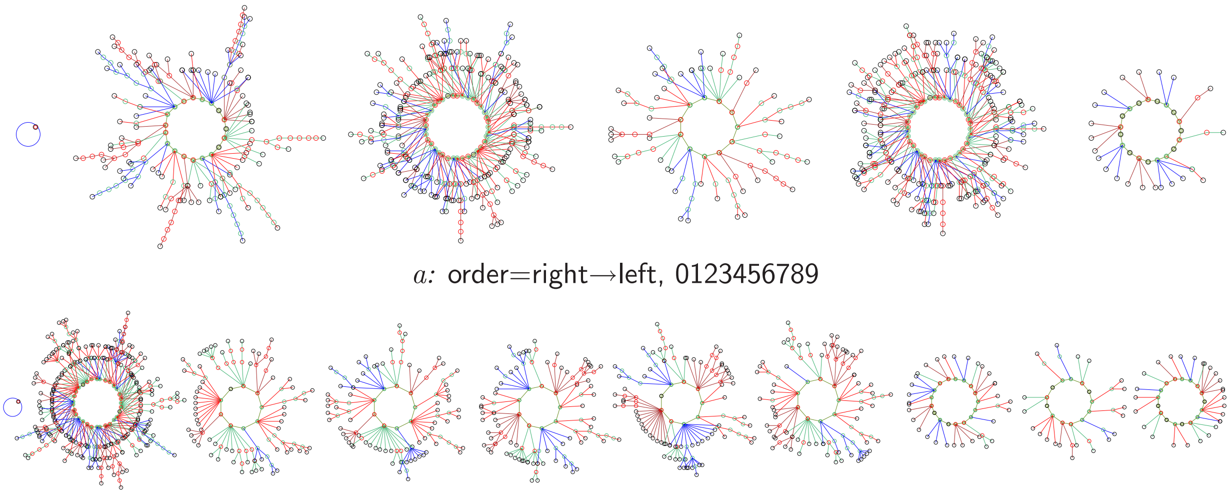
>-(def) . . . enter **return** for the default order from left to right in a 1d network, or in general counting down the cell index $n-1$ to 0 (section 10.6).

<-b . . . for the opposite order, from right to left, cell index 0 to n-1.

rnd-r . . . for a random order (section 29.9.1).

set-s . . . to set a specific order by hand (section 29.9.2).

all-S . . . *(if $n \leq 12$)* to list all n! possible orders and select a specific order from the list (section 29.9.3).



a: order=right→left, 0123456789

b: randomly ordered, order=4536780129

Figure 29.7: A basin of attraction field for a 1d CA with sequential updating, $v2k3$ rule (dec)110, n=10. a: sequential left to right, b: randomly sequential. The order is shown in the lower rule window.

29.9.1 Random order

Enter **r** in section 29.9 to set a random order. A top-right prompt shows the order (which can be revised), together with the order-index in decimal and hex if $n \leq 12$, for example,

```
random order, n=12
5 0 1 3 10 9 4 8 11 7 2 6 (dec)=199658374 (hex)=be68b84 (for n=14)
update-tags shown from 0-11
save/load-s/l >-f <-b re-order-r cont-ret:
```

In the example above, cell index 5 (update-tag 0) is the first to be updated, and cell index 6 (update-tag 13) is the last to be updated. The info and options are described below,

info and options ...	*what they mean*
n=12 ...	the system size.
5 0 1 ... 6 ...	the random order as a list of cell indexes.
(dec)= ...	*(if $n \leq 12$)* the order-index in decimal, a mumber defining the order.
(hex)= ...	*(if $n \leq 12$)* the order-index in hexadecimal.
update-tag shown from 0-11 ...	reminder of the update-tags.
save-s ...	to save the order to a `.ord` file (see Filing, chapter 35).
load-l ...	to load an order.
>-f ...	for a left to right order, as in section 29.9.
<-b ...	for a right to left order, as in section 29.9.
re-order-r ...	for a new random order.
cont-ret ...	to accept the order and continue.

29.9.2 Set specific order

Enter **s** in section 29.9 to set a specific order by hand. The sequence of cell indexes is set with the following top-right prompt,

enter cell index (no repeats) 0-11 or rnd-(def) back-b
update 4: 4 8 10 3 * * * * * * * * *(for n=12, for example)*

In this example, update-tags 0 – 3 have been entered as cell indexes 4, 8, 10, 3, and the cell index for update-tag 4 is requested. A prompt for each update-tag from 0 to n-1 is presented in turn. Stars (*) indicate unallocated update-tags, otherwise the allocated cell indexes are shown. Enter a unique cell index (repeats will be rejected) for each sucessive update-tag. **return** gives a valid random cell index. Enter **b** to backtrack to the previous update-tag.

Once the selected sequence is complete, the following top-right prompt is presented,

sequence complet 0-11 *(for n=12)*
4 8 10 3 0 9 11 1 6 5 2 7 (dec)=188097014 (hex)=b3621f6
save/load-s/l reselect-r cont-ret:

info and options ...	*what they mean*
0-11 ...	the range of cell indexes, and update-tags for n=12.
4 8 10 3 ... **7** ...	the sequential order as a list of cell indexes.
(dec)= (hex)= ...	*(if $n \leq 12$)* the order-index in decimal and hex, as in section 29.9.1.
save/load-s/l ...	to save or load the order to a `.ord` file as in section 29.9.1.
reselect-r ...	to start again – set a new order.
cont-ret ...	to accept the order and continue.

29.9.3 List all orders

$n \leq 12$ only

Enter **S** in section 29.9 to list all n! possible orders and select a specific order. As many orders as will fit will be displayed in a window on the right of the screen, followed by further options. For example, for n=12 (the maximum list size permitted) the first two orders and last two orders in the list are shown below,

all possible orders n=12, 0-479001599
decindex - order - hexindex
0 - 0123456789ab - 0 *(this is the right to left order)*
1 - 0123456789ba - 1

... *(the list continues, the last two order are shown below)*

479001598 - ba987654b3201 - 1c8cfbfe
479001599 - dcba9876543210 - 1c8cfbff *(this is the left to right order)*
jump/copy-j orderseed-s accept-a more-ret: *(or **top-ret** if on the last page)*

The center column is the sequential order, with cell indexes shown in hex for compactness. The left and right columns show the order-index (0 to n!) which specifies and codes for each order. The left column is in decimal, the right in hex.

The prompts on the bottom line are explained below,

options . . . *what they mean*

jump/copy-j . . . to jump to a new initial order-index. **jump to index (0-479001599):** *(for* $n = 12$*)* appears at the bottom of the list. Enter a decimal order-index, the list will be redrawn with that index at the top. Then **a** may be entered to accept the top selection.

orderseed-s . . . to set the order-index as if it were a seed – see section 29.9.4.

more-ret . . . the default to show the next window of entries in a long list.

top-ret . . . the default if on the last page (or single page) to re-list from the top.

29.9.4 List all orders – set order seed

Enter **s** in section 29.9.3 to set the order-index as if it were a seed (chapter 21). A lower left *order seed* prompt appears with similar options[4] to the seed prompt (sections 21.1,21.2),

Select order seed, max=(dec)479001598=(hex)1c8cfbff prtx-x
rnd-r bits1d-b bits2d-B hex-h dec-d (def-d): *(for* $n = 12$*)*

These options are fully described in section 21.2 and further sections in chapter 21, and are briefly summarised below,

options . . . *what they mean*

dec-d . . . *(the default)* enter **return** or **d** for the order seed in decimal, if the number is greater than **max** – **too big! back-ret** will appear to re-enter.

rnd-r bits1d-b bits2d-B hex-h . . . any of these methods also apply for the order seed If the entry exceeds **max**, the following message appears re-enter,

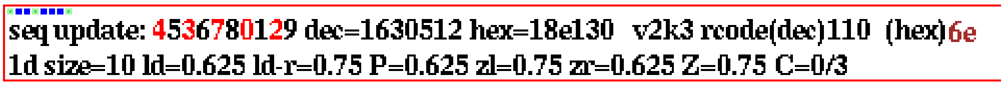
order index 55889 $>$ **max(40319) cont-ret:** *(for* $n = 8$*)*

As soon as the list (section 29.9.3) and its prompts reactivate, enter **a** to accept the selection.

29.9.5 Drawing the attractor basin with sequential updating

Once the sequential order has been set, the exhaustive pairs prompt will be shown (section 29.7). – enter **return** to compute[5] while displaying the prompt in section 29.7.1.

Once the exhaustive pairs are complete, the attractor basin will be drawn. At the same time, the "rule window" will be displayed at the bottom of the screen, showing the rule (or the rule at index 0 for an RBN), and other information (section 16.19), and in addition the updating order is shown in alternating red-black colors. For $n \leq 12$ the sequential order is shown in hex, and the order-index is also shown in decimal and hex. For $n > 12$ just the the sequential order is shown in decimal.



seq update: 4536780129 dec=1630512 hex=18e130 v2k3 rcode(dec)110 (hex)6e
1d size=10 ld=0.625 ld-r=0.75 P=0.625 zl=0.75 zr=0.625 Z=0.75 C=0/3

Figure 29.8: The rule window for sequential updating. This example is for $v2k3$ rcode (dec)110, n=10, and a sequential order as in figure 29.7b.

[4] Note that some of the sub-options in section 29.9.4 may be irrelevant in the context of the order seed.

[5] This may take some time for larger networks, as noted in section 29.7)

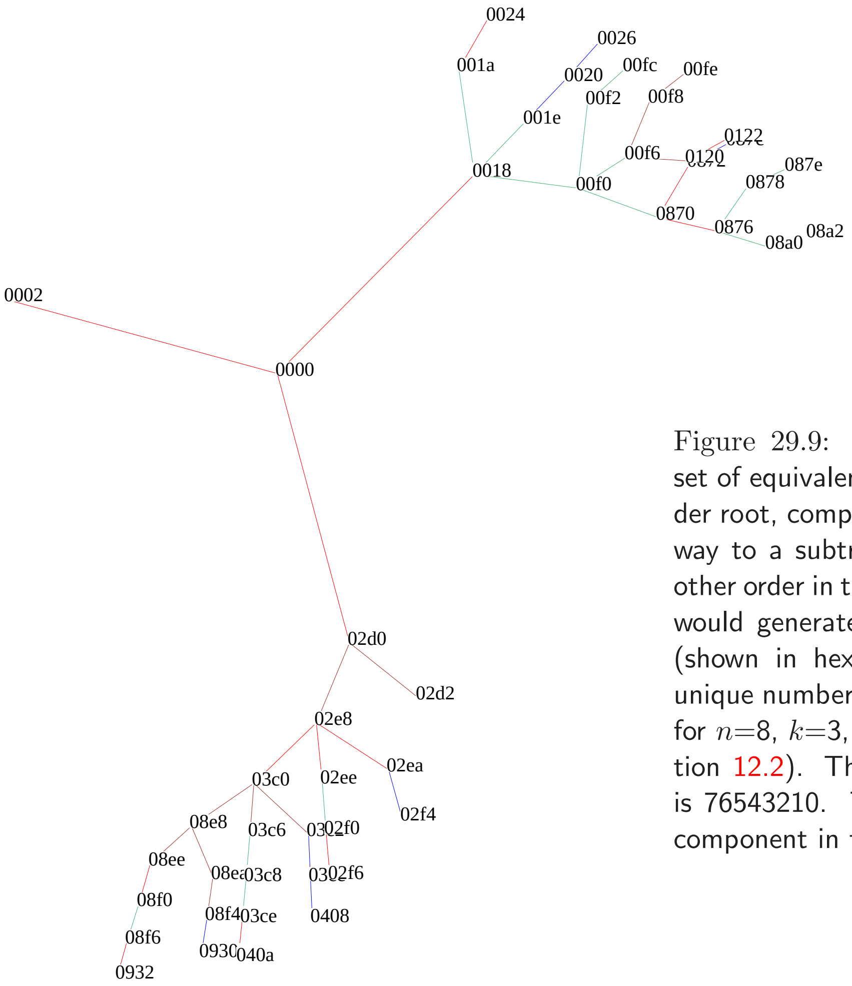


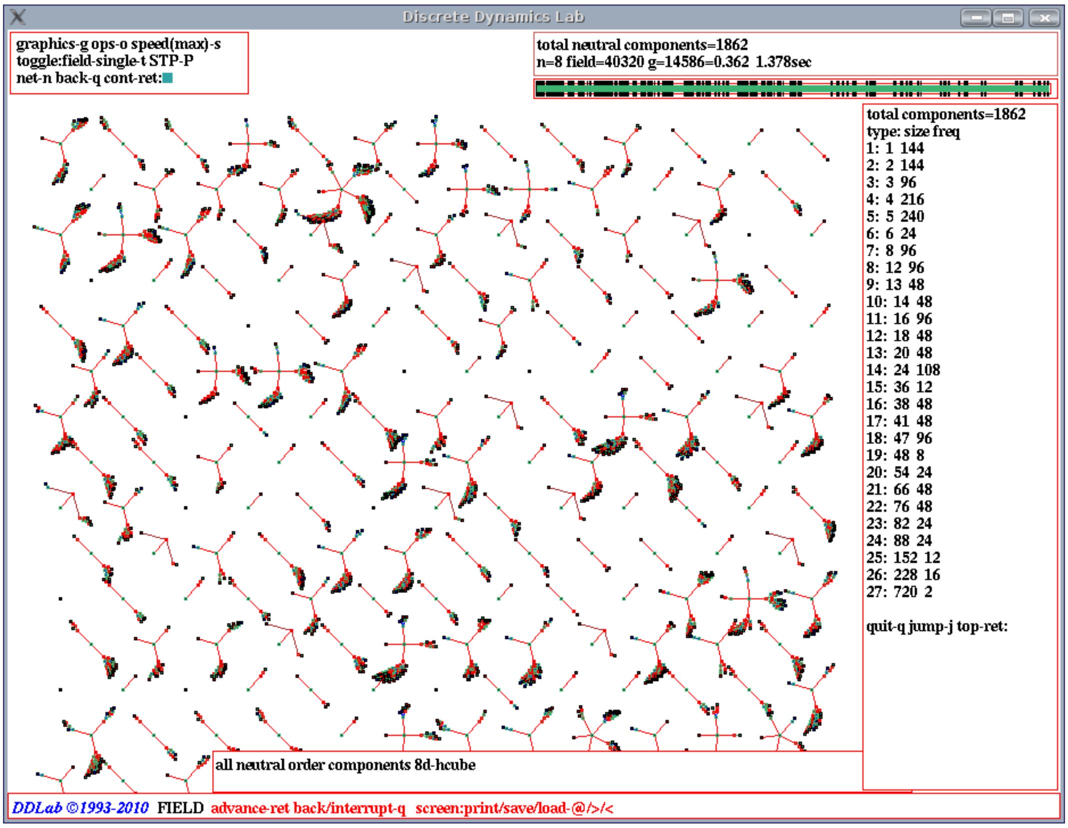
Figure 29.9: A neutral order component, a set of equivalent orders generated from the order root, computed and drawn in an analogous way to a subtree for network dynamics. Any other order in the component acting as the root would generate the same component. Nodes (shown in hex) represent the order-index, a unique number for each order. This example is for n=8, k=3, with 8d-hypercube wiring (section 12.2). The order seed=0, i.e. the order is 76543210. The subtree is the first neutral component in the neutral field in figure 29.10.

29.10 Neutral order components

for v=2 and $n \leq 12$ and only, and a basin field, or a subtree with zero forward steps (section 29.2)

In sequential updating, different orders may produce equivalent dynamics. Its of some interest from a mathematical perspective to compute the neutral order components, how sequence space is divided up into sets of orders with identical dynamics. This turns out to depend on the network's wiring scheme. The algorithm in DDLab that computes neutral order components (limited to network size $n \leq 12$), does so by starting with a particular order (the order seed), and generating its equivalent orders by swapping the order of pairs of neighboring cells that have no links to each other, neutral pairs. Valid swaps within the first order are treated as “pre-images”, and these pre-images may have pre-images in turn (repeat orders are discounted). When no valid swaps remain the neutral component is complete.

DDlab treats these components as subtrees. A single component subtree (figure 29.8), or the entire set of components subtrees (figure 29.10) making up sequence space (the neutral field) can be computed and drawn in an analogous way to a single subtree or basin of attraction field for network dynamics. The layout and node label options described in chapter 26 apply.



```
graphics-g ops-o speed(max)-s
toggle:field-single-t STP-P
net-n back-q cont-ret:
```

```
total neutral components=1862
n=8 field=40320 g=14586=0.362  1.378sec
```

```
total components=1862
type: size freq
1: 1 144
2: 2 144
3: 3 96
4: 4 216
5: 5 240
6: 6 24
7: 8 96
8: 12 96
9: 13 48
10: 14 48
11: 16 96
12: 18 48
13: 20 48
14: 24 108
15: 36 12
16: 38 48
17: 41 48
18: 47 96
19: 48 8
20: 54 24
21: 66 48
22: 76 48
23: 82 24
24: 88 24
25: 152 12
26: 228 16
27: 720 2

quit-q jump-j top-ret:
```

```
all neutral order components 8d-hcube
```

```
DDLab ©1993-2010  FIELD  advance-ret back/interrupt-q  screen:print/save/load-@/>/<
```

Figure 29.10: The neutral field, the subtrees making up sequence space – the first 175 or so, out of a total of 1862, are shown – computed and drawn in an analogous way to a basin of attraction field for network dynamics. A list of component sizes and the frequency of different sizes is shown in a window on the right once the neutral field is complete. This example is for n=8, k=3, with 8d-hypercube wiring (section 12.2). The first subtree (top right, order seed=0) is shown in detail in figure 29.9.5

Nodes are labeled according to the order-index (a unique number, 0 to $n!-1$ (see section 29.9.3), in decimal, hex, as a bitstring, etc as previously selected in section 26.2. The default is as a spot.

Enter *nto*-**o** at the prompt in section 29.4 to generate the neutral subtree (section 29.10.1) or neutral field (section 29.10.2), depending on whether a subtree or basin of attraction field is currently selected[6]. Note that the rule or rule scheme, or a subtree's initial state, selected previously, plays no role in computing neutral order components.

29.10.1 Neutral subtree

The generate a neutral subtree, select **backward for subtree-b** in section 29.1, then **return** in section 29.2 to ensure zero steps for **forwards before backwards?**. Then enter *nto*-**o** in section 29.4 for a neutral subtree. A lower center window prompts for the neutral order seed, or subtree root,

[6] The neutral order prompt **nto-o** is not activated if a single basin of attraction was selected in section 29.1).

Select order seed, max=(dec)40319=(hex)9d7f prtx-x
rnd-r bits1d-b birs2d-B hex-h dec-d (def-d): *(for n=8)*

Enter an order-index, 0 to $n! - 1$ as the seed, in decimal, hex, as a bitstring, etc. (as section 29.9.4). The selection works in the same way as selecting a network seed, described at length in chapter 21. A selection greater than $n! - 1$ will be rejected, with the message,

order-index 44444 > max(4318) cont-ret: *(for n=8), for example)*

The following examples relate to figure 29.9.5, for n=8, k=3, with 8d-hypercube wiring, (see section 12.2) As the subtree is dawn, a message appears in a window at the bottom of the screen,

one neutral component 8d-hypercube

When the subtree is complete, a top-right window gives some data on the subtree as follows,

neutral component size=48 root(hex)=00 00
g=16=0.333 ml=8 mp=3   0.006sec

29.10.2 Neutral field

Enter *nto*-**o** in section 29.4 for a neutral field. All neutral order components making up sequence space will be generated[7] (figure 29.10). The order components are drawn as a series of "subtrees" (figure 29.9.5), and the data on each subtree appears in a top-right window while the next subtree is being drawn (section 29.10.1 describes the data). In order to pause to see the data for each subtree, select a pause in section 27.3.

Once the neutral field is complete, a top-right window shows data on the last subtree, and component types are listed on the right of the screen. The list gives the type according to size, the size itself, and the frequency of different sizes. As many component types as will fit will be listed, followed by further options. For example, for n=8, k=3, with 8d-hypercube wiring, shown in figure 29.10, the list is as follows,

total components=1862
type: size freq
1: 1 144
2: 2 144
3: 3 96
4: 4 216

. . . *(the list continues, the last three types are shown below)*

25: 152 12
26: 228 16
27: 720 2
quit-q jump-j top-ret: *(***top-ret** *if on the last page)*
or
quit-q jump-j more-ret: *(***more-ret** *if more than one window is required)*

[7] Although all neutral order component subtrees are generated, you only see those that fit within the DDLab screen, which depends on the layout and scale set in chapter 25.

The prompts on the bottom line are explained below,

options . . . *what they mean*

quit-q . . . Enter **q** to backtrack and show a summery of the neutral field in a top-right window, as in figure 29.10, for example,

```
total neutral components=1862
n=8 field=40320 g=14585=0.362     1.345sec
```

The neutral field list will remain visible but inactive.

jump-j . . . to jump to a new type index (useful for a long list),

```
jump to index (0-100):
```
(for example)

appears at the bottom of the list. Enter a type index, the list will be redrawn with that index at the top.

more-ret . . . the default to show the next window of entries in a long list.

top-ret . . . the default if on the last page (or single page) to re-list from the top.

Its a good idea to drastically reduce the scale of the subtrees in section 25.2 as there are usually many small neutral order components. To reserve a vacant space for the list window and not to hide some subtrees, set the minimum right border at a high value in section 25.2.7.

Intentionally Blank

Chapter 30

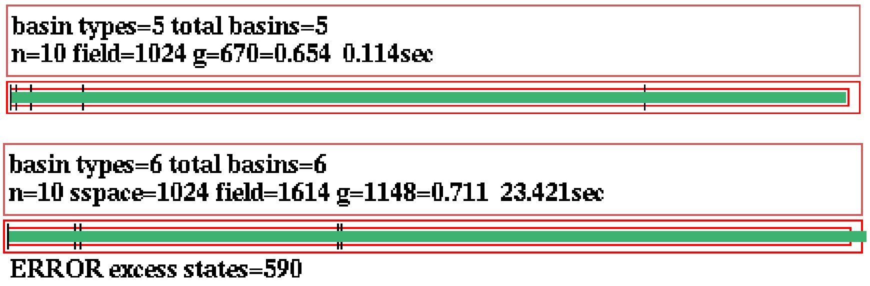
Drawing attractor basins, and changes on-the-fly

Following the final options described in section 29.4, the drawing of attractor basins will commence, according to the current presentation, layout and other settings. During the drawing, some of these settings can be reset, either by pausing the drawing and responding to further prompts, or immediately, on-the-fly, by various key hits. Many network parameters may also be reset without the need to backtrack to the early prompts. This chapter describes these options.

Note also that if one of various "pause" options was selected in section 27.3, the angular orientation, spacing and/or position of each successive basin, tree or subtree can be amended during the pause (section 25.3).

30.1 The progress bar for basin of attraction fields

For basin of attraction fields only, a horizontal green progress bar below the data window (section 27.5) indicates how much of state-space has been used up so far. A bar that over- or under-shoots indicates an error. The vertical bars on the horizontal bar indicate the seeds (by their decimal equivalent distance along the bar) for successive basins, starting with 0 on the left.

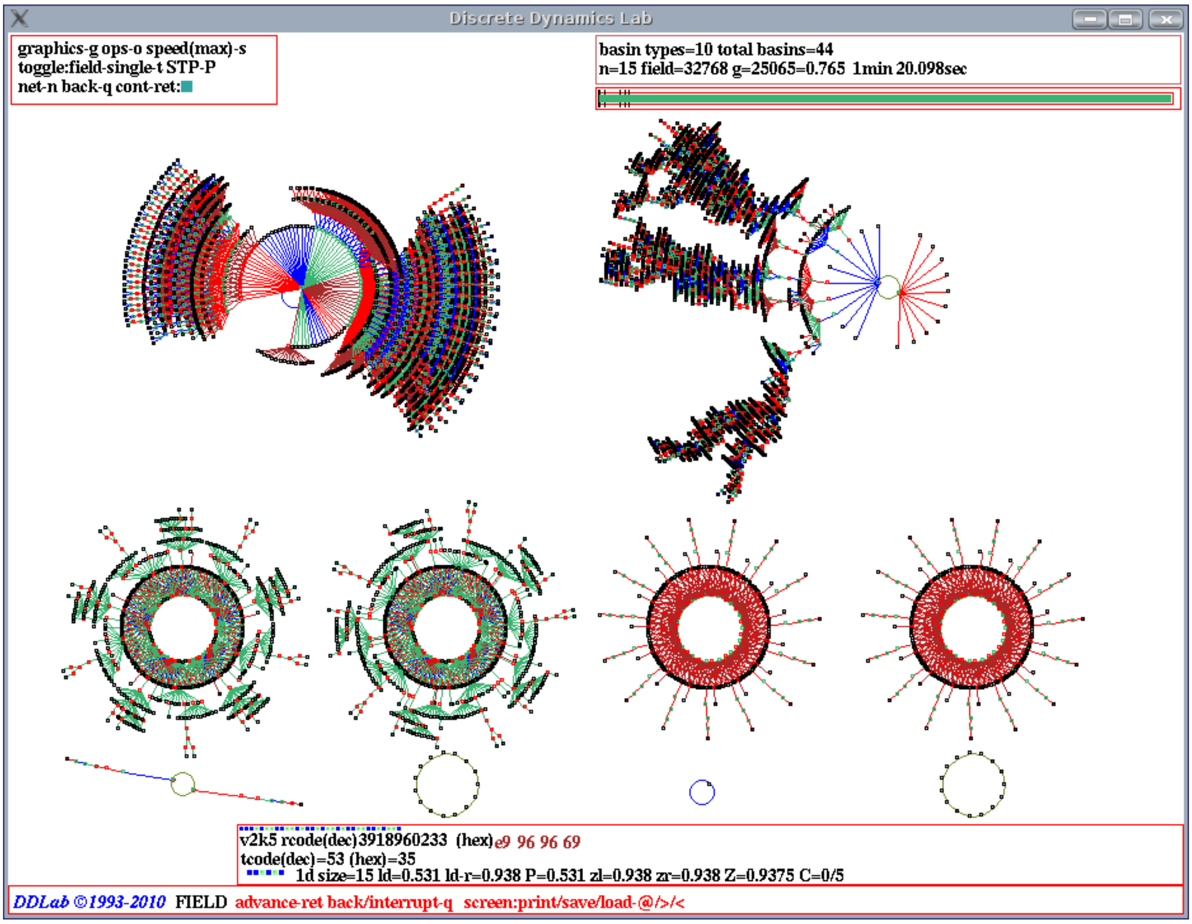


```
basin types=5 total basins=5
n=10 field=1024 g=670=0.654  0.114sec
```

```
basin types=6 total basins=6
n=10 sspace=1024 field=1614 g=1148=0.711  23.421sec
```

```
ERROR excess states=590
```

Figure 30.1: Examples of the basin of attraction field progress bar and the data window, for an RBN. *top:* For a normal run without error. *bottom:* An example where the wiring was changed during a pause, leading to inconsistent data where the size of the field is greater than state-space. The progress bar has gone off its scale on the right.



graphics-g ops-o speed(max)-s
toggle:field-single-t STP-P
net-n back-q cont-ret:■

basin types=10 total basins=44
n=15 field=32768 g=25065=0.765 1min 20.098sec

v2k5 rcode(dec)3918960233 (hex)e9 96 96 69
tcode(dec)=53 (hex)=35
1d size=15 ld=0.531 ld-r=0.938 P=0.531 zl=0.938 zr=0.938 Z=0.9375 C=0/5

DDLab ©1993-2010 FIELD advance-ret back/interrupt-q screen:print/save/load-@/>/<

Figure 30.2: The DDLab screen showing a basin of attraction field. This example is for a 1d CA, n=15, $v2k5$ tcode 53. To achieve this layout, a pause was selected after each basin in section 27.3, and the position and spacing of basins were amended as described in section 25.3. Various typical indications on the screen are as follows,

top-left: ...	the "attractor basin complete" options window (section 30.4).
top-right: ...	the data window (section 27.5).
below top-right: ...	the progress bar (section 30.1).
bottom-centre: ...	the rule window (section 16.19.
foot of screen: ...	the title bar (section 5.5). While attractor basins are being drawn, this also shows reminders about on-the-fly options (section 30.3).

30.2 Attractor basins, interrupting and changing

Enter **q** to interrupt attractor basins with a top right prompt. If the interrupt occurred between successive pre-images within one pre-image fan (as opposed to between successive pre-image fans) this is indicated in an extra first line,

early exit - in pre-image fan

Then one of the following prompts is presented, depending on whether the attractor basin is a subtree, single basin or a basin of attraction field,

for a subtree
options-o stopsubtree-q speed-s cont-ret:
for a single basin
next-tree-n options-o stopbasin-q speed-s cont-ret:
for a basin of attraction field
next-tree/basin-n options-o stopfield-q speed-s cont-ret:

If the in-degree histogram was set in section 24.6, a further option **inhist-h** is offered to view the histogram as computed so far, for example,

inhist-h, options-o, stop subtree-q, cont-ret: *(for a subtree)*

There options are summarised below, with some described in more detail in the rest of this section,

options ...	*what they mean*
inhist-h ...	to view the in-degree histogram (section 24.6).
next-tree-n ...	to abandon the tree that is currently being generated and start the next tree. This will create errors in a basin of attraction field (section 30.2.1).
options-o ...	to show a truncated version of the top-left "attractor basin complete" options (section 30.4).
speed-s ...	to slow down drawing attractor basins and backwards space-time patterns or revert to full speed, as in section 24.15. Slow motion can also be invoked on-the-fly (section 30.3), from the "attractor basin complete" prompt (section 30.4), and beforehand from section 24.4.
stopfield-q ...	(or **stopbasin-q**, or **stopsubtree**) to abandon the attractor basin that was interrupted (section 30.2.3).
cont-ret ...	enter **return** to continue the attractor basin from the point of interrupt.

When the implimentation of these options is finished, the attractor basin will either continue from the point at which it was interrupted, or if certain parameters were changed, including a new rule or seed, a new attractor basin will be started.

30.2.1 Abandoning a tree and continuing with the next tree

Enter **n** in section 30.2, to abandon the tree that is currently being generated and start the next tree. This also applies to a subtree from the uniform states (section 26.1.2). If compression (section 26.1) was not suppressed in section 26.1.1 all the equivalent trees (or subtrees from the uniform states) will be abandoned, and the next set of equivalent trees started.

Abandoning a subtree before it is complete will result in errors in attractor basin data (section 27.5). For a basin of attraction field, the progress bar is likely to go off its scale, with a message such as **ERROR excess states=19124** below it. This indicates that more states were computed than the size of state-space.

30.2.2 Errors in attractor basins

Its important to note that any changes to the network parameters made during a pause or interrupt will inevitably result in errors/inconsistencies in data and attractor basin structure. The **view/revise/learn-n** option allows the network to be changed. If changes are made during a pause, and the attractor basin is then continued, the result will be inconsistent with any given network. For a basin of attraction field, the progress bar is likely to go off its scale. The size of state-space as well as the field will be indicated in the data window. For example,

... sspace=1024 field=1486

A message will also appear below the progress bar. For example,

ERROR excess states=462

This indicates that more states were computed than the size of state-space.

Note also that although the wiring in a 1d CA may be examined and changed during the pause, the change is only temporary, and will be automatically restored to allow the attractor basin of the 1d CA to continue using the 1d CA reverse algorithm. For other networks, RBN and 2d or 3d CA this does not apply, and the wiring can be changed during a pause, but of course this will lead to inconsistencies and errors as described earlier. Rule changes during a pause will remain in effect for any network, again leading to inconsistencies and errors.

30.2.3 Abandon the attractor basin

Enter **q** in section 30.2 to abandon the attractor basin that was interrupted. The data on the incomplete attractor basin is shown in a top-right window (section 27.5), preceded by the words "EARLY EXIT" to indicate that the data applies to an incomplete attractor basin. For example,

EARLY EXIT basin types=7 totalbasins=76
n=15 sspace=32768 field=23222 g=16711=0.72

At the same time the "attractor basin complete" prompt is presented in a top-left window (section 30.4).

30.3 Attractor basin options, on-the-fly

A number of key hits allow changing the presentation of attractor basins on-the-fly – the following reminder is shown in the bottom title window, depending on the type of system selected in section 29.4,

for non-local wiring, which includes the full set of options
matrix-m STP:tog/scroll/exp/contr-s/#/e/c ppstack:tog-P slow/max</>

The option **ppstack:tog-P** applies only for the non-local wiring reverse algorithm (section 29.6.2). The option **matrix-m** does not apply for neutral order components (section 29.10).

on-the-fly key hits . . . *what they means*

matrix-m . . . Toggle the state-space matrix (section 24.5) on and off.

tog/scroll/exp/contr-s/#/e/c. . . Enter **s** to toggle backwards space-time patterns (section 24.14) on and off, # to toggle between scrolling (the default) and sweeping the patterns, **e/c** to expand or contract the scale of both the backwards space-time patterns and the nodes of the basins when showm as bit/value patterns (section 26.2).

ppstack:tog-P . . . *(for the non-local reverse algorithm)* enter **P** to toggle showing the partial pre-image histogram (section 29.6.2). A top-right prompt is presented,

partial pre-image histogram: start pause-p:

Enter **p** to pause the histogram – section 29.6.2 gives further details.

slow/max:</> . . . slow down with < which halves computation speed each time its hit. Enter > to revert to maximum speed.

Computation may be too fast for on-the-fly key hits – it may be slowed during an interrupt (sections 30.2, at the "attractor basin complete" prompt (section 30.4), or beforehand (sections 24.15 or 24.4). The key hits are described below,

30.4 Attractor basin complete prompt

When the attractor basin is complete, or if abandoned (section 30.2.3), the following options are presented in a top-left window,

graphics-g ops-o speed(max)-s *(or* **speed(slow)-s** *if not max)*
tog: single-field-t STP-P *(or* **field-single-t** *whichever is active)*
seed-e net-n, back-q cont-ret: *(***seed-e** *for single basin or subtree)*

Enter **q** to backtrack to previous prompts. Enter **return** to generate another attractor basin, for a mutated network according to the mutations set in chapter 28, all other parameters and settings remaining unchanged.

If **o** was entered when interrupting in section 30.2, the prompt is truncated with fewer options as follows,

speed(max)-s *(or* **speed(slow)-s** *if not max)*
STP-P
net-n, back-q cont-ret:

Enter **q** or **return** to revert to the interrupt prompt (section 30.2.

The other options are summarised below,

option . . . *what it means*

graphics-g . . . to alter the graphics setup described in section 6.3.

ops-o . . . to open up a range of further rule, network and seed options, which are presented in a top-right window (section 30.5).

speed(max)-s . . . showing the current speed status **(max)** or **(slow)**. Enter **s** to slow down drawing attractor basins and backwards space-time patterns or revert to full speed. as in section 24.15. Slow motion can also be invoked on-the-fly (section 30.3), while interrupting (section 30.2), and beforehand from section 24.4.

tog: single-field-t . . . (or **field-single-t**) to toggle between a single basin and the basin of attraction field. When changing to a single basin the prompt sequence restarts at the seed prompt (chapter 21). When changing to a field the prompt sequence restarts at the first output parameters prompt (section 24.1). Note that the "layout" defaults (chapter 25) may need to be reset.

tog: STP-P . . . to toggle between showing and not showing "backwards" space-time patterns (see section 24.14).

seed-e . . . (*for a single basin or subtree*) to revise the seed for a single basin or subtree. The seed options described in chapter 21 are presented in a lower central window. This option does not appear for a basin of attraction field.

net-n . . . for the network architecture options described in chapter 17, including viewing, revising and filing, the Derrida plot (chapter 22, and learning (chapter 34), a repeat of the option in section 30.4.

30.5 Further attractor basin complete options

Enter **o** in section 30.4 above to display a range of further (context dependent) options for amending the rule, the network, or seed (for a single basin or subtree onle). A top-right prompt similar to the following is presented,

rule:save/load/revise/rnd/trans-s/l/v/r/t net-n
bitflip-1, allv-b, Z:higher/lower-Z/z canal-C *(***Z:** *for single rule networks only)*
seed:rev-e rnd/blk-R/k sng:pos/neg-5/6 save/load-S/L *(this line for a single basin or subtree)*
goback-q cont-ret:

Enter **q** to backtrack to previous prompts in section 30.4.

Enter **return** to generate another attractor basin, for a mutated network according to the mutations set (chapter 28). Mutations are only put into effect if the rule and seed parameters were not revised (below).

The remaining options are listed in two categories, revising the rule (section 30.5.1), or the seed (section 30.5.2) – for single basins or subtrees only. Changes to rules or seeds generaly result in new attractor basins being started. However, changes made during an interrupt when attractor basins are incomplete, and continue from the point of interrupt, will result in errors as described in sections 30.2.2.

30.5.1 Attractor basin - revising rule/s

Part of the prompt (section 30.5) for revising the rule, or rules in mixed rule networks, is as follows,

rule:save/load/revise/rnd/trans-s/l/v/r/t, net-n
valueflip-1, flip ends-w/b, Z:higher/lower-Z/z canal-C *(***Z:** *for single rule networks only)*

The meaning of the options are summarised below,

options . . . *what they mean*

save/load-s/l . . . enter **s** or **l** to save or load the rule as a `.rul` file (see Filing, chapter 35). For mixed rule networks this will apply to the rule at cell index 0 only.

revise-v . . . to revise the rule. For mixed rule networks the following prompt is presented below the top-right window to select the cell index,

enter cell index, 9-0: *(for example)*

A lower right prompt appears to revise the rule, similar to the methods in chapter 16.

rnd-r . . . to reset a new random rule, or all rules at random in a rule mix.

trans-t . . . to transform the the rule as described in chapter 18. For mixed rule networks this will apply to the rule at cell index 0 only (section 18.5), but Canalizing inputs can be set for the whole network (chapter 15).

net-n . . . for the many network architecture options described in chapter 17, including viewing, revising and filing, the Derrida plot (chapter 22), and learning (chapter 34).

valueflip-1 . . . for a random bit/value-flip in the rule-table, or for all rule-tables in a rule-mix.

allv-b . . . to flip outputs of uniform nhoods to the nhood value, all0s→0, all1s→1, all2s→2 etc. – in a single rule, or for all rules in a rule-mix.

Z:higher/lower-Z/z . . . *(for single rule network only)* enter **Z** to progressively force the Z-parameter higher, or enter **z** to force it lower, as in section 16.3.

canal-C . . . to reset canalizing inputs in one rule (as in section 18.5), or for all rules in a rule-mix (as in chapter 15).

Once the rule is revised it will be shown in the lower rule window described in section 16.19.

30.5.2 Attractor basin - revising the seed

for a single basin or subtree only

Part of the prompt (section 30.5) for revising the seed, which applies to a single basin or subtree only, not for a basin of attraction field, is as follows,

seed:rev-e rnd/blk-R/k sng:pos/neg-5/6 save/load-S/L

The meaning of the options are summarised below,

options ...	*what they mean*
rev-e ...	to revise the seed (also available in section 30.4). A seed prompt (as in chapter 21) is presented in a lower center window.
rnd/blk-R/k ...	enter **R** for a new random seed. Enter **k** for a random central block of cells, the remainder set to 0. The block size was specified in section section 21.3.
sng:pos/neg-5/6 ...	*for $v = 2$:* enter **5** for a positive singleton seed, a central cell set to 1, the remainder set to 0. Enter **6** for a negative singleton seed. *For $v \geq 3$:* enter **5** for a a central cell set to any random value except zero against a zero background. Enter **6** for a central cell with any random value against a different uniform background.
save-S ...	enter **S** to save the original seed, or the current state – the latest state in the attractor basin. The following top-right prompt is presented, **save: original seed-s, current state-c:** Then save the state – an `.eed` file, (see Filing, chapter 35).
load-L ...	enter **L** to load a new seed – an `.eed` file (see Filing, chapter 35).

Chapter 31

Output parameters for space-time patterns

This chapter describes output parameter options that allow various default settings to be revised before space-time patterns start. Many of these options can also be invoked on-the-fly (chapter 32), and also from the interrupt window (section 32.14). Although there are many output parameter options and sub-options, all have defaults. The options may be skipped unless particular defaults need to be changed, and its possible to jump directly to a particular category of options, namely *analysis* and *histograms*, apart from the first set of of miscellaneous options.

The options apply only if the network was first set up to run "forward only" and see just space-time patterns. This means that a "single basin/subtree" should have been selected at the first DDLab prompt in section 6.2 by entering **s** (a seed would also have been set). Also, "space-time patterns only" should have been selected at the the first output parameter prompt for attractor basins (section 24.1) enter **s**, or alternatively at a final stage at the "subtree or single basin" prompt (section 29.1 enter **S**).

31.1 The first output parameter prompt for space-time patterns

If **s** is selected in section 24.1 (or section **s** in section 29.1), the first output parameter prompt for attractor basins is presented in a top-right window as follows,

FORWARD ONLY options: analysis-a histograms-h (ALL-def):

Enter **d** to accept all output parameter defaults and skip these options. This can be done at any stage in the output parameter prompt sequence to accept remaining defaults and skip remaining options. A reminder to this effect **accept defaults-d** remains displayed in a top center window.

All the output parameter options may be looked at in turn, but they are divided into four categories to allow jumping directly to the category where options need to be set. The categories are,

revise from start-ret . . . various miscellaneous options difficult to categorize: inverting the kcode in TFO-mode, sequential updating, cell color by neighborhood or value, options for pausing or stepping through space-time patterns, displaying space-time patterns in other than the "native" dimension, scrolling, and probabilistic networks.

analysis-a ... various methods of viewing and analyzing space-time patterns, including pattern density, input-frequency/entropy, applied to the whole network or one chosen cell. The size of the window of time-steps for these analyses, and the threshold for 'frozen" cells, can be amended.
Plots of the "state-space matrix", and the "return map".

histograms-h ... various statistical methods of analysing space-time patterns, where the data is displayed as a histogram, including damage spread from perturbations (1d and 2d only), and data on attractors and "skeletons".

Enter **a** to skip directly to the *analysis* category of options (section 31.9), or enter **h** for the *histogram* category (section 31.15). Enter **return** for the first set of miscellaneous options, described below,

31.2 Inverting the kcode in TFO-mode

for kcode in TFO-mode only

The kcode-table can be expressed according to two alternative conventions. In DDLab the the all-v-max is on the left and all-0's on the right. To allow rules with the opposite convention to run as intended, the following top-right prompt is presented,

invert kcode value order -A:

Enter **A** to run the kcode according to the inverted convention, or return to maintain DDLab's standard convention. The kcode itself is not changed, just the way it is accessed while running forward. This is similar to inverting the rcode in section 18.3 but in that case the rcode itself is changed.

31.3 Sequential updating, space-time patterns

By default, network updating is synchronous, in parallel. Alternatively, the updating can be sequential. There are $n!$ possible sequential updating orders in a network of size n. DDLab provides simple default orders, forward or backward along the network index, or a random order. For $n \leq 31$, any specific order can also be set, filed and loaded (from a `.ord` file). These options are the same as in section 29.9 for attractor basins, so are not described in detail here. Note that parallel and sequential updating (whatever order was set) can be toggled on-the-fly with **U**.

If **return** is entered in section 31.1, the following top-right prompt is presented,

sequential updating-s, parallel-def:

Enter **s** to select sequential updating. The following top-right prompt is presented to specify the order,

for $n > 31$
sequential updating:->(def) <-b rand-r:
for $n \leq 31$
sequential updating:->(def) <-b rand-r set-s all-S:

For $n > 31$, usually the case when studying space-time patterns, the sequential order can be specified as the default left to right (enter **return**) or the opposite (backward) order, from right to left (enter **b**). Enter **r** for a random order, see section 29.9.1 for details.

For $n \leq 31$, a specific order can be set (enter **s**), see section 29.9.2 for details. To list all possible orders (enter **S**), an order can be copied from the list, see section 29.9.3 for details.

31.4 Color cells by value or neighborhood

By default each cell is colored according to the rule-table index that was "looked up" to determine the cell's value, in other words the cell's neighborhood at the previous time-step. These colors are assigned from a palette of 16 (see section 6.3.4 for the color code). Alternatively cells may be colored simply according to their value, 0 or 1. Figures 31.1, 31.2 and 31.3 show the difference between neighborhood and value display of space-time patterns for 1d, 2d and 3d (these were all generated form a single central 1 seed).

To select color by value, the following prompt is presented in a top-right window,

color cells by value-v, by nhood-def:

Enter **v** to color cells by value, or accept the default, by neighborhood. Note that this can be toggled on-the-fly (see section 32.8.3). Other color presentations that depend on "frozen" cells, the frequency of 1s in a time window, and filtering space-time patterns, can also be toggled on-the-fly (see section 32.10).

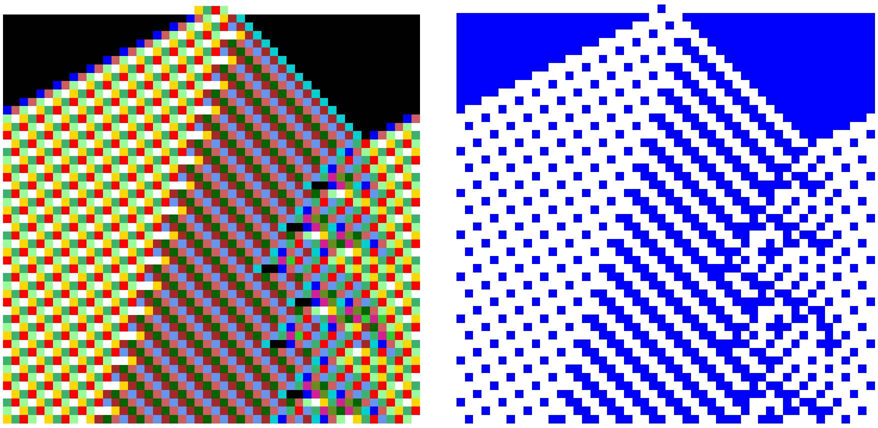


Figure 31.1: 1d space-time pattern colors, 50 time-steps for a CA n=50, $k = 5$, rule 82 26 dc 23. Space is across, time is down the page.
left: by neighborhood, the default.
right: by cell value.

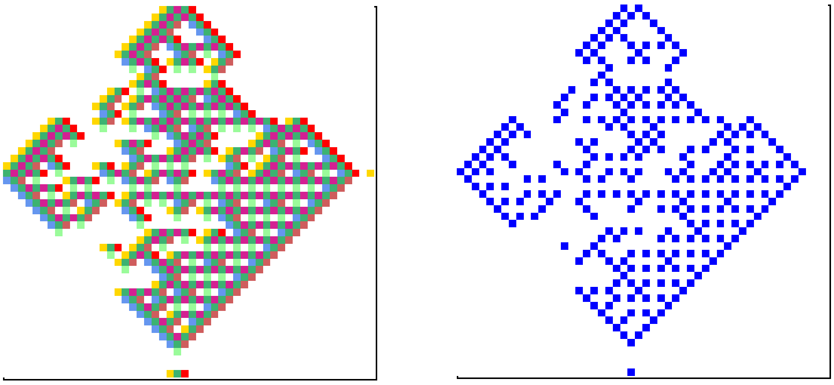


Figure 31.2: 2d space-time pattern colors, the 50th time-step for a CA n=50×50, $k=5$, rule 1a 1b c1 26.
left: by neighborhood, the default.
right: by cell value.

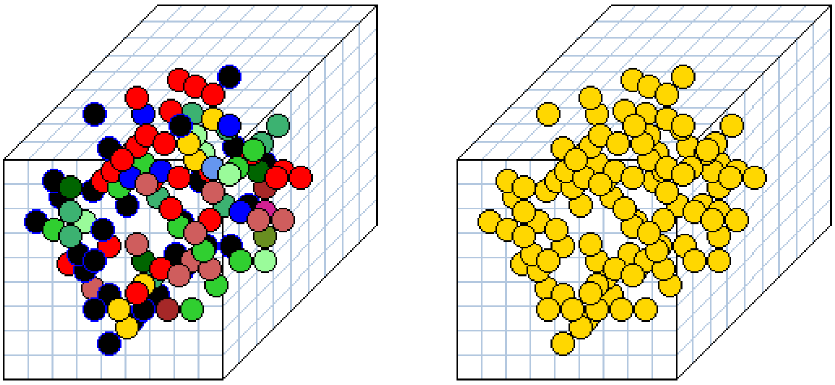


Figure 31.3: 3d space-time pattern colors. The 5th time-step for a CA n=9×9×9, $k=7$.
left: by neighborhood, the default.
right: by cell value.
In both cases cells only cells with value 1 are displayed.

31.5 Pause and step

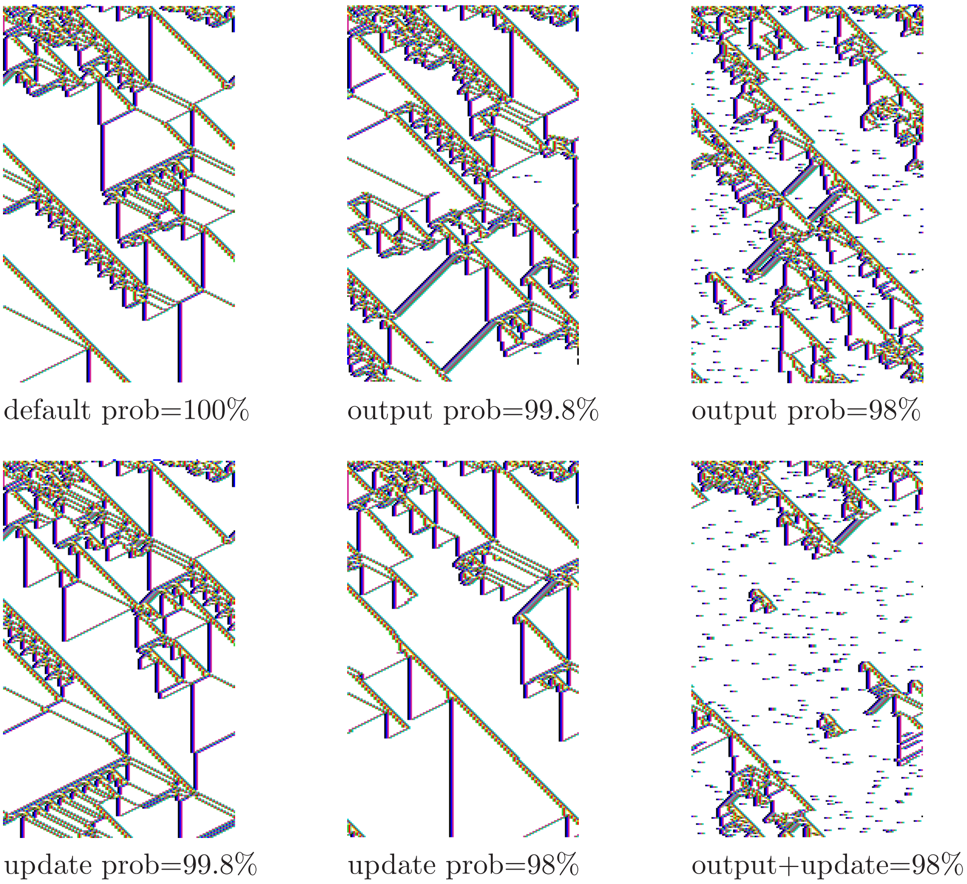
DDLab offers various methods of scanning space-time patterns, some of which are available on-the-fly (see section 32.2), and some from the interrupt window (see section 32.14). From the output parameters, options are offered to make single time-steps, to pause after a given number of time-steps, or to always pause when 1d (or 2d isometric) space-time patterns reach the foot of the screen. No pause is the default. The following top-right prompt is presented,

step-s, pause screen full-p, or enter time-step (no pause-def):

Enter **s** to show each next iteration by pressing **return**. Enter **p** to pause when the screen is full. Alternatively, enter a number to specify a run of time-steps before a pause. At the pause the pause options window will appear (see section 32.14), enter **x** for the next run of time-steps.

31.6 Space-time patters in other than the native dimension

Space-time pattern can be displayed in a dimension other than the "native" dimension. A 1d network can be displayed in 2d or 3d, a 2d in 1d or 3d, and 3d in 1d or 2d. This can also be changed on-the-fly in section 32.9.1. The following top-right prompt is presented,



default prob=100%

output prob=99.8%

output prob=98%

update prob=99.8%

update prob=98%

output+update=98%

Figure 31.4: Probabilistic networks. The "update" probability that a cell will be updated at all, and the "output" probability updating will be according to the rule-table (otherwise at random), see "probabilistic networks" (section 31.7). These examples are for the k=5 CA rule e9 f6 a8 15, n=150, 245 time-steps from the same initial state. Cells colors are by neighborhood (see section 31.4) and filtered to emphasize gliders (see section 32.10).

for 1d
1d network, show STP in 1d-1 2d-2 3d-3 (def 1d):
for 2d
2d network, show STP in 1d-1 2d-2 3d-3 (def 2d):
for 3d
3d network, show STP in 1d-1 2d-2 3d-3 (def 3d):

If not the native dimension, the 2d and 3d axes i, j, h are set automatically by factorizing n. A native 3d network will be displayed in 2d as a stack of horizontal 2d slices as in figure 32.9.1.

Note that if n is prime, i will equal n, and both j and h will equal 1. Then if 2d or 3d is selected, a single row of cells will be displayed across the top of the screen in the 2d or 3d format.

31.7 Probabilistic networks

There are two types of probabilistic networks where the probability, by default 100%, can be reset to a lower value.

Firstly the "update" probability, that a cell will be updated at all, gives some asynchronicity in the dynamics. If this probability is set to P, there is a $1 - P$ chance that a cell is not updated, thus remaining the same at the next time-step.

Secondly the "output" probability, that a cell is updated according to the rule-table, which introduces noise. If the probability is set to P, there is a $1 - P$ chance that a cell's value will be randomly assigned.

Figure 31.4 gives examples. The following top-right prompt is presented,

set probability (def 100%) 0-100, update: output:

First set the the required update probability, then the output probability. These are set as a percentage, which may be a decimal number.

If *both* probabilities are set to less than 100%, then the "update" probability will be implemented first, and only cells that were updated normally will be subject to "output" probability.

31.8 Scrolling 1d space-time patterns

For 1d networks displayed in 1d (see section 31.6 above), the following prompt allows space-time patterns to be scrolled upwards rather than displayed in successive vertical sweeps.

scroll space-time pattern-s:

Enter **s** to scroll. The two methods can also be toggled on-the-fly (see section 32.12.3). Note that if the input-entropy or pattern density is set in section 31.10, these plots will also be scrolled.

31.9 The "analysis" category of options

Options from section 31.10 to section 31.14 belong to the *analysis* category, somewhat arbitrary named, which describes various methods to analyze space-time patterns.

The methods are provided in this category include,

- The pattern density.
- Input-frequency and input-entropy.
- The state-space matrix.
- The return map.

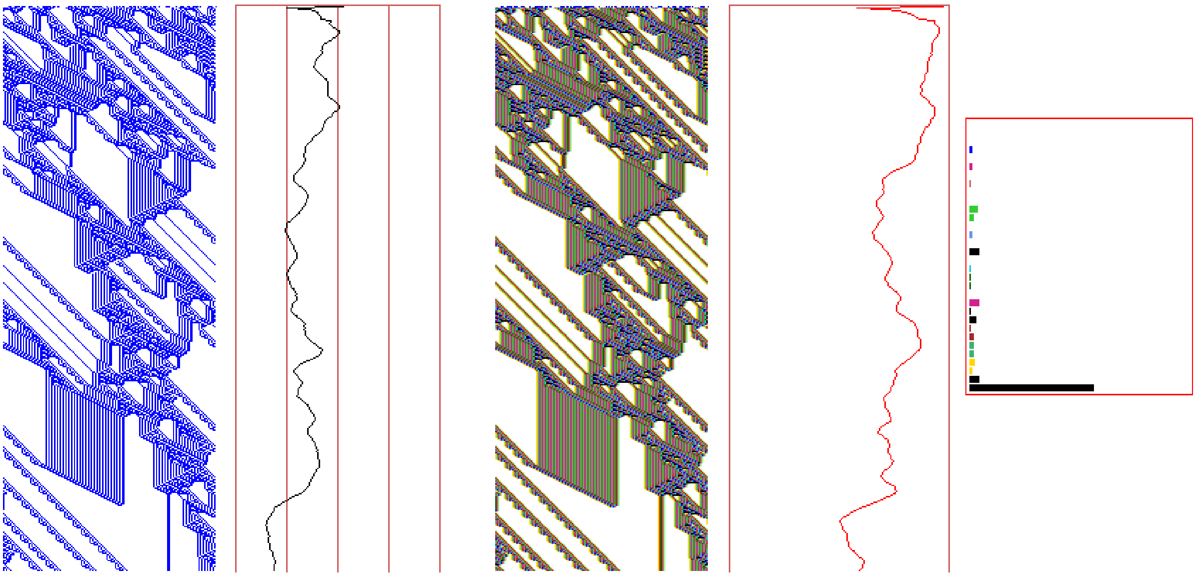
Enter **a** in section 31.1 to skip directly to the first "analysis" option, section 31.10, or arrive here by looking at each output parameter in turn.

Further methods are in the *histograms* category of prompts (section 31.15). Enter **h** in section 31.1 to skip directly to the first histogram option.

31.10 Input-entropy and pattern density

The first two methods provided in the "analysis" category are,

- The pattern density (the density of 1s at each time-step).
- The look-up frequency of neighborhoods in the rule-table (input-frequency) and the resulting "input-entropy".



Density plot: A 1d space-time pattern (color by value), with a density plot alongside. The x axis shows the density of 1s. Vertical divisions (from the left) mark densities of 0, 0.25, 0.5, 0.75 and 1. Time is on the y axis.

Input-Entropy plot: The same 1d space-time pattern (color by neighborhood). To the extreme right is the input frequency histogram, with the neighborhood all 1's top, and all 0's bottom. The input-entropy, the entropy of this histogram is shown center, on the x axis, with zero entropy on the left and maximum entropy on the right. Time is on the y axis.

Figure 31.5: The pattern density plot (left), and input-entropy plot and input frequency histogram (right), for the k=5 1d CA 6e ee 93 60, n-150, 40 time steps from the same random initial state, measured over a moving window of 10 time-steps.

These measures are displayed as a graph alongside the space-time patterns. The input-frequency is simultaneously displayed as a histogram, which may be expanded into a 3d histogram, with an extra time axis (see figures 31.5 and 31.6 and the on-the-fly options in section 32.11.3). A further on-the-fly toggle option (section 32.11.2) shows superimposed graphs of the input frequency for each neighborhood, with or without the input-entropy plot.

The following top-right prompt is presented,

for 1d networks with homogeneous k
ANALYSIS: none-n, pattern density-1,
input-frequency/entropy: histogram-(def), +time-3:

for 2d and 3d networks, or for mixed k
ANALYSIS: none-(def), pattern density-1,
input-frequency/entropy: histogram-2, +time-3:

The default for 1d networks with homogeneous k is to display a plot of the input-entropy, and the input-frequency as a histogram. For 2d or 3d networks, or networks with mixed k, the default is *not* to display either analysis,

Its possible also to restrict these methods to selected single cells rather than the entire network. For input entropy in mixed k networks this is the only option.

If input-entropy and pattern density is set, the display can be toggled on-the-fly between the two (see section 32.11.1), and a scatter plot of the two measures can be displayed where rules have characteristic signatures (see figure 32.15 and section 32.11.4).

Further on-the-fly options plot the mean entropy against the standard deviation of the entropy for a sample of rules, giving a 3d histogram that separates out order, complexity and chaos (see chapter 33).

31.10.1 Pattern density

Enter **1** in section 31.10 above at to show the pattern density. The horizontal scale represents the density of 1s at each time-step, or for the last x time-steps. The size of this moving window of time-steps can be reset (see section 31.12 below). The vertical time axis corresponds to each time-step as drawn.

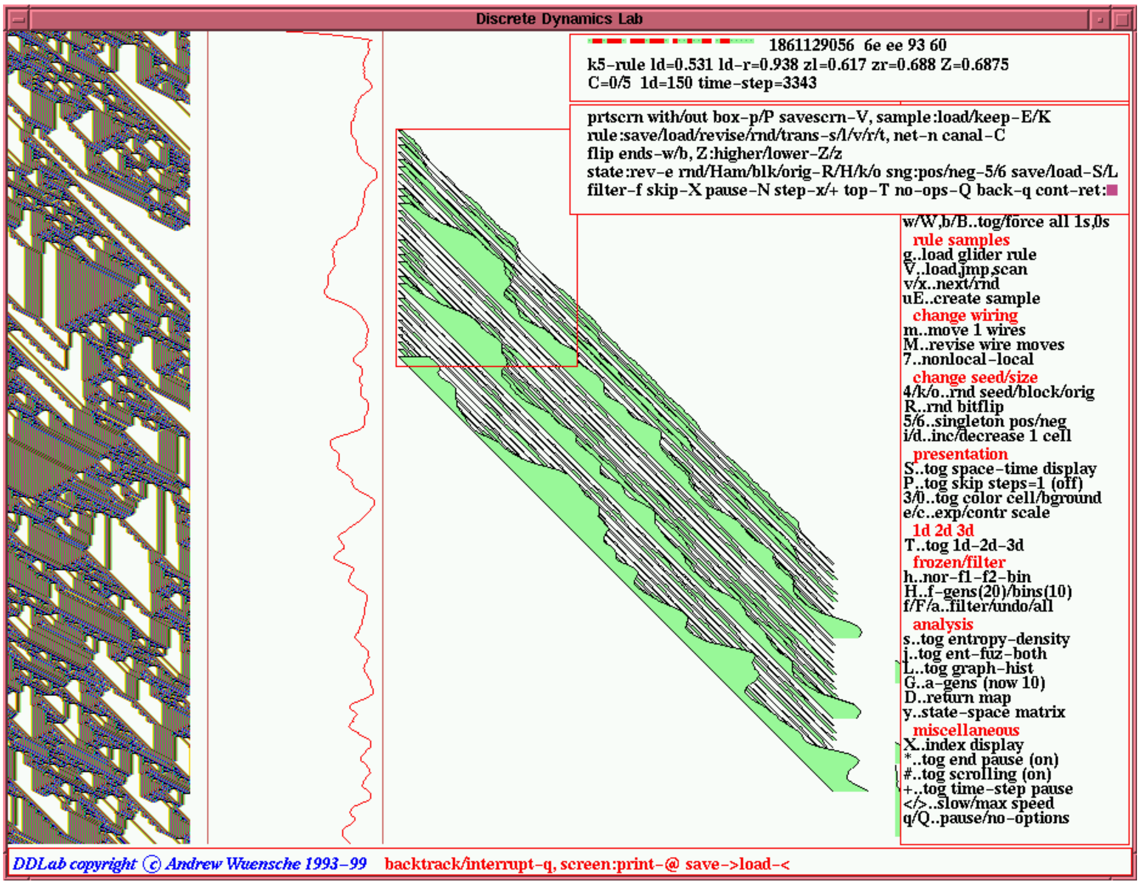
31.10.2 Input-frequency and input-entropy

Enter **return** (or **2** for 2d or 3d networks) in section 31.10 to show a 2d histogram of the input-frequency (on its side), with 2^k columns representing the lookup frequency of each rule-table entry, and colored accordingly. If **3** is entered in section 31.10, the histogram will be expanded to 3d, with an extra time axis, shown as an isometric projection, as in figure 31.6. An on-the-fly option allows toggling between the 2d and 3d histogram (see section 32.11.3).

The entropy of the histogram frequency distribution, the input-entropy, will be displayed to the right of the space-time pattern. Its possible to toggle on-the-fly between between a display of the input-entropy, superimposed graphs of the input frequency of each neighborhood, and both of these plots shown simultaneously (see figure 31.6 and section section 32.11.4).

31.11 Single cell input-entropy and pattern density

All the analysis options and methods of presentation described in section 31.10 can be applied to just one selected singe cell. This is usually done over a much larger moving window of time-steps, the default in 500.



```
1861129056  6e ee 93 60
k5-rule ld=0.531 ld-r=0.938 zl=0.617 zr=0.688 Z=0.6875
C=0/5  1d=150 time-step=3343

prtscrn with/out box-p/P savescrn-V, sample:load/keep-E/K
rule:save/load/revise/rnd/trans-s/l/v/r/t, net-n canal-C
flip ends-w/b, Z:higher/lower-Z/z
state:rev-e rnd/Ham/blk/orig-R/H/k/o sng:pos/neg-5/6 save/load-S/L
filter-f skip-X pause-N step-x/+ top-T no-ops-Q back-q cont-ret:

w/W,b/B..tog/force all 1s,0s
  rule samples
g..load glider rule
V..load,jmp,scan
v/x..next/rnd
uE..create sample
  change wiring
m..move 1 wires
M..revise wire moves
7..nonlocal-local
  change seed/size
4/k/o..rnd seed/block/orig
R..rnd bitflip
5/6..singleton pos/neg
i/d..inc/decrease 1 cell
  presentation
S..tog space-time display
P..tog skip steps=1 (off)
3/0..tog color cell/bground
e/c..exp/contr scale
  1d 2d 3d
T..tog 1d-2d-3d
  frozen/filter
h..nor-f1-f2-bin
H..f-gens(20)/bins(10)
f/F/a..filter/undo/all
  analysis
s..tog entropy-density
j..tog ent-fuz-both
L..tog graph-hist
G..a-gens (now 10)
D..return map
y..state-space matrix
  miscellaneous
X..index display
*..tog end pause (on)
#..tog scrolling (on)
+..tog time-step pause
</>..slow/max speed
q/Q..pause/no-options

DDLab copyright © Andrew Wuensche 1993-99   backtrack/interrupt-q, screen:print-@ save->load-<
```

Figure 31.6: The input frequency histogram in 3d. The DDLab screen showing: (*left*) The space-time patterns for the k=5 1d CA 6e ee 93 60, n-150 (color by neighborhood). (*center left*) The input-entropy plot, which was subsequently toggled to show a superimposed graphs of the input frequency, or both plots together (see section 32.11.2). (*center*) The input frequency histogram in 3d. This can be preset, or toggled on-the-fly between 2d (as in figure 31.5 and 3d. (*right*) The key to on-the-fly options (see sections 32.1 - 32.2). (*top-right*) The options window that appears when space-time patterns are interrupted.

In a mixed k network, if input-entropy was selected (if **2** or **3** was entered in section 31.10), if a single cell position is not specified, it will be applied automatically, usually at position 0.

In general, the following top-right prompt is presented,

single-s all-(def)

If **s** is selected, the position of the single cell needs to be specified (in 1d, 2d or 3d) with the following top-right prompt (for a mixed k network with input-entropy selected, this will be the first prompt),

for a 1d network n=150
single: enter index (max1149 def 0):

for a 2d network 40× 0

single 2d: 2d index (max 39,39, now 0,0)
enter i: enter j:

for a 3d network 22×22×22

single 3d: 3d index (max 21,21,21, now 0,0,0)
enter i: enter j: enter h:

When the space-time pattern is running, a reminder of the single cell position (and the generation size, see section 31.12 below) will appear in a top-right window, for example in a 3d network,

single entropy: pos i,j,h=11,12,13 gens=500

A similar reminder also appears in the "on-the-fly key index" (see section 32.11.5).

31.12 Analysis generation size

The analysis of space-time patterns set in section 31.10 relates to a moving window of time-steps, or generations. For input-entropy these are past time-steps, for pattern density the present time-step is also included.

The default analysis generation size is 10, unless a single cell was selected in section 31.10, in which case the default size is 500. These values can be reset, new setting becomes the default. The following top-right prompt is presented, for example,

enter new 'analysis' generation size (now 10):

The analysis generation size can also be reset on-the-fly, and is shown in the "on-the-fly key index", together with the single cell position, if selected in section 31.11.

31.13 Frozen generation size

Once running, the space-time pattern presentation can be reset on-the-fly to highlight "frozen" cells in a number of ways. These are cells that have not changed in the previous x time-steps (see section 32.10.1). The number of time-steps, or generations, making up the frozen threshold may be reset from the initial default of 20. The new setting becomes the default. The following prompt is presented,

enter new 'frozen' generation size (now 10):

The frozen generation size can also be reset on-the-fly, and is shown in the "on-the-fly key index" (see section 32.10.2).

31.14 State-space matrix and return map

As in section 24.5 for attractor basins, the state-space matrix plots each state in the space-time pattern on a 2d grid near the bottom of the screen, plotting the left half of the bit string against the right half. The y-axis represents the right $n/2$ bits. If n is odd, the extra bit is included on the left, and the grid is a flat rectangle, otherwise the grid is square.

In the return map, each state (bitstring), length n, $B_0, B_1, B_2, B_3 \ldots B_{n-1}$ is converted into a decimal number r, 0-2, where,

$$B_0 + B_1/2 + B_2/4 + B_3/8 + \cdots + B_{n-1}/2^{n-1}$$

As the network is iterated, this state value at time-step t (x-axis) is plotted against the state value at timestep t+1 (*y-axis*) in the bottom center of the screen. From a classical dynamical systems viewpoint, note the fractal structure of the resulting trajectories and attractors.

The following prompt is presented,

show state-space matrix-m, return map-r:

Enter **m** to show the state-space matrix, **r** to show the return map. Both options can also be toggled on-the-fly (see sections 32.11.6 and 32.11.7).

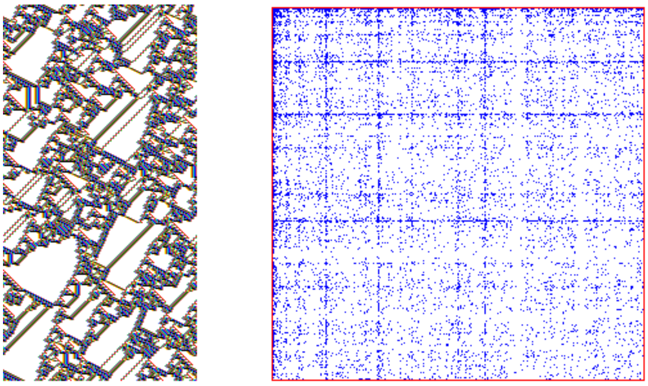


Figure 31.7: The state-space matrix for the $k = 5$ rule 7a 6e 69 84, n=150, plotting the left half against the right half of each state bit string, for about 10,000 time-steps.
left: an example of the space-time pattern, color by cell value, and filtered to show gliders.
right: the state-space matrix.

31.15 The "histograms" category of options

Options from section 31.16 to 31.20.5 belong to the *histogram* category, with further, mainly statistical, methods for analyzing space-time patterns, including,

- The difference between two networks, or "damage".
- Statistical data on attractors types, and "skeletons", suitable for very large networks.

Note that to considerably speed up computation for generating these histograms, the display of space-time pattern graphics can be toggled off on-the-fly with key **S**, see section 32.8.

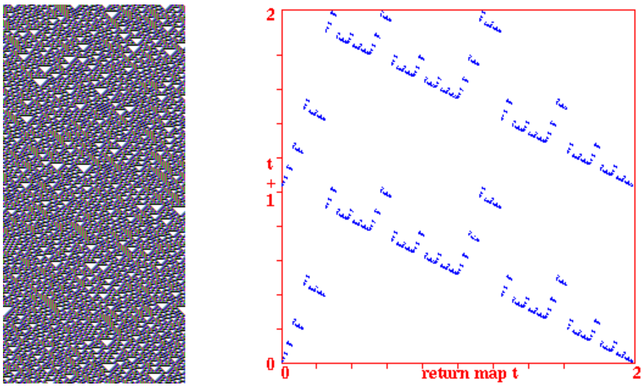


Figure 31.8: The return map for the $k = 3$ rule 30, n=150, plotting the state at each time-step, converted into a decimal number 0-2, against its successor, for about 10,000 time-steps.
left: an example of the space-time pattern, cells colored by value.
right: the return map, note the fractal structure.

Enter **h** in section 31.1 to skip directly to the first "histograms' option, section 31.16 below, or arrive here by looking at each output parameter in turn.

31.16 Damage, the difference between two networks

not applicable to 3d

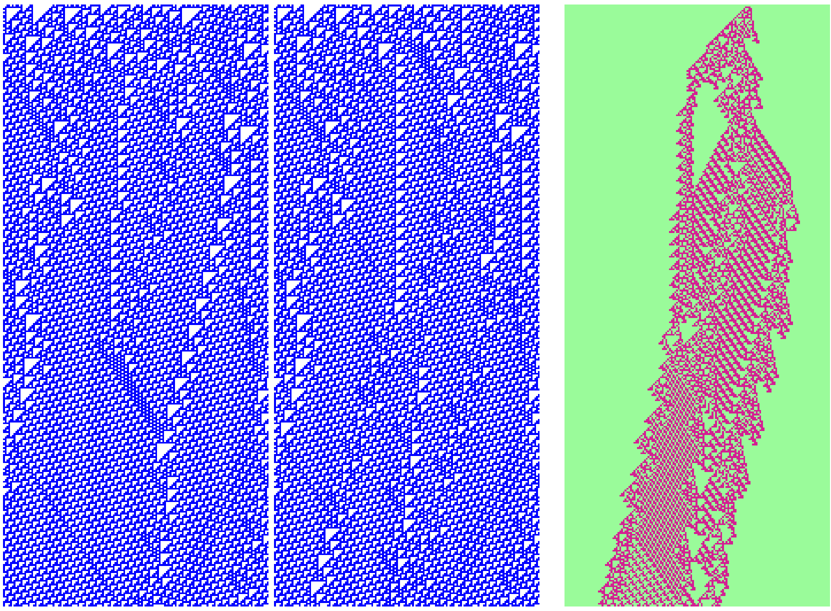


Figure 31.9: The difference in the dynamics between two 1d CA. *left*: The space-time patterns of two identical CA (color by value), except that the random initial state differs by one bit. *right*: The difference between the two at each time-step shown as a third space-time pattern. k=3 rule 110, n=150.

Its possible to graphically represent the difference, or the spread of "damage", between the space-time patterns of two networks. Typically the networks are identical except for a 1 bit difference in their initial state, and the damage is shown as a "purple stain" on a third space-time pattern, spreading from the different bit. This option works for 1d and 2d networks only.

Damaged cells can be defined in two ways. Once damaged, keep the damage irrespective of future dynamics, or show simply the difference at each time-step.

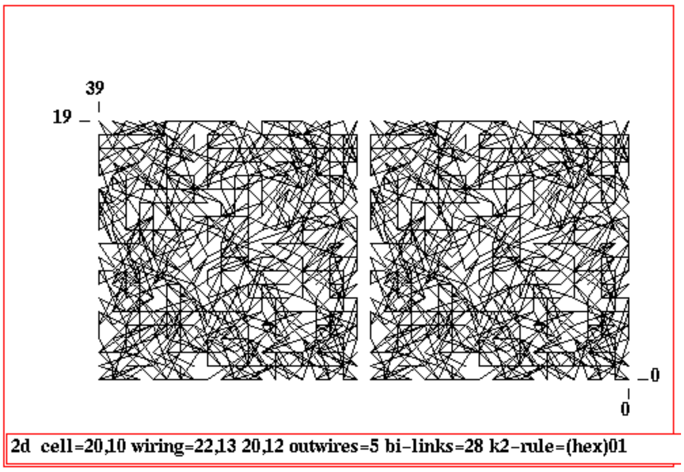
Methods are also provided for automatically gathering statistical data on the sizes of damage spread from many random initial states that differ by one bit, shown as a histogram. This is a method of analyzing order and chaos in networks, especially in random Boolean networks applied as idealized models of genetic regulatory networks[9, 25]. The following top-right prompt is presented,

HISTOGRAMS: show difference between 2 networks
damage: keep-f, dont keep-F, show stats-s:

Enter **s** to shown as a histogram of damage spread for a sample of initial states generated automatically.

Otherwise, for a manual version of damage spread, not showing a histogram, enter **f** to keep the damage, or **F** to show just the difference at each time-step.

31.16.1 Duplicate the network and seed



2d cell=20,10 wiring=22,13 20,12 outwires=5 bi-links=28 k2-rule=(hex)01

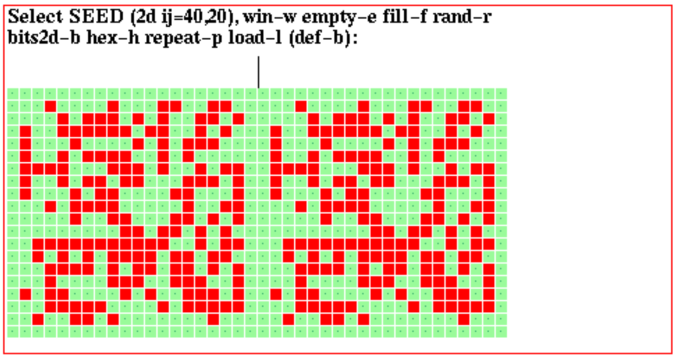
Figure 31.10: A duplicated 2d RBN, 40×20, k=2, showing just all the links. The network consists of two identical 20×20 sub-networks

A subsequent option, normally selected together with the "damage" option (section 31.16) above, duplicates both the network and the initial state. This actually doubles the size of the original network, making a network containing two identical sub-networks, each with the same initial state. The following top-right prompt is presented,

duplicate network and seed-y:

Enter **y** to duplicate the network. Though normally used together, the 'damage" option (section 31.16) and this option can be used independently.

If "damage" was selected and the network is not duplicated, the original network and its initial state will simply be notionaly divided in two, and the "damage" displayed will be between the two halves of the network. However, the network can only be divided if n for 1d, or i for 2d, is even, otherwise the following top-right message is displayed,



Select SEED (2d ij=40,20), win-w empty-e fill-f rand-r
bits2d-b hex-h repeat-p load-l (def-b):

Figure 31.11: A duplicated 2d seed, 40×20. consisting of two identical 20×20 sub-seeds

cant divide odd width network, cont-ret:

Once the network is duplicated, prompts are presented to look at and possibly modify the new network and the seed. Firstly the "network architecture" prompt described in section 17.1, secondly the seed prompt in section 21.1.

When the network is run with these option set, a third space-time pattern shows the difference between the two sub-networks as in figures 31.9 and 31.16.1.

Note that if **f** or **F**, for a manual version of damage spread, was selected in section 31.16, and the network was duplicated in this section, then there will be no difference in the initial states, thus no damage. To introduce a difference, a random bitflip can be made on-the-fly (with **R**, see section 32.7.2). A new random initial state (the same for each sub-network) can also be set on-the-fly (with **4**, see section 32.7.1).

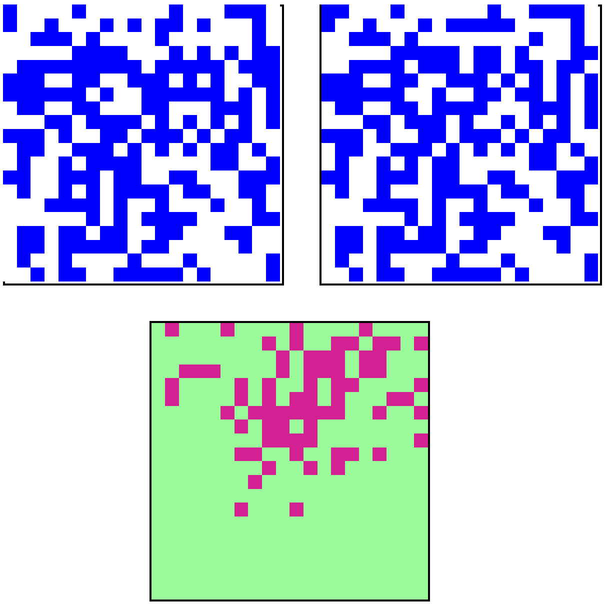
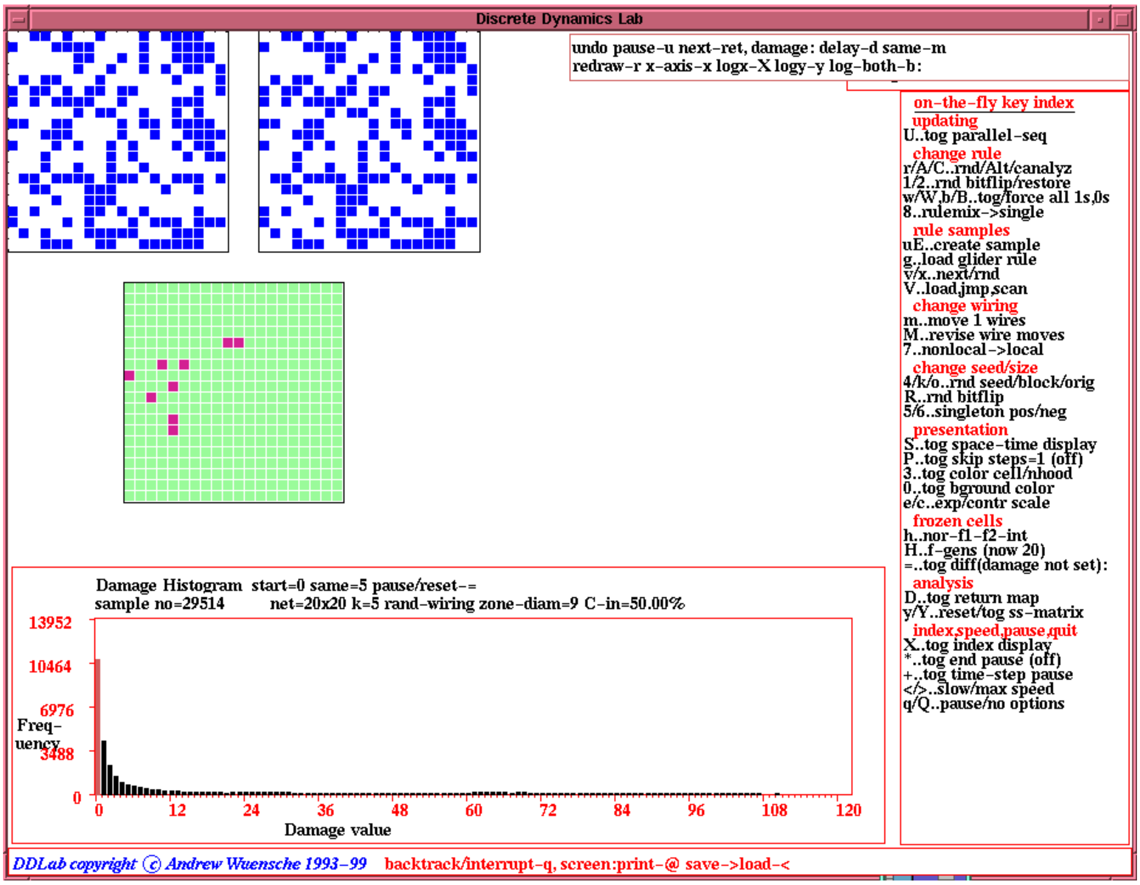


Figure 31.12: An example of 2d "damage". The difference in the dynamics between two 2d RBN. *top*: The space-time patterns of two identical 20×20 RBN (color by value), except that one bit was flipped in the the initial state. The RBN has k=2, with connections as in in figure 31.16.1. *bottom*: The "damage spread" between the two, which has stabilized after a number of time-steps, shown as a third space-time pattern. In this example **keep-f** was selected in section 31.16, so that once a cell is "damaged" - it stays "damaged".



undo pause-u next-ret, damage: delay-d same-m
redraw-r x-axis-x logx-X logy-y log-both-b:

Damage Histogram start=0 same=5 pause/reset-=
sample no=29514 net=20x20 k=5 rand-wiring zone-diam=9 C-in=50.00%

Figure 31.13: The DDLab screen, showing the automatic process for gathering statistics on damage. This example is for two 20×20 RBN sub-networks, k=5, with canalizing inputs set to 50%, see chapter 15). At each pass, a new random initial state is assigned to the sub-networks, differing by one bit at a random position. When the damage stops spreading, the histogram of the damage size (x axis) against the frequency of damage size (y axis), is updated.

31.17 The Damage Histogram

not applicable if $n < 16$ or 3d networks

If **s** (for statistics) is entered in section 31.16, a histogram of damage spread for a sample of initial states will be generated automatically. First the prompt to duplicate the network and seed in section 31.16.1 is presented. Dont duplicate if this was done previously, otherwise the duplicated network will be reduplicated, making 4 copies of the original.

A sequence of top-right prompts are presented to set various parameters,

Damage Spread: show binned damage-b, actual sizes-def:
if **return** *is entered...*

select cut-off damage size (min 20, def 400): *(for sub-networks* 20×20*)*
if **b** *is entered...*
select no of damage bins (min 10 def 100):
followed by...
delay damage, time-steps before damage start (def 0):
define damage, time-steps same damage (def 5):

The "delay damage" is the number of time-steps before the two sub-networks are compared and the measure of the damage starts. This can be changed this from the default 0.

"Define damage" defines at what point the damage is deemed to have stopped spreading. By default, if the damage has not changed for 5 consecutive time-steps, it is considered to be complete - to have stabilized, but this number can be altered.

The "actual" and "binned" options are described in sections 31.17.1 and 31.17.2 below.

31.17.1 Actual damage sizes

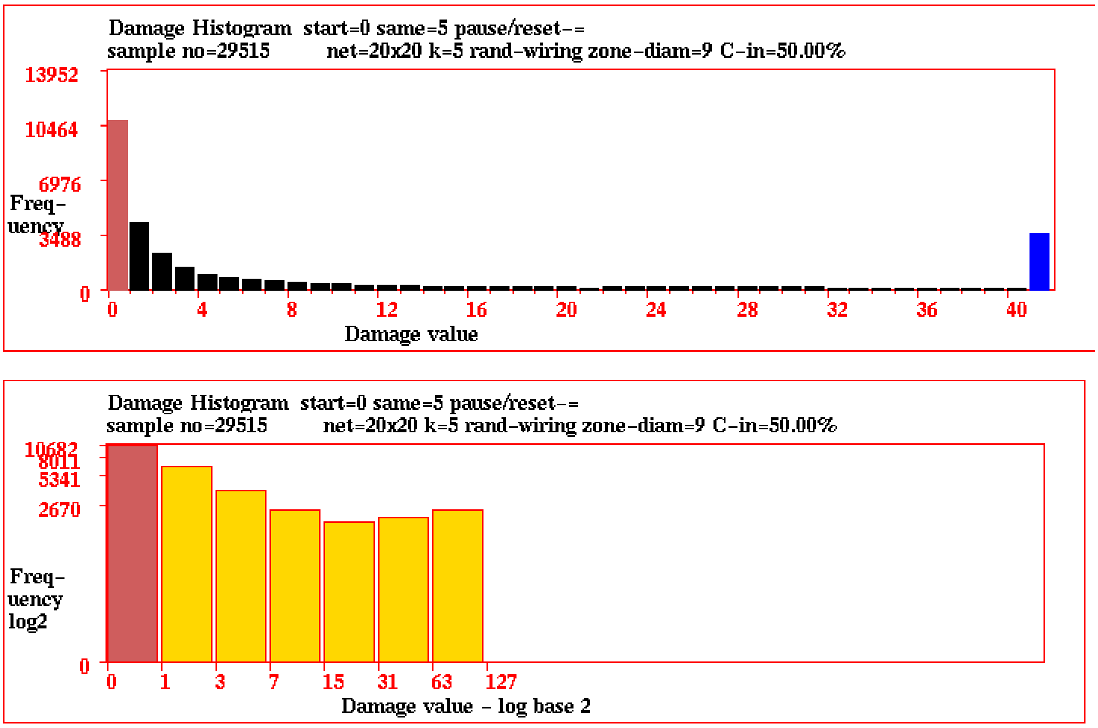


Figure 31.14: The damage histogram showing actual damage, for the same network as in figure 31.13. *top:* With the damage cut-off set at 40 (see above). The frequency of damage of 41 and above is represented by the blue bar on the extreme right. *bottom:* The the damage histogram as a log-log plot.

Enter **return** at the first damage histogram prompt (section 31.17) to present the histogram according to the actual damage sizes. This option will also allow the axes of the histogram to be subsequently rescaled, and shown as a log-log plot. The next prompt sets the cut-off size of the damage, where any damage above the cut-off is lumped together in one excess bin.

select cut-off damage size (min 20, def 400): *(for sub-networks* 20 × 20*)*

When the histogram is being drawn (see section 31.17.3), enter = (the equals sign) to pause, the following top-right prompt is presented to amend parameters and presentation,

undo pause-u save/load-s/l next-ret, damage: delay-d same-m
redraw-r x-axis-x logx-X logy-y log-both-b:

These prompts are explained below,

undo pause-u . . . enter **u** to continue the histogram without pause.

save/load-l/s . . . enter **s** to save the historam data as a `.his` file (see Filing, chapter 35). Enter **l** to load the data and see it in the Xterm window (not for DOS). The file holds the frequencies (0 to the maximum current x axis) in successive pairs of bytes.

next-ret . . . enter **return** to continue the histogram, but pause after the next sample.

delay-d . . . enter **d** to alter the damage delay, described in section 31.17.

same-m . . . enter **m** to define damage, the number of time-steps the damage must stay the same, described in section 31.17.

redraw-r . . . enter **r** to redraw the histogram, which may have been overwritten by other graphics.

x-axis-x . . . enter **x** to re-scale the x-axis, the following top-right prompt is presented,

revise cut-off damage size, now 400 (min 20, def 400):

enter the new cut-off. Any damage above the cut-off is lumped together in one excess bin.

logx-X . . . enter **x** to show the x-axis in log form, with the following top-right reminder,

x-axis bined y powers of 2, cont-ret:

logy-y . . . enter **y** to show the y-axis in log form, with the following top-right reminder,

y axis log2, cont-ret:

log-both-b . . . enter **b** to show both the x and y-axis in log form, with the following top-right reminder,

y axis log2, x-axis bined y powers of 2, cont-ret:

31.17.2 Binned damage sizes

Enter **b** at the first damage histogram prompt (section 31.17) to allocate the damage to a number of equal size "bins", the next prompt sets how many,

select no of damage bins (min 10 def 100):

When the histogram is being drawn (see section 31.17.3), enter = (the equals sign) to pause,

undo pause-u save/load-s/lnext-ret

Only these options are available,

undo pause-u . . . enter **u** to continue the histogram without pause.

save/load-l/s . . . enter **s** to save the historam data as a `.his` file (see Filing, chapter 35). Enter **l** to load the data and see it in the Xterm window (not for DOS). The file holds the frequencies (0 to the maximum current x axis) in successive pairs of bytes.

next-ret . . . enter **return** to continue the histogram, but pause after the next sample.

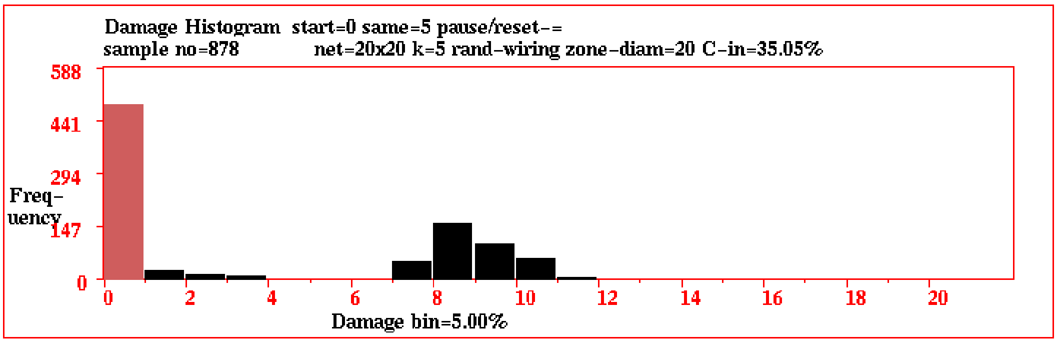


Figure 31.15: The damage histogram showing binned damage. An example of the damage histogram with the damage frequency allocated to 20 of equal size "bins". This example is for two 20×20 RBN sub-networks, k=5, (as in figure 31.13, but with canalizing inputs set to 35%. This results in less ordered dynamics and a different, bimodal, pattern of damage frequency.

31.17.3 Drawing the histogram, and pausing

The histogram is drawn in a window in the lower part of the screen, as in figure 31.13. At each pass, a new random initial state is assigned to the sub-networks, differing by one bit at a random position. When the damage stops spreading (see "define damage" in section 31.17), the histogram of the damage size (x axis) against the frequency of each size (y axis), is updated. The y axis is rescaled automatically to allow the highest histogram bar to fit.

A top-right data window keeps track of the current damage, for example $\boxed{\text{damage=11/400=2.8\%}}$.

Information at the top of the histogram window, including some reminders about the network (see figure 31.14) is decoded as follows,

start= . . . the number of time-steps before the measure of the damage begins, see section 31.17.

same= . . . the number of time-steps required to define damage as having stabilized, see section 31.17.

pause/reset-= . . . reminder: enter = to pause and reset parameters.

sample no= . . . the size of the sample so far.

net=20×20 . . . network size (for example).

k= . . . neighborhood size.

local wiring . . .	local wiring, as in CA
rand-wiring . . .	random wiring, as in RBN
zone diam=. . .	the size of the relative local zone within which random wiring is confined, see section 12.5.2.
C-in= . . .	the percentage of canalizing inputs.

To pause, enter enter = (the equals sign) for the histogram options described in sections 31.17.1 and 31.17.2. Alternatively, pause by entering **q** for the general pause options for space-time patterns described in section 32.13..

The space-time pattern on-the-fly options work during the histogram routine. To speed the damage histogram, toggle the space-time graphics off with the on-the-fly option **S** (see section 32.8.1.

31.18 The Histogram of Attractors or Skeletons

For large networks, especially RBN, it may be not be possible, or practical, to generate the basin of attraction field, (see Network size limitations, section 1.6).

However, statistical data on the range of attractors and their relative sizes can be obtained by running a network forwards from many random initial states, identifying different attractor "types", and creating a histogram of the frequency of each type. The data also includes the period of each attractor type, and the average transient length, the number of time-steps to reach each type. This method is suitable for networks where the attractor can actually be found in a reasonable time.

For large chaotic networks, both the typical transient length and the attractor period may be extremely (astronomically) large, making it impractical to find attractors. However, if a fraction of the network is able to stabilize (the rest remaining chaotic) as in the case of RBN models of genetic regulatory networks, a different type of quasi-attractor can be defined, called a frozen skeleton[24]. The portion of the network that must stabilize, or freeze, not change for a given number of time-steps, needs to be defined. If the stability threshold is reach, the particular pattern of stability is defined as a skeleton type. The network is run forwards from many random initial states, identifying frozen skeleton types, and making a histogram of the frequency of each type. The data also includes the average transient length, the number of time-steps to reach each skeleton type.

The option for the histogram of attractors or skeletons is presented after the damage options (sections 31.16 and 31.16.1). Enter **h** in section 31.1 to skip directly to the first "histograms" option (section 31.16), or arrive there by looking at each output parameter in turn, then enter **return** twice to skip the damage options. The following top-right prompt is presented,

histogram of attractors-a skeletons-s:

31.19 Attractor histogram

Enter **a** at the prompt in section 31.18 above to generate the attractor histogram.

The next top-right prompt allows the inclusion of data on initial states (seeds) that fall into the all 0s or all 1s point attractors. This is intended mainly to check the fitness of rules that have been evolved to solve the "density problem", where initial states with a density of 1s greater than

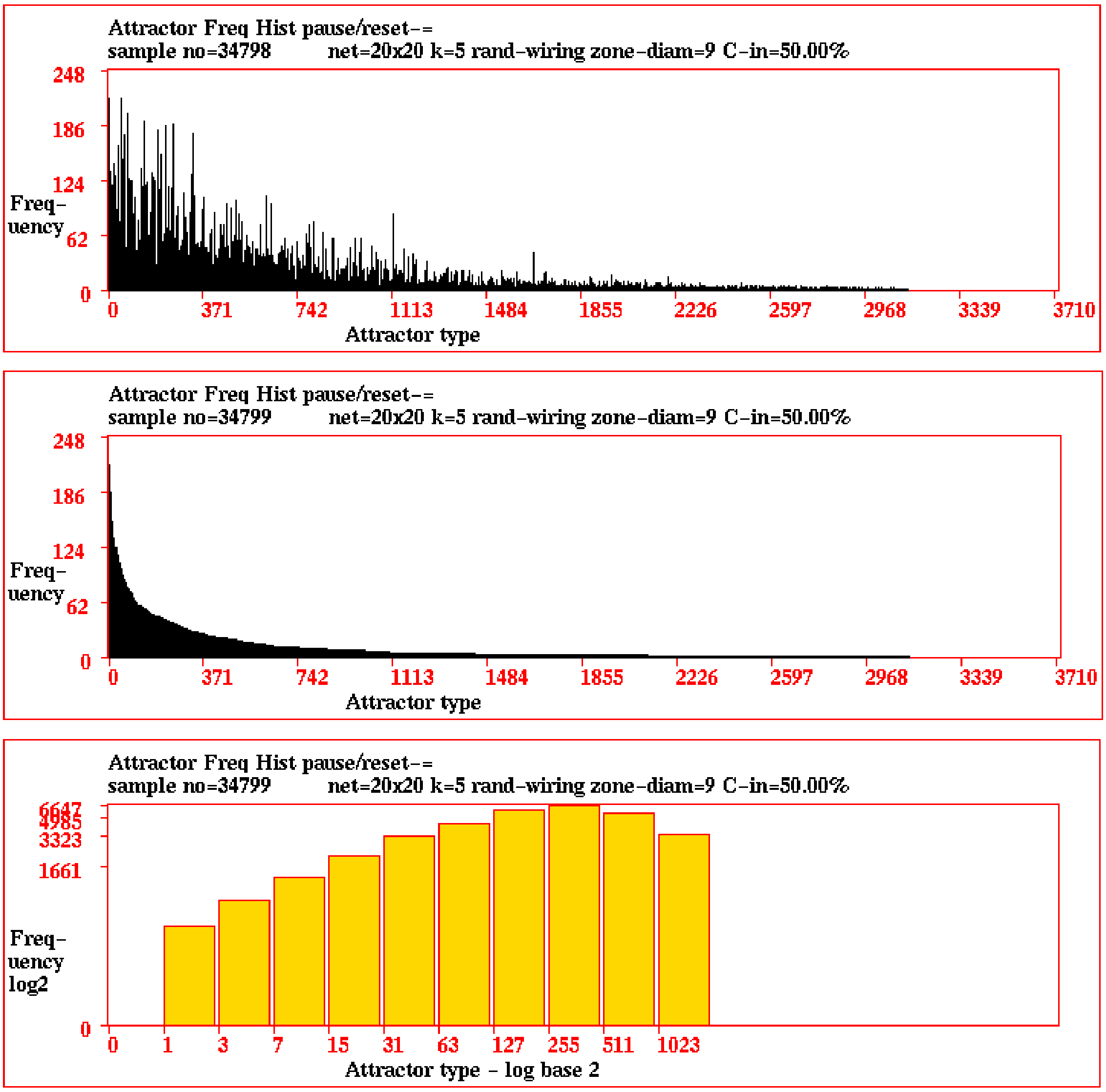
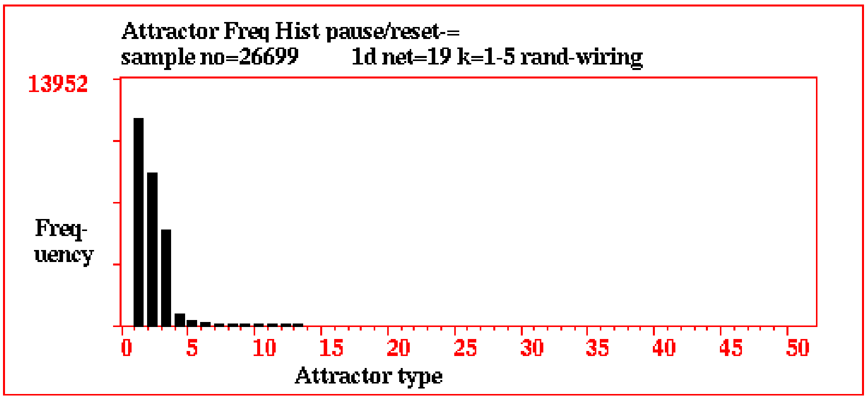


Figure 31.16: The attractor frequency histogram, showing the automatic process for gathering statistics on attractor types. The histogram was paused at sample 34799. This example is for a 20×20 RBN, k=5, with canalizing inputs set to 50%, (see chapter 15). At each pass, a new random initial state is assigned to the network, and the attractor type it falls into is identified. If the type is new it will be added as a new type. The histogram is continuously updated, showing the attractor type (x axis) against the frequency of arriving at that type (y axis), which indicates the relative size of the basin of attraction. *top:* The unsorted histogram. *center:* The sorted histogram, according to frequency. *bottom:* The sorted histogram, shown as a log-log plot.

0.5 are supposed to fall into the all 1s point attractor, and less than 0.5 to the all 0s point attractor,

include seed density data for all=0/1s point attractors-a:

Enter **a** to include this data.



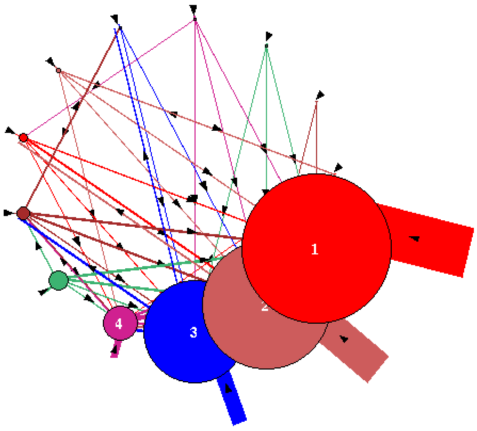


Figure 31.17: *above*: The attractor frequency histogram for a DDN v=3, mixed-k n=19 1d network with random wiring (AtH-stM1.wrs). 13 basins have been found after a sample of about 10000 initial states. *left*: The jump-graph of the attractor histogram (see chapter 20.3), Nodes are scaled to reflect basin volume.

When the histogram is being drawn (see section 31.19.7, A top-right data window gives current data, the types of attractors found, the current type, its period and average transient length, for example,

types=44 this=8 attperiod=12 avtrans=14.0

31.19.1 Pausing the attractor histogram

While the histogram is being drawn, enter = (the equals sign) to pause. The following top-right prompt is presented for the attractor jump-graph (see chapter 20.3), to show data, to sort the order, clear the current histogram and restart the sample, or to redraw the histogram (in case it was covered by other graphics) and to give further options to rescale, save and load.

attractors histogram: jump-graph-j states:print-p/+p save-S/+S data-d sort-s clear-c redraw/options-o cont-ret::

These options are described below.

31.19.2 Attractor histogram jump-graph

The attractor jump-graph, also applied when drawing the complete basin of attraction field (see chapter 20.3), can be applied to much larger networks, or network that are difficult to compute as basin of attraction field, when generated from the attractor histogram.

Enter **j** in section 31.19.1 to show the jump-graph of the attractor histogram.

Although this example took just a few seconds to compute, a jump-graph with many long period attractors would take a longer time. While being computed the following top-right message is shown (enter **q** to abandon),

computing jump-graph, quit-q, or wait...

and the progress is displayed in the xterm window (not for DOS). The basin number is shown progressively as each basin is computed, for example,

```
basins=10 max_period=6 max_trans=53
 1 2 3 4 5 6 7 8 9 10
```

Although this example took just a few seconds to compute, a jump-graph with many long period attractors would take a longer time.

31.19.3 Save/Print the attractor states

A list of the attactor states for each attractor found so far, and other information, can be printed to the terminal, or saved to a (`.dat`) ascii file.. Enter **p** to print to the terminal, enter **S** to save, in section 31.19.1. If saving, further prompts will appear to set the file name etc. (see Filing, chapter 35). Saving itself may involve a brief time interval.

Here is an example of the attactor states ascii data, for the DDN in figure 31.17.3, showing the first 5 attractors of the 13 types that were found from a sample of initial states of about 24000.

```
1d net=19 v=3 k=1-5 rand-wiring att-types=13 sample=24524

att-type 1: period=1 basin-vol=10749=0.438
0000000000000000000

att-type 2: period=4 basin-vol=7852=0.320
1001121101100200022
1001121101100200021
1001121101000200021
1001121101000200022

att-type 3: period=2 basin-vol=4985=0.203
1001101101121210021
1001101101210220000

att-type 4: period=16 basin-vol=605=0.025
0101100100210220002
1202001001111010021
1101100100221220000
1201001001201010011
1101100100102220010
1201001001100000012
0101100100100220010
1202001001000000002
0101100100210220000
1202001001011010021
1101100100221220002
1201001001001010011
1101100100102220012
1201001001200000012
0101100100100220012
1202001001200000002

att-type 5: period=5 basin-vol=172=0.007
2000000001211020020
1022120000111020002
2202000001001020021
1022120000120000011
0202020001110010010
```

Generating just attractor data by this method is very quick, just a few seconds to find most of the the attractors, though the run should be prolonged to catch more of the attractors within very small basins of attraction. The attractors have been reordered by frquency, which indicates basin volume, the proportion of state-space within each basin of attraction.

31.19.4 Attractor histogram data

Enter **d** in section 31.19.1 to show the data. A list of attractor types, as many as will fit, will be displayed in a window on the right of the screen, followed by further options.

The column headings denote the following,

type ... the attractor type index.
no ... the number of this attractor type found.
freq ... the frequency of this type
period ... the attractor period.
avtrans ... the average transient (or run-in) length.

The list order depends on whether the list was sorted in section 31.19.5, if not, the list is ranked in the order that attractor types were found.

This example shows the data for the network in figure 31.16 (center), sorted according to attractor frequency,

```
attractor types so far=3135
type no freq period avtrans
1 217 0.0062 12 29.4
2 216 0.0062 12 29.5
3 200 0.0057 12 28.3
... (the list continues)
39 113 0.0032 12 23.1
40 110 0.0032 4 28.6
41 107 0.0031 12 24.0
quit-q jump-j: (if the complete list fits the window, as in this case)
or
more-ret quit-q jump-j : (if more than one window is required, as in this case)
or
cont-ret jump-j : (if the end of the list is visible)
```

Enter **q** to quit the list and continue the histogram, **j** to jump to a new type index, and **return** to see more of a long list, or to quit the list and continue if the end of the list is visible.

31.19.4.1 Histogram data for density rules

Another example shows the data for a CA, $\boldsymbol{n}$=149, k=7, rule (hex) fa f3 ca fo fa ff 88 e8 fa 73 0a 00 3a 0c 8a 28, one of the "density rules" mentioned above, where the option in section 31.19 was set to include data on initial states that fall into the all 0s or all 1s point attractors. An additional column, headed **den-/50%/+**, shows the percentage of initial states where the density of 1s was less than 50%, exactly 50%, and more than 50%, and the % error in categorizing the seeds (**e=**), for example,

```
attractor types so far=2
type no freq period den-/50%/+ avtrans
1 1121 0.4706 1(all 0s) 985/0/136 e=12% 84.2
2 1261 0.5294 1(all 1s) 212/0/1049 e=16% 77.8
cont-ret jump-j :
```

31.19.5 Sorting the attractor histogram

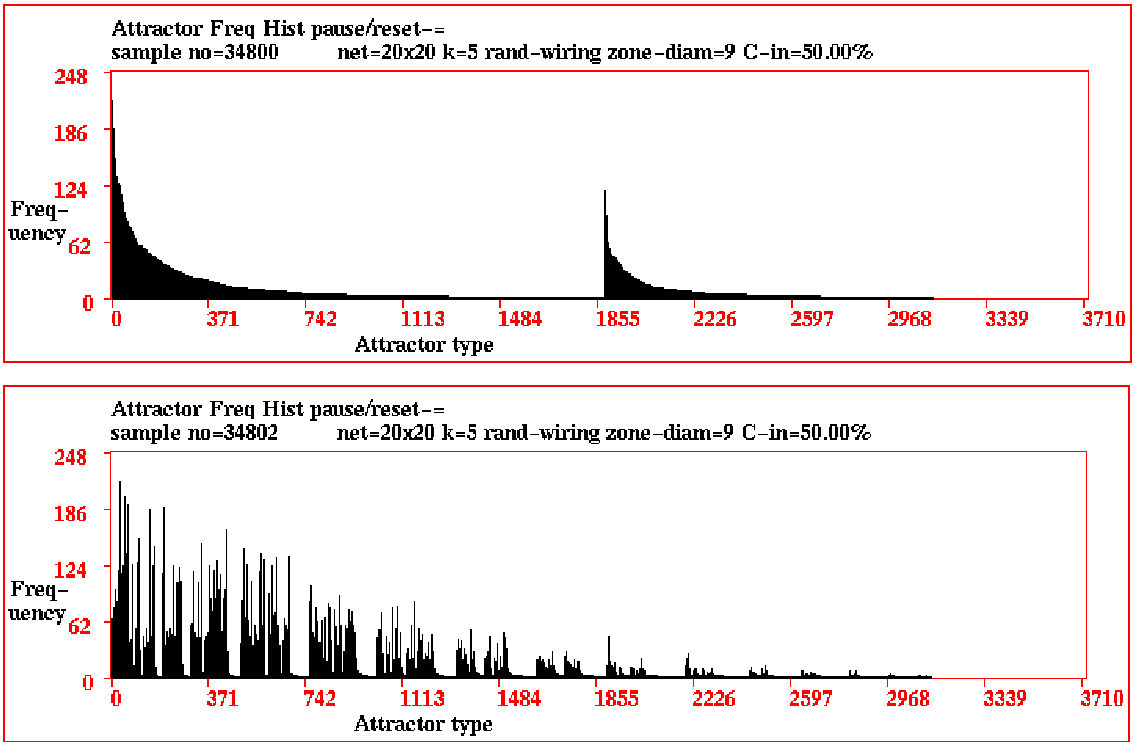


Figure 31.18: The attractor frequency histogram was first sorted according to frequency (figure 31.16, center). It was then by sorted by sorted by period (*top*), then by average transient (*bottom*.

Enter **s** in section 31.19.1 to sort the histogram. The following top-right prompt is presented,

```
sort by: attperiod-a avtrans-t (default freq):
```

Initially the attractor types are ranked in the order they were found, as in figure 31.16 (top). Enter **return** to sort by frequency. An example is shown in figure 31.16 (center). This corresponds to sorting according to the size of the basin of attraction, as the probability of falling into a basin depends on its size.

Enter **a** to sort by attractor period, **t** to sort by average transient (run-in) length. The result of the new sorting depends on how the list is currently sorted. Having first sorted by frequency, the examples in figure 31.18 show the data sorted first by the attractor period, then by the average transient.

31.19.6 Rescaling the attractor histogram

If **s** to sort, or **o** to redraw the histogram and provide further options, is entered in section 31.19.1, a top-right window is presented with options as follows,

> **undo pause-u save/load-l/s next-ret**
> **redraw-r x-axis-x logx-X logy-y log-both-b:**

These options are explained below,

undo pause-u . . . enter **u** to continue the histogram without pause.

save/load-l/s . . . enter **s** to save the historam data as a `.rul` file (see Filing, chapter 35). Enter **l** to load the data and see it in the Xterm window (not for DOS). The file holds frequencies (0-max) in successive pairs of bytes.

next-ret . . . enter **return** to continue the histogram, but pause after the next sample.

redraw-r . . . enter **r** to redraw the histogram, which may have been overwritten by other graphics.

x-axis-x . . . enter **x** to rescale the x-axis, the following top-right prompt is presented,

> **revise cut-off damage size, now 10000 (min 20, def 10000):**

enter the new cut-off. Any damage above the cut-off is lumped together in one excess bin.

logx-X . . . enter **x** to show the x-axis in log form, with the following top-right reminder,

> **x-axis bined y powers of 2, cont-ret:**

logy-y . . . enter **y** to show the y-axis in log form, with the following top-right reminder,

> **y axis log2, cont-ret:**

log-both-b . . . enter **b** to show both the ***x*** and ***y***-axis in log form (as in figure 31.16, bottom), with the following top-right reminder,

> **y axis log2, x-axis bined y powers of 2, cont-ret:**

31.19.7 Attractor histogram window

While the attractor histogram is being drawn, information is displayed at the top of the histogram window, including some reminders about the network (see figure 31.14). This information is decoded as follows,

pause/reset-= . . .	reminder: enter = to pause and reset parameters.
sample no= . . .	the size of the sample so far.
net=20×20 . . .	network size (for example).
k= . . .	neighbourhood size, *k*.
k=3-5. . .	for a range of *k* (for example).
local wiring . . .	local wiring, as in CA
rand-wiring . . .	random wiring, as in RBN

zone diam=... the size of the relative local zone within which random wiring is confined, see section 12.5.2.
C-in= ... the percentage of canalizing inputs.

To pause, enter = (the equals sign) for the histogram options described in sections 31.19.1 to 31.19.6. Alternatively, pause by entering **q** for the general pause options for space-time patterns described in section 32.13.

The space-time pattern on-the-fly options work during the histogram routine. To speed the attractor histogram, toggle the space-time graphics off with the on-the-fly option **S** (see section 32.8.1.

31.20 Skeleton histogram

Enter **s** at the prompt in section 31.18 above to generate the frozen skeleton histogram.

In this context network elements are refered to as "genes" as the method is applied mainly to study models of genetic regulatory networks.

Five parameters need to be set to define a skeleton type, as follows,

1. **1s only** The frozen pattern under consideration, 0s only, 1s only, or both 0s and 1s.
2. **g** The number of time-steps that a gene must remain unchanged to be considered frozen.
3. **s** The number of time-steps the *pattern* of frozen genes must remain unchanged (within a set Hamming distance) to be considered a frozen skeleton type.
4. **sp** The skeleton percentage Hamming distance limits in (3) above.
5. **tp** How close (within a percentage Hamming distance) must a skeleton type be to a pre-established type to qualify as that type.

A series of five top-right prompts are presented to set these parameters,

frozen skeletons: 0s-0 1s-1 both-(def):
no of steps frozen: gene (now 20): skeleton (now 20):
% frozen (def-100): % same type (def-100):

When the histogram is being drawn (see section 31.20.5, a representation of the skeleton is shown in the top center of the screen, noting if the skeleton consists of frozen 0s, 1s, or both.

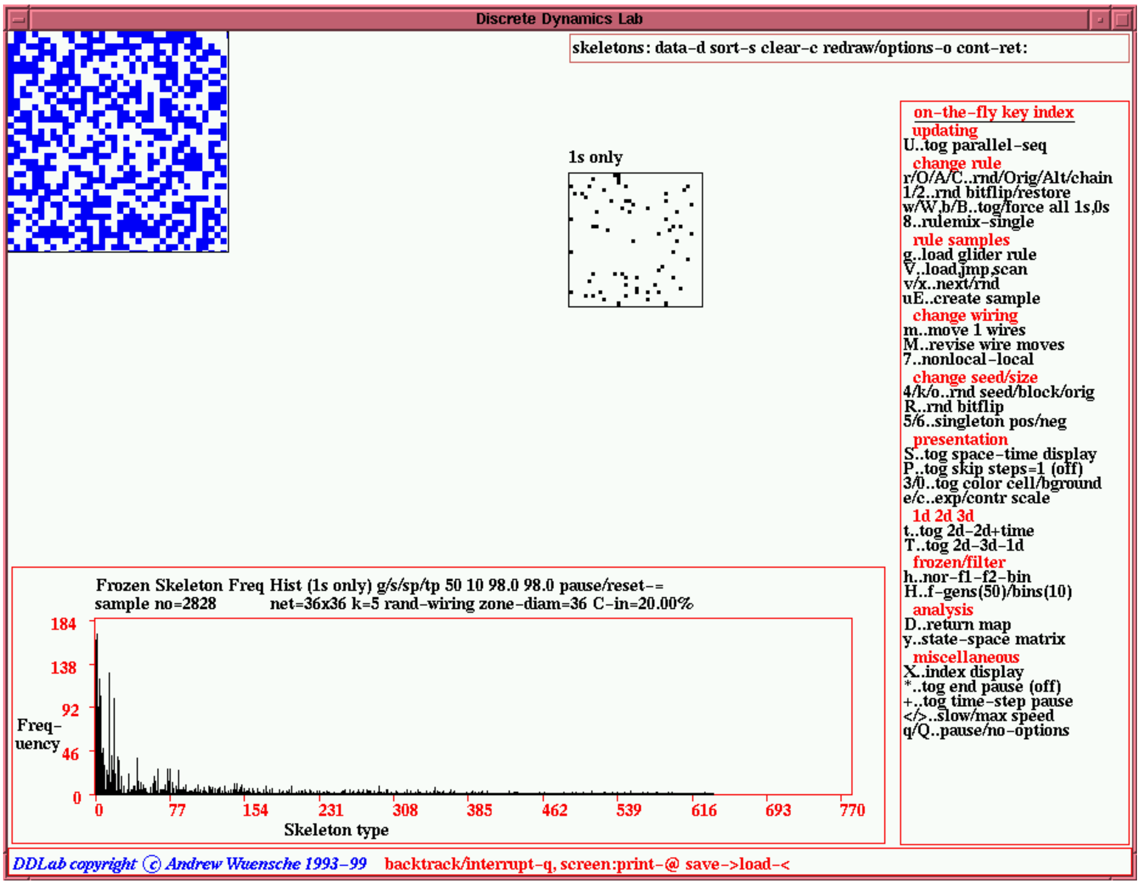
At the same time, a top-right data window gives current data, the types of skeletons found, the current type, and average transient length, for example,

types=639 this=25 avtrans=53.2

31.20.1 Pausing the skeleton histogram

While the histogram is being drawn, enter = (the equals sign) to pause.

The following top-right prompt is presented to show data, to sort the order, clear the current histogram and restart the sample, or to redraw the histogram, in case it was covered by other graphics.



Discrete Dynamics Lab

skeletons: data-d sort-s clear-c redraw/options-o cont-ret:

Frozen Skeleton Freq Hist (1s only) g/s/sp/tp 50 10 98.0 98.0 pause/reset-=
sample no=2828 net=36x36 k=5 rand-wiring zone-diam=36 C-in=20.00%

DDLab copyright © Andrew Wuensche 1993–99 backtrack/interrupt-q, screen:print-@ save->load-<

Figure 31.19: The DDLab screen, showing the automatic proceess for gathering statistics on frozen skeleton types. The histogram was paused at sample 2834. This example is for a 36×36 RBN, k=5, with canalizing inputs set to 20%, see chapter 15). The parameters in section 31.20 (for frozen 1s only), labelled **g/s/sp/tp** are 50, 10 98% 98%. At each pass, a new random initial state is assigned to the network, and the skeleton type it falls into is identified. If the skeleton cannot be matched with a pre-existing skeleton type it is added as a new type. The histogram is continuously updated, showing the skeleton type (x axis) against the frequency of arriving at that type (y axis), which indicates the relative size of the quasi basin of attraction. *top left:* A snapshot of the space-time pattern. *top centre:* The frozen skelaton, pattern of frozen 1s. *bottom:* The unsorted histogram.

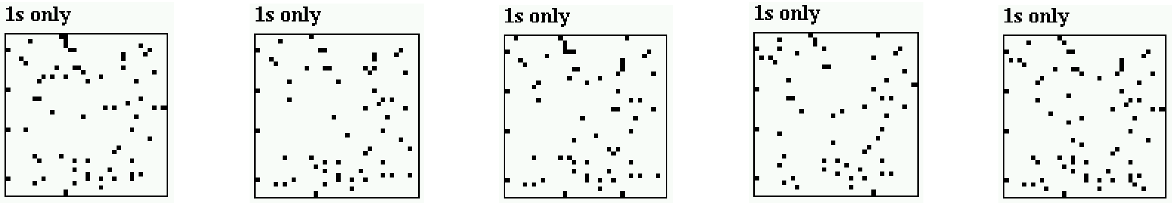


Figure 31.20: Examples of diffeent skeletons, consisting of frozen 1s, from the histogram in figure 31.19. The current skeleton is shown in the top centre of the screen as the histogram is updated.

```
skeletons: data-d sort-s clear-c redraw/options-o cont-ret:
```

Showing data and sorting are described below.

31.20.2 Skeleton histogram data

Enter **d** in section 31.20.1 to show the data. A list of skeleton types, as many as will fit, will be displayed in a window on the right of the screen, followed by further options.

The column headings denote the following,

type . . . the skeleton type index.
no . . . the number of this skeleton type found.
freq . . . the frequency of this type
%fr-0s-1s . . . the percentage of frozen 0s and 1s making up the skeleton.
avtrans . . . the average transient (or run-in) length.

The list order depends on whether the list was sorted in section 31.20.3, if not, the list is ranked in the order that attractor types were found.

This example shows the data for the network in figure 31.16 (center), sorted according to attractor frequency,

```
skeleton types so far=3135
type no freq %fr-0s-1s avtrans
1 165 0.0581 5.3 4.6 53.5
2 171 0.0602 5.5 5.6 53.3
3 94 0.0331 5.4 5.2 53.5
. . . (the list continues)
39 9 0.0032 5.6 5.7 52.8
40 9 0.0032 6.0 5.0 52.6
41 9 0.0032 5.6 5.2 55.1
quit-q jump-j: (if the complete list fits the window, as in this case)
or
more-ret quit-q jump-j : (if more than one window is required, as in this case)
or
cont-ret jump-j : (if the end of the list is visible)
```

Enter **q** to quit the list and continue the histogram, **j** to jump to a new type index, and **return** to see more of a long list, or to quit the list and continue if the end of the list is visible.

31.20.3 Sorting the skeleton histogram

Enter **s** in section 31.20.1 to sort the histogram. The following top-right prompt is presented,

```
sort by: frozen-0-1-b avtrans-t (default freq):
```

Initialy the skeleton types are ranked in the order they were found, as in figure 31.19. Enter **return** to sort by frequency. as in in figure 31.21. This corresponds to sorting according to the fraction of state-space “drained” by the skeleton.

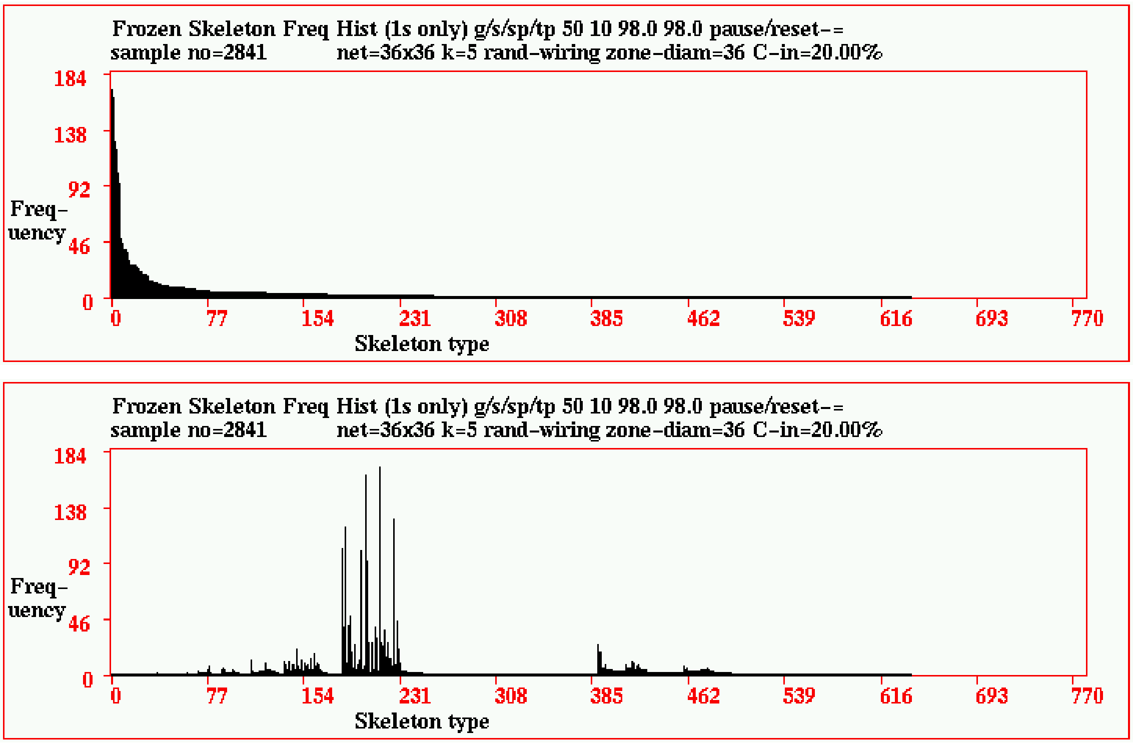


Figure 31.21: The attractor frequency histogram was first sorted acording to frequency (*top*), then by average transient (*bottom*.

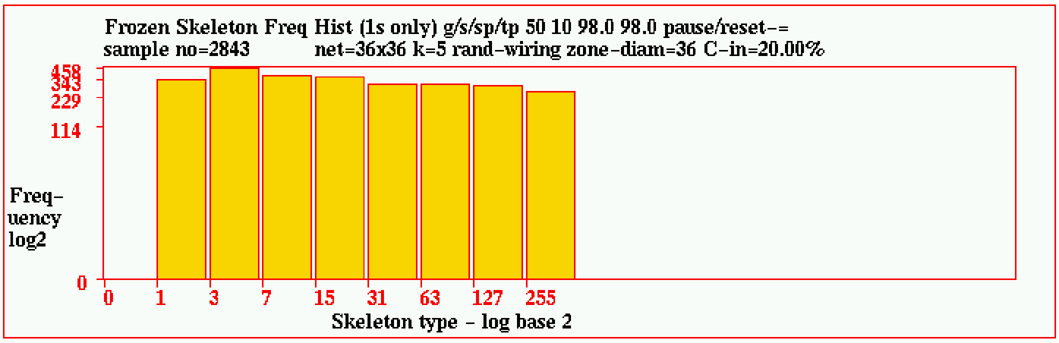


Figure 31.22: The skeleton frequency histogram shown as a log-log plot. The histogram in figure 31.19, was first sorted by frequency (figure 31.21), then rescaled as a log-log plot.

Enter **t** to sort by average transient (run-in) length.

The histogram can also be sorted by number of frozen 0s, 1s or both 0s and 1s, enter **0**, **1** or **b**.

The result of the new sorting depends on the on how the list is currently sorted. Having first sorted by frequency, the examples in figure 31.21 show the data sorted first by the attractor period, then by the average transient.

31.20.4 Rescaling the skeleton histogram

If **s** to sort, or **o** to redraw the histogram and provide further options, is entered in section 31.20.1, a top-right window is presented with options as follows,

undo pause-u save/load-l/s next-ret
redraw-r x-axis-x logx-X logy-y log-both-b:

These options are the same as for the attractor histogram and are explained in section 31.19.6.

31.20.5 Skeleton histogram window

While the skeleton histogram is being drawn, information is displayed at the top of the histogram window, including some reminders about the network (see figure 31.14). This information is decoded as follows,

(1s only) . . .	
(0s only) . . .	
(0s or 1s) . . .	reminder of the basis of the skeleton.
g/s/sp/tp . . .	see below, and section 31.20.
g . . .	gene frozen time-steps.
s . . .	pattern frozen time-steps.
sp . . .	skeletonfrozen time-steps.
s . . .	skeleton % Hamming distance.
tp . . .	skeleton type % Hamming distance.
pause/reset-= . . .	reminder: enter = to pause and reset parameters.
sample no= . . .	the size of the sample so far.
net=36×36 . . .	network size (for example).
k= . . .	neighbourhood size, *k*.
k=3-5. . .	for a range of *k* (for example).
local wiring . . .	local wiring, as in CA
rand-wiring . . .	random wiring, as in RBN
zone diam=. . .	the size of the relative local zone within which random wiring is confined, see section 12.5.2.
C-in= . . .	the percentage of canalizing inputs.

To pause, enter = (the equals sign) for the histogram options described in sections 31.20.1 to 31.20.4. Alternatively, pause by entering **q** for the general pause options for space-time patterns described in section 32.13.

The space-time pattern on-the-fly options work during the histogram routine. To speed the skeleton histogram, toggle the space-time graphics off with the on-the-fly option **S** (see section 32.8.1.

Chapter 32

Drawing space-time patterns, and changes on-the-fly

Space-time patterns are drawn if **s** was selected at the first output parameter prompt (section 24.1), followed by the various options in chapter 31, "Output parameters for space-time patterns" (enter **d** to accept all defaults). Space-time patterns can also be selected at the subtree or single basin prompt (section 29.1 (enter **S**. Note that a "single basin/subtree" would have been selected at the first DDLab prompt in section 6.2 by entering **s**.

This chapter describes parameters and options that can be reset or activated while space-time patterns are being drawn (on the left of the screen), either on-the-fly, or by further prompts if the drawing is interupted.

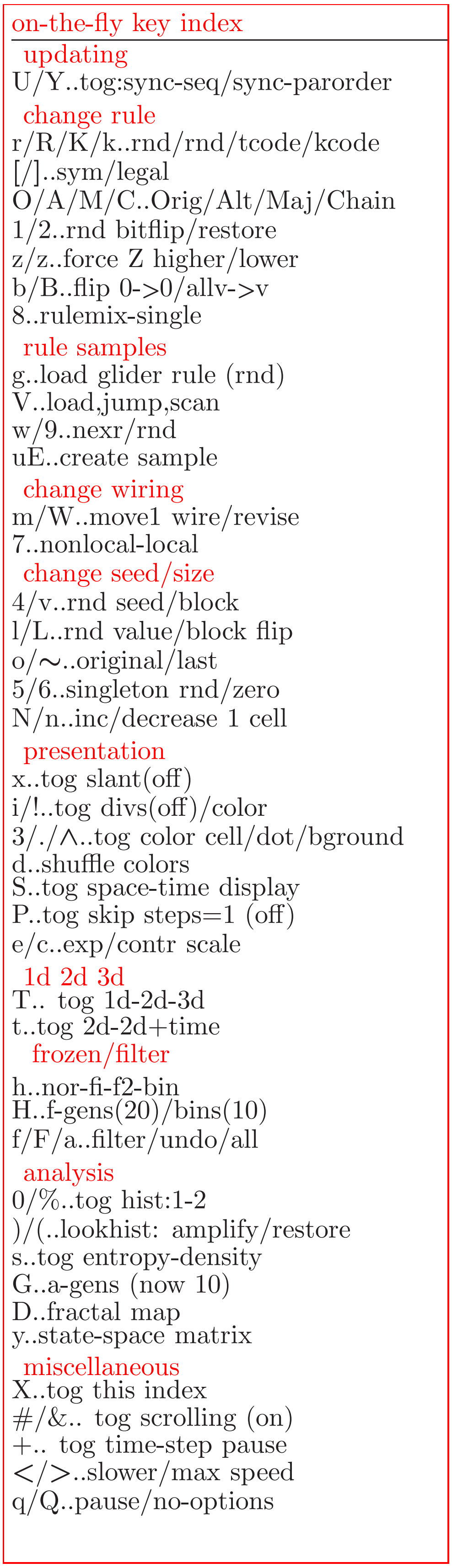
On-the-fly options include: changes to updating, rule/s, wiring, seed and presentation, showing "frozen" regions and filtering, and various methods of analysis. There are also options for loading glider rules, and automaticaly classifying rule-space (described in greater detail in chapter 33).

If the drawing is interupted, extra options are presented in a top-right window. One of these is the network-graph which can be rearanged in the many ways described in chapter 20, after which an additional version of the space-time graphics will be run according to the graph layout and node sizes, with corresponding presentation flexibility.

Further options include filing, canalyzing, and further methods for manipulating the space-time patterns. Some of these parameters were first set in section 31, so can be reset without the need to backtrack. Others are new parameteres and options which can only be activated once space-time patterns have started.

32.1 On-the-fly key index

An "on-the-fly key index" window is usually displayed on the right of the screen while space-time patterns are drawn as a reminder of the various options which may be activated on-the-fly from the keyboard. An example is shown in figure 32.1. The options are divided into categories, with subheadings in red, and may vary for different types of network and current settings. The on-the-fly window can be toggled on-off by entering with **X**. The on-the-fly options are summarized in section 32.2 and explained in greater detail in sections 32.3 - 32.12.6.



on-the-fly key index

updating

U/Y..tog:sync-seq/sync-parorder

change rule

r/R/K/k..rnd/rnd/tcode/kcode
[/]..sym/legal
O/A/M/C..Orig/Alt/Maj/Chain
1/2..rnd bitflip/restore
z/z..force Z higher/lower
b/B..flip 0->0/allv->v
8..rulemix-single

rule samples

g..load glider rule (rnd)
V..load,jump,scan
w/9..nexr/rnd
uE..create sample

change wiring

m/W..move1 wire/revise
7..nonlocal-local

change seed/size

4/v..rnd seed/block
l/L..rnd value/block flip
o/∼..original/last
5/6..singleton rnd/zero
N/n..inc/decrease 1 cell

presentation

x..tog slant(off)
i/!..tog divs(off)/color
3/./∧..tog color cell/dot/bground
d..shuffle colors
S..tog space-time display
P..tog skip steps=1 (off)
e/c..exp/contr scale

1d 2d 3d

T.. tog 1d-2d-3d
t..tog 2d-2d+time

frozen/filter

h..nor-fi-f2-bin
H..f-gens(20)/bins(10)
f/F/a..filter/undo/all

analysis

0/%..tog hist:1-2
)/(..lookhist: amplify/restore
s..tog entropy-density
G..a-gens (now 10)
D..fractal map
y..state-space matrix

miscellaneous

X..tog this index
#/&.. tog scrolling (on)
+.. tog time-step pause
</>..slower/max speed
q/Q..pause/no-options

Figure 32.1: An example of the "on-the-fly key index" for space-time patterns which is usually displayed on the right of the screen while space-time patterns are drawn on the left side, as a reminder of the various on-the-fly options which can be activated from the keyboard.
The options are divided into subcategories, with subheadings in red. They are context dependent, and may vary for different types of network or according to current settings, so some options will be missing or different in an actual run of DDLab.
The window can be toggled on-off with the **X** key.

32.2 Summary of on-the-fly options for space-time patterns

updating	
U..parallel-seq . . .	toggle parallel/sequential updating (section 32.3)
change rule	
r/O/A/C..rnd/Orig/Alt/Chain . . .	new rule/s (section 32.4.1)
r . . .	random rule
O . . .	restore original rule
A . . .	Altenberg rule
C . . .	Chain rule
1/2..rnd bitflip/restore . . .	mutate/restore rule/s (section 32.4.2)
1 . . .	mutate rule-table by 1 random bit
2 . . .	restore mutated bits in reverse order
Z/z..force Z higher/lower . . .	change rule's Z-parameter (section 32.4.3)
Z . . .	force Z higher
z . . .	force Z lower
w/W,b/B..tog/force all 1s,0s . . .	change the "hot bits" (section 32.4.4)
w . . .	flip the output of all 1s
W . . .	force the output of all 1s to 1
b . . .	flip the output of all 0s
B . . .	force the output of all 0s to 0
8..rulemix-single . . .	toggle rule-mix/single rule at cell 0 (section 32.4.5)
rule samples	
g..load glider rule . . .	load a glider rule (section 32.5.1)
V..load,jmp,scan . . .	load rule sample (section 32.5.2 and chapter 33)
v/x..next/rnd . . .	scan successive rules
x . . .	jump to a random sample index
uE..create sample . . .	create sample of classified rule-space (section 32.5.4 and chapter 33.
change wiring	
m..move 1 wires . . .	randomly move one or more wires (section 32.6)
M....revise wire moves . . .	change number of wires to move (section 32.6.2)
7..nonlocal-local . . .	toggle non-local/local wiring (section 32.6.3)
change seed/size	
4/k/o..rnd seed/block/orig . . .	change state/seed (section 32.7.1)
4 . . .	random state
k . . .	random block

o . . . restore original state

R..rnd bitflip . . . mutate state by 1 random bit (section 32.7.2)
5/6..singleton pos/neg . . . mutate state by 1 random bit (section 32.7.3)

5 . . . set positive singleton seed
6 . . . set negative singleton seed

i/d..inc/decrease 1 cell . . . change network size (section 32.7.4)

i . . . increase network size by 1 cell
d . . . decrease network size by 1 cell

presentation

S..tog space-time display . . . toggle the display of space-time graphics on/off (section 32.8.1)
P..tog skip steps=1 (off) . . . toggle skipping time-steps (section 32.8.2)
3/0..tog color cell/bground . . . toggle skipping time-steps (section 32.8.3)

3 . . . toggle cell colors neighborhood/value
0 . . . toggle background color white/light green

$..tog sound . . . toggle sound *DOS only* (section 32.8.4)
e/c..expand/contr scale . . . expand/contract cell scale (section 32.8.5)

e . . . expand cell scale by 1 pixel
c . . . contract cell scale by 1 pixel

1d 2d 3d change dimension (section 32.9)

T..tog 1d-2d-3d . . . toggle 1d/2d/3d (section 32.9.1)
t..tog 2d-2d+time . . . toggle 2d/2d+time (section 32.9.2)
p..2d draw plane . . . draw 2d plane in 2d+time (section 32.9.3)
I..tog 3d slice . . . toggle 3d in successive slices (section 32.9.4)

frozen/filter (section 32.10)

h..nor-f1-f2-bin . . . toggle between different ways of showing “frozen” cells (section 32.10.1)
H..f-gens (now 20)/bins(10) . . . change “frozen” parameters (section 32.10.2)
f/F/a..filter/undo/all . . . (section 32.10.5)

f . . . progressively filter
F . . . progressively unfilter in reverse order
a . . . restore all filtering

analysis

s..tog entropy-density . . . toggle input-entropy/pattern density (section 32.11.1)
j..tog ent-fuz-both . . . toggle between different input-entropy presentions (section 32.11.2)

L..tog hist-graph . . . show the input-frequency histogram with a time dimension (section 32.11.3)

u..tog entropy/density graph . . . show the density/entropy scatter plot (section 32.11.4)

G..a-gens (now 10) . . . change analysis generations (section 32.11.5)

D..return map . . . show/delete return map (section 32.11.6)

y..state-space matrix . . . show/delete state-space matrix (section 32.11.7)

miscellaneous

X..index display . . . show/delete the on-the-fly key index (section 32.12.1)

***..tog end pause (on)** . . . toggle the end pause on/off (section 32.12.2)

#/&..tog scrolling . . . toggle the scrolling on/off with either # or & (section 32.12.3)

+..time-step pause . . . pause after each time-step (section 32.12.4)

</>..slow/max speed: . . . speed (section 32.12.5)

< . . . slow down the space-time patterns

> . . . restore full speed

q/Q..pause/no-options . . . (section 32.12.6)

q . . . pause space-time patterns and display data/options

Q . . . turn off data/options when pausing

32.3 updating

U..parallel-seq: enter **U** to toggle on-the-fly between parallel and sequential updating (see also section 31.3). The current status is given by the prompt order, i.e. **U..seq-parallel** indicates that sequential updating is current.

32.4 change rules

These on-the-fly options amend (mutate) the rule, or the rules in a mixed rule network.

32.4.1 rnd/Orig/Alt/Chain

r/O/A/C..rnd/Orig/Alt/Chain: enter **r**, **O**, **A** or **C** to amend rule/s as described below,

r . . . enter **r** to assign a new random rule. For mixed rule networks, random rules are assigned for all cells in the network. For a single rule network, the Z-parameter and the rule (for $\boldsymbol{k} \leq \mathbf{5}$) is shown in a top right window, for example,

Z=0.671875 hex=bc a9 15 4b

O . . . enter **O** to restore the rule that was originaly selected. For mixed rule networks the rule is assigned to cell index 0 only.

A . . . enter **A** set an an "Altenberg" rule, chosen at random (see section 16.9). For mixed rule networks, Altenberg rules are assigned for all cells in the network. For a single rule network, the Z-parameter and the rule (for $\boldsymbol{k} \leq \mathbf{5}$) is shown in a top right window, for example,

Z=0.507812 hex=eb 48 e8 c8

C . . . enter **C** set a maximally chaotic rule chain-rule, chosen at random (see section 16.11). For mixed rule networks, chain-rules are assigned for all cells in the network. For a single rule network, the $\boldsymbol{Z_{left}}$ and $\boldsymbol{Z_{right}}$ parameters, and the rule (for $\boldsymbol{k} \leq \mathbf{5}$), are shown in a top-right window, for example,

zl=0.875 zr=1 hex=97 96 68 69

32.4.2 rnd bitflip/restore

1/2..rnd bitflip/restore: enter **1** or **2** to mutate or restore the rule/s, as described below,

1 . . . enter **1** to mutate the rule by 1 bit, a random bitflip in the rule-table, or a different random bitflip for each network element in a rule-mix. For single rule networks only, data will appear in a top-right window, for example,

bitflip=at pos 30, flip=2 Z=0.671875 hex=4e 68 63 46

This shows the cell index of the bitflip, the number of flips so far, the new Z parameter, and the new rule in decimal (for $\boldsymbol{k} \leq \mathbf{3}$) and hex (for $\boldsymbol{k} \leq \mathbf{5}$).

2 . . . for single rule networks only, enter **2** to restore the random bit-flips above, or the sequence of bitflips in reverse order. Data will appear in a top-right window, for example,

bit back =at pos 24, flip=0 Z=0.671875 hex=0f 68 63 46

This shows the cell index of the bitflip, the flip backs in reverse order (0 means the original rule has been restored), Z parameter, and the rule in decimal (for $\boldsymbol{k} \leq \mathbf{3}$) and in hex (for $\boldsymbol{k} \leq \mathbf{5}$).

32.4.3 force Z higher/lower

for single rule networks only

Z/z..force Z higher/lower: enter **Z** (upper case) to progressively force the Z-parameter higher, or enter **z** (lower case) to force it lower, as in section 16.3. For a single rule network, the Z-parameter and the rule (for $\boldsymbol{k} \leq \mathbf{5}$) is shown in a top right window, for example,

Z=0.53125 hex=77 6f f1 8f

Another window below shows more information: canalizing, $\boldsymbol{Z_{left}}$, $\boldsymbol{Z_{right}}$, $\boldsymbol{\lambda}$-parameter, $\boldsymbol{\lambda_{ratio}}$, $\boldsymbol{P}$-parameter, and Z-parameter, for example,

C=0/5 zl=0.375 zr=0.53125
ld=0.6875 ld-r=0.625 P=0.6875 Z=0.53125

32.4.4 tog/force all 1s, 0s

w/W,b/B..tog/force all 1s,0s: these options affect the "hot bits" at the extreme ends of the rule-table, for the neighbourhoods all 1s and all 0s.

w/W . . . enter **w** to flip the output of the all 1s neighborhood, or enter **W** to force the all 1s neighborhood to output a 1. This applies for a single rule, or for all rules in a mixed rule network.

b/B . . . enter **b** to flip the output of the all 0s neighborhood, or enter **B** to force the all 0s neighborhood to output a 0. This applies for a single rule, or for all rules in a mixed rule network.

32.4.5 rulemix-single

for a mixed rule network only

8..rulemix-single: enter **8** to toggle between running the network according to the its rule-mix, or according to the single rule at cell index 0. The current status is given by the prompt order, i.e. **8..single-rulemix** indicates that the single rule is current.

32.5 rule samples

These on-the-fly options relate to loading "glider rules", and to automatically classifying rule-space[27], including creating, sorting, analysing and probing the sample. The latter is a larger topic described in detail in chapter 33.

32.5.1 load glider rule

g..load glider rule: enter **g** to load a "glider" rule from a file. A collection of "complex" rules that produce coherent interacting structures (known as gliders) are provided in separate files with DDLab. The samples currently available are for k=5, 6 and 7 rules only, for 1d networks, though the command will also work for 2d and 3d. These are held in the files `glider5.r_s` (62 rules), `glider6.r_s` (31 rules) and `glider7.r_s` (31 rules). The glider rule will become the new rule in a single rule network. For a mixed rule or mixed k network, provided that k for cell index 0 is 5, 6 or 7, **g** will load a glider rule at cell index 0. This rule can be applied to the whole network by toggling **8..rulemix-single** in section 32.4.5 above. The new glider rule will be shown in a top-right window. For example (for k=6)

1d glider rule=2 hex=b3 87 b6 b8 8c d4 af a0

rule=2 indicates that the rule is number 2 in the current sample.

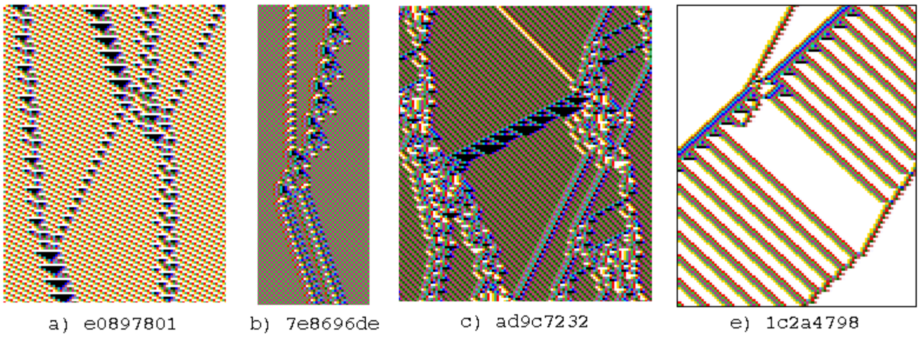


Figure 32.2: Examples of k=5 complex space-time patterns with interacting gliders from the sample of glider rules included with DDLab, which can be loaded on-th-fly while space-time patterns are being drawn. The $\boldsymbol{k}$=5 rule numbers are shown in hex.

32.5.2 load,jmp,scan

for single rule networks only

V..load,jmp,scan: enter **V** to load, display, scan and probe a sample of automatically classified rule-space. Various top-right prompts will appear, firstly to allow loading the file (see Filing, chapter 35). The samples currently provided with DDLab are for 1d k=5, 6 and 7 rules only, and are held in the files `five5ss.sta`, `six5ss.sta` and `sev5ss.sta`. New samples can be created (see chapter 33). Once loaded, a further top-right prompt is presented as follows,

list: all-l pos+p, sort-s, quit-q
entropy sandard dev plot: Z-Z lda-L ent-def, smalldots+d:

Enter **q** to skip the further options and continue with the space-time pattern. Otherwise enter **return** to display the mean entropy - standard deviation scatter plot with various further options, for example to probe points on the plot to select rules, and to display the scatter plot in 3d with a vertical frequency axis. These further options are explaned in chapter.

Once a rule sample has been loaded, enterering **V** allows loading a new rule sample, or scanning the space-time patterns in the current sample. The following top-right prompt is presented, for example if the k=5 file `five5ss5.sta` was loaded, which contains 17680 rule,

rule sample: load new-n, rule index/scan 1-17680: *(values shown are examples)*

Enter a number in the range to jump to a sample index. The following further prompr is presented to allow scanning space-time patterns in blocks, for example if index 72 was entered above,

scan in blocks from index 72 -i: *(values shown are examples)*

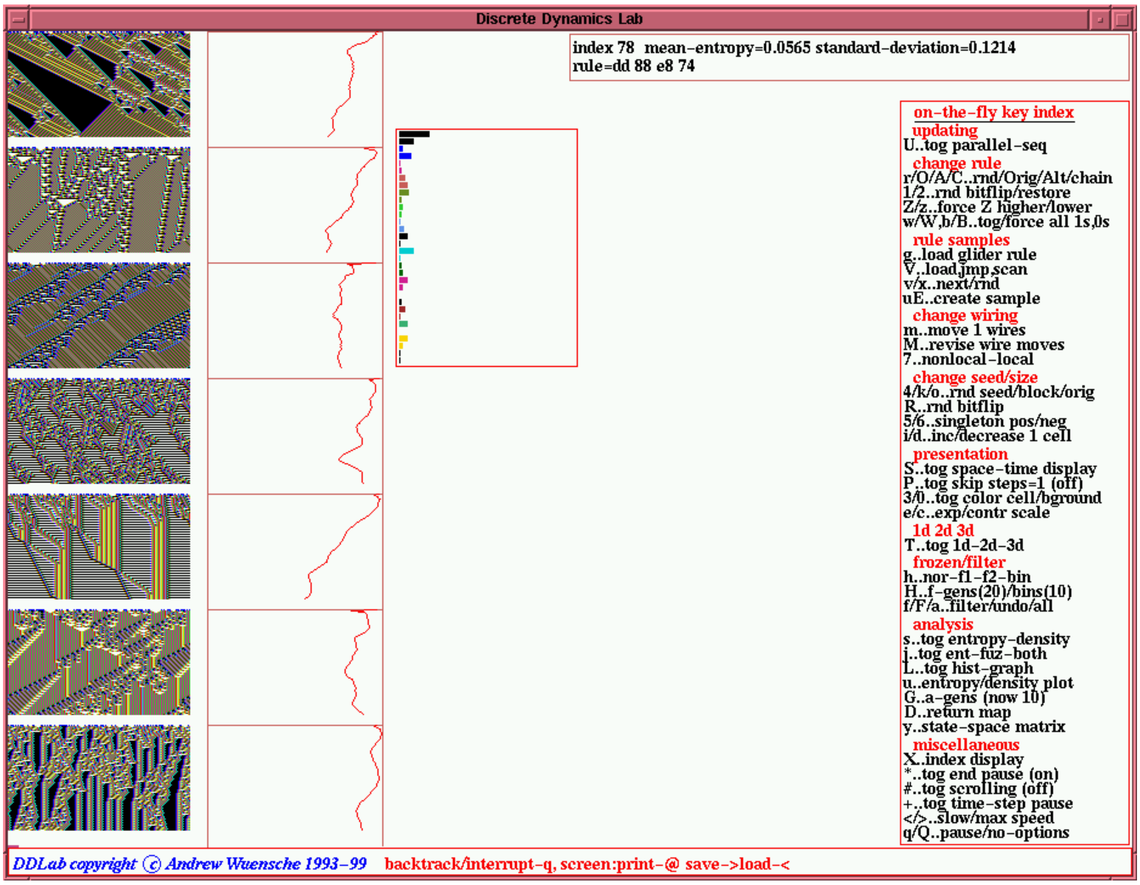
Enter **return** to select the rule at index 72 in the sample and view its space-time patterns in the normal way, or enter **i** to scan successive rules in blocks of time-steps starting at index 72.

If **i** is selected, a further prompt, **set time-steps (def 150):** alows the number of time-steps in the block to be resized.

It is best to toggle "scrolling" off when scanning successive blocks, with on-the-fly option # (see section 32.12.3). Furthermore, the blocks will scan continuously unless the "end pause" option is toggled on with on-the-fly option * (see section 32.12.2). Figure 32.3 shows the DDLab screen scanning in blocks of 88 time-steps. The current rule is shown in hex in a top-right window, for exanple,

index 78 mean-entropy=0.0565 standard-deviation=0.1214
rule=dd 88 e8 74

Note that with "end pause" on, the pause prompt described in section 32.14 will appear. This can be suppressed if required by entering **Q** (see section 32.12.6). Enter **return** for the next set of blocks. To exit scanning in block, interupt with **q**.



index 78 mean-entropy=0.0565 standard-deviation=0.1214
rule=dd 88 e8 74

Figure 32.3: Scanning rule-sample space-time patterns in blocks. In this case the k=5 file `five5ss.sta`, provided with DDLab, was first loaded, then scanning from sample index 72. The blocks wee resized to 88 time-steps, n=150.

32.5.3 next/rnd

for single rule networks only

v/x..next/rnd: once the rule-space sample has been loaded (with **V** above, or in section 32.14.1), enter **v** to scan the space-time patterns of rules successive in the sample, or **x** to jump to a random sample index. In both cases a top-right window shows the sample index, the rule's mean-entropy and standard-deviation (see chapter 33), and the rule itself in hex, for exanple (for a *k*=7 rule),

> **index 316 mean-entropy=0.1165 standard-deviation=0.0805**
> **rule=ae ae 83 a3 35**

Entering **v** will scan successive rules in the sample starting from rule index 0, or from the rule that was "jumped to" with **V** in section 32.5.2.

32.5.4 create sample

uE..create sample: enter **u** followed by **E** to create a sample of automatically classified rule-space, described in chapter 33. Note that the initial **u** initiates an entropy-density plot (see section 32.11.4). If this plot is current, the on-the-fly prompt is **E..create sample**, so enter just **E** to create a sample.

32.6 change wiring

These options allow network wiring to be changed, mutated, on-the-fly.

32.6.1 move x wires

m..move 1 wires: enter **m** to randomly move one wire or a preset number of wires. For a local 1d, 2d or 3d network, first toggle running the network according to its non-local wiring with key **7** (section 32.6.3) to see the effect of the change.

The number of wires to be moved can be reset with **M** in section 32.6.2 below, and will show up in the on-the-fly prompt, for example,

> **m..move 25 wires**

32.6.2 revise wire moves

M..revise wire moves: enter **M** to change the number of wires to be moved with **m** in section 32.6.1 above. The "mutate wiring" prompts (section 28.2) will be presented in a top-right window, for example,

> **relocal-l, wires to move=1/750=0.133%: all-a number-n percentage-p:**

This option also allows the wiring to be made local for 1d, 2d and 3d networks.

32.6.3 nonlocal-local

7..nonlocal-local: enter **7** to toggle between running the network according to its non-local wiring (if it has it), or running as if the wiring was local (for 1d, 2d or 3d networks). The original non-local wiring held in memory remains intact when local wiring is toggled. The current status is given by the prompt order, i.e. **7..local-nonlocal** indicates that the network is currently being run as if it

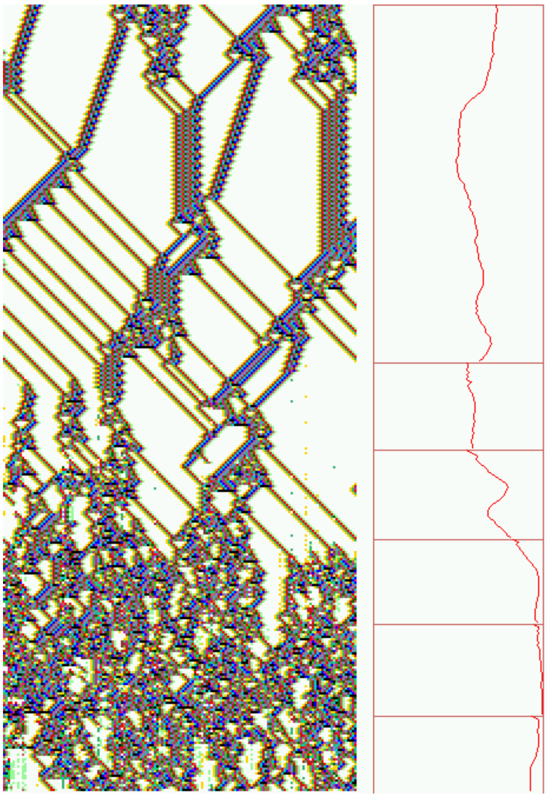


Figure 32.4: Randomizing the wiring of a 1d CA by stages, for rule 6c 1e 53 a8, n=150. The wire moves were first revised to 2% (15 out of 750 wires) with on-the-fly option **M** (section 32.6.2), then the wiring was cumulativly randomized with on-the-fly option **m** (section 32.6.1), at the points indicated by horizontal lines in the input-entropy plot. Note that glider structure is progressively degraded.

has local wiring. If the wiring held in memory is itself local, there will be no difference between local and nonlocal. However, wire moves made to the local wiring with **m** in section 32.6.1 will only show up if non-local wiring is toggled.

32.7 change seed/size

These on-the-fly options amend, mutate, the current state or seed, and also the of size of the network (for 1d networks 1d only displayed in 1d).

32.7.1 rnd seed/block/orig

4/k/o..rnd seed/block/orig: enter enter **4**, **k** or **0** to amend the seed as described below,

4 . . . enter **4** to reset the current network state at random, but with the density of 1s bias set in 21.3.2.

k . . . enter **k** to reset the network state as a central random block, but with the density of 1s bias of the block set in 21.3.2, (in 1d, 2d or 3d). The size of the block was set in section 21.3, and the density of 1s bias of the block in section 21.3.2.

o . . . enter **o** to restore the network state to the original, (as set in section 21).

32.7.2 rnd bitflip

R..rnd bitflip: enter **R** to flip a bit in the current state at a random position.

32.7.3 singleton pos/neg

5/6..singleton pos/neg: enter enter **5** or **6** to reset the seed as a positive or negative singleton seed, a central 1 surounded by 0s, or a central 0 surounded by 1s.

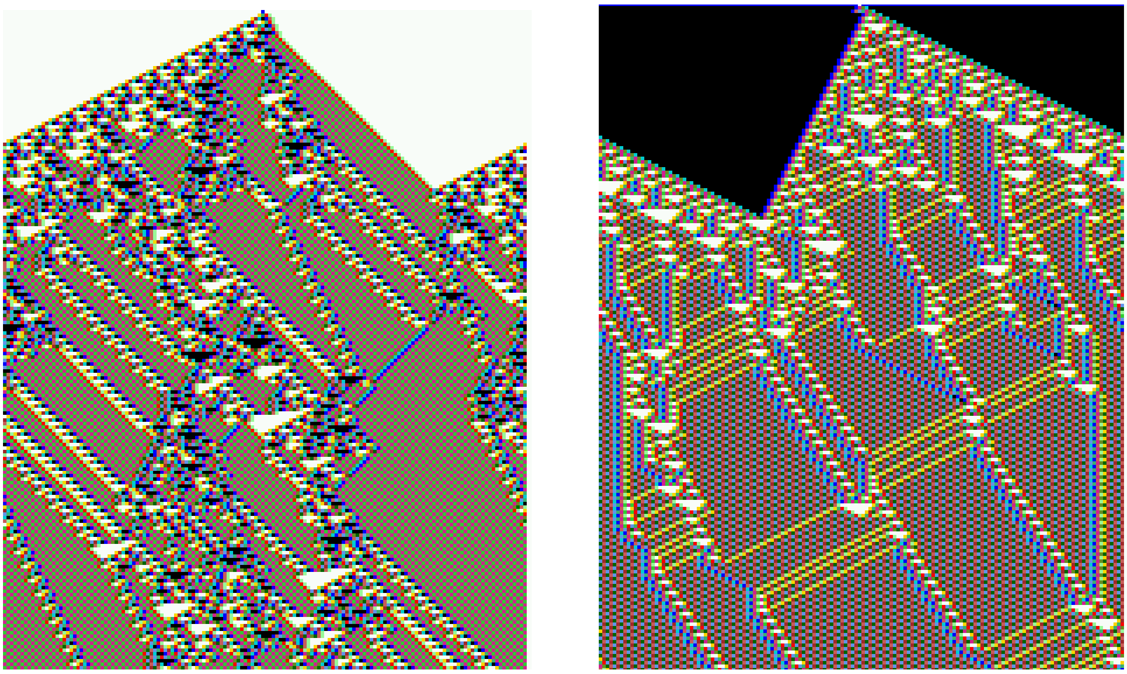


Figure 32.5: *left*: a positive singleton seed, a central 1 surounded by 0s, and *right*: a negative singleton seed, a central 0 surounded by 1s, set on-the-fly by entering **5** or **6**. *left*: rule 96 8c 4f 76, *right*: rule d2 8f 02 2c. n=150, about 190 time-steps.

32.7.4 inc/decrease 1 cell

for 1d networks only, shown in 1d

i/d..inc/decrease 1 cell: enter enter **i** or **d** to increase or decrease the size of the network by 1 cell. The original network is preserved as far as possible. The extra cell is assigned local wiring for networks that were originaly defined as local, otherwise random wiring is assigned. For a mixed rule, homogeneous k network, one of the rules from the current set is assigned to the extra cell. For a mixed k network, a random k in the current range is assigned, and a random rule. The change in size is indicated in a top-right window, for exanple,

n increased 166 -> 167 or **n decreased 167 -> 166**.

32.8 presentation

These options amend various aspect of the presentation of space-time patterns on-the-fly, including color and scale.

32.8.1 tog space-time display

S..tog space-time display: enter **S** to toggle the display of space-time graphics on/off. Suppressing space-time patterns, which are still iterating, speeds up other processes, for example creating a sample of automatically classified rule-space (chapter 33), histograms of "damage" (section 31.17), and histograms of attractors or skeletons section 31.18.

32.8.2 tog skip steps

P..tog skip steps=1 (off): enter **P** to toggle between normal space-time patterns and skipping the presentation of a given number of time-steps. The number of time-steps to be skipped can be changed in section 32.14.6. The default is 1, i.e. showing every seconf time-step. The current number of time-steps, and if skipping is currently on or off, is indicated in the prompt, for example,

> **P..tog skip steps=3 (on)**

32.8.3 tog color cell/bground

3/0..tog color cell/bground: These on-the-fly options amend the color of space-time patterns.

3 ... enter **3** to toggle between cell colors according to the neighborhood "looked up" and the actual cell value 0 or 1, represented by a dark and a light color. This can also be preset in section 31.4. Examples are shown in figure 32.10.3 (a,b), and figures 31.1, 31.2 and 31.3 for 1d, 2d and 3d networks.

0 ... if cells are colored by value (see above), and for 1d and 2d networks only, enter **0** to toggle between the default background color, white, and light green (for a white background), or black and dark green (for a white background). this is sometime necessary to show up the background.

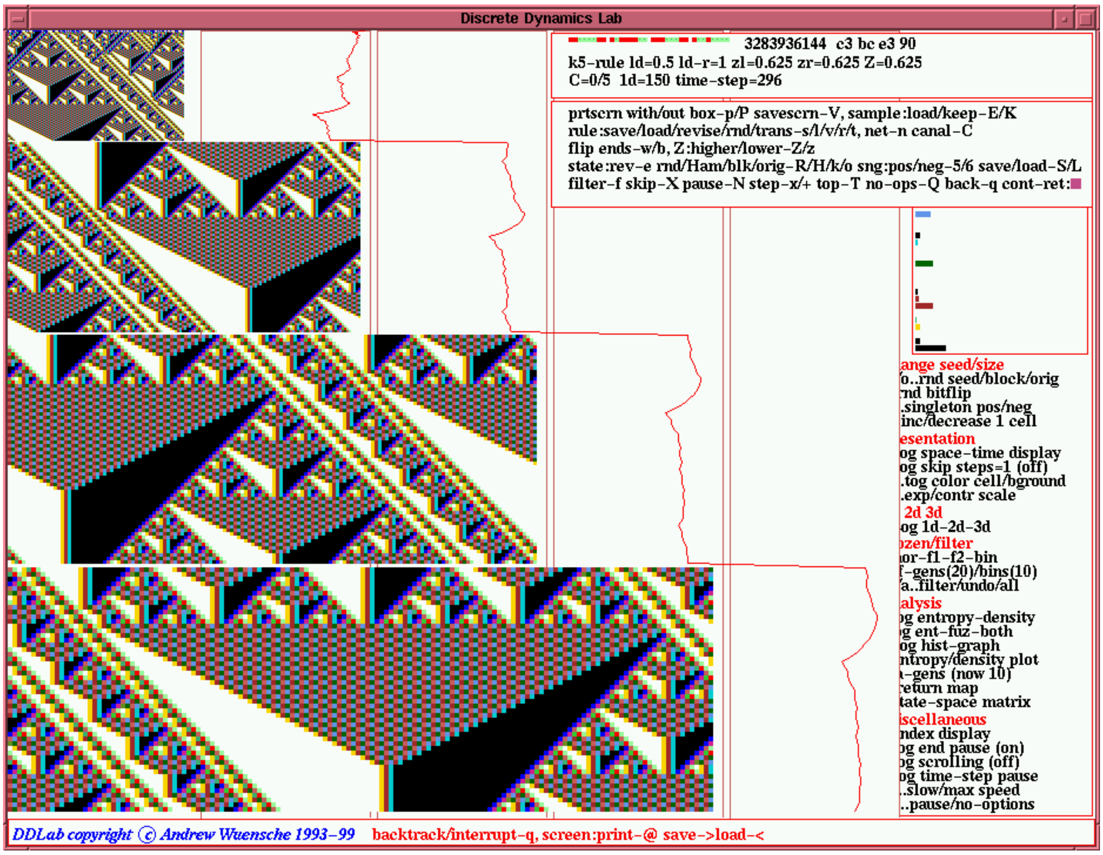
32.8.4 tog sound

DOS only

$..tog sound: for DOS only, toggle sound generated by the network. Both the pitch and delay are generated by 8 bits near the center of the network.

32.8.5 expand/contr scale

e/c..expand/contr scale: enter **e** to expand the scale of cells in the space-time pattern by 1 pixel, enter **c** to contract the scale by 1 pixel. This changes the scale of the space-time patern in 1d, 2d or 3d.



```
3283936144  c3 bc e3 90
k5-rule ld=0.5 ld-r=1 zl=0.625 zr=0.625 Z=0.625
C=0/5  1d=150 time-step=296

prtscrn with/out box-p/P savescrn-V, sample:load/keep-E/K
rule:save/load/revise/rnd/trans-s/l/v/r/t, net-n canal-C
flip ends-w/b, Z:higher/lower-Z/z
state:rev-e rnd/Ham/blk/orig-R/H/k/o sng:pos/neg-5/6 save/load-S/L
filter-f skip-X pause-N step-x/+ top-T no-ops-Q back-q cont-ret:
```

```
ange seed/size
/o..rnd seed/block/orig
rnd bitflip
.singleton pos/neg
inc/decrease 1 cell
esentation
og space-time display
og skip steps=1 (off)
.tog color cell/bground
.exp/contr scale
 2d 3d
og 1d-2d-3d
ozen/filter
nor-f1-f2-bin
f-gens(20)/bins(10)
/a..filter/undo/all
alysis
og entropy-density
og ent-fuz-both
og hist-graph
ntropy/density plot
n-gens (now 10)
return map
tate-space matrix
iscellaneous
ndex display
og end pause (on)
og scrolling (off)
og time-step pause
..slow/max speed
..pause/no-options
```

DDLab copyright © Andrew Wuensche 1993–99   backtrack/interrupt–q, screen:print–@ save–>load–<

Figure 32.6: Expanding and contracting 1d space-time patterns. At the top the scale of a cell is 1 pixel, then below: 2, 3 and 4 pixels. **e** was entered to increase the scale by 1 pixel on-the-fly at each key press. **c** contracts the scale by one pixel. n=150, k=5 rule c3 bc e3 90. The network details are shown in the top-right window. Below are further prompts described in section 32.14. Expanding and contracting also works for 2d, 2d + time, and 3d networks.

32.9 1d 2d 3d

Space-time pattern can be displayed in a dimension other than the "native" dimension. A 1d network can be displayed in 2d or 3d, a 2d in 1d or 3d, and 3d in 1d or 2d. The initial display dimension can be set to someting other than the native dimention in section 31.6, or changes can be toggled on-the-fly, described here. In addition: If a network is being displayed in 2d, it can be toggled between 2d and 2d+time.

If a network is being displayed in 3d, the default display method in Unix and Linux is to show each time-step as one 3d snapshot. This can be toggled to build up successive horizontal slices, similar to 2d+time. For DOS, this is the default and only method.

32.9.1 tog 1d-2d-3d

T..1d-2d-3d: if 2d+time is not active (see section 32.9.2 above), **T** is a 3-way toggle to change the display of space-time patterns between 1d, 2d and 3d. Space-time pattern can be displayed in a dimension that is not the “native” dimension as described in section 31.6, from where the presentation dimension can also be pre-selected. The current status is shown in the prompt, i.e. if 2d is active the prompt is **2d-3d-1d**, if 3d is active the prompt is **3d-1d-2d**. Note that only native 1d networks shown in 1d can be scrolled (see section 32.12.3).

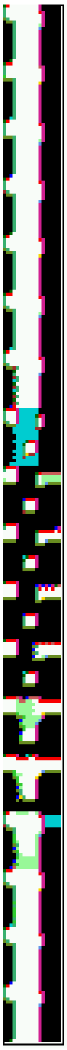
The automatic method for converting between dimensions is as follows:

1d to 2d: $i =$ the smallest factor $\geq \sqrt{\boldsymbol{n}}$, $j = \boldsymbol{n}/\boldsymbol{i}$

1d or 2d to 3d: $i =$ the smallest factor $\geq \sqrt[3]{\boldsymbol{n}}$, $j =$ the smallest factor $\geq \sqrt{\boldsymbol{n}/\boldsymbol{i}}$, $h = \boldsymbol{n}/(\boldsymbol{i} \times \boldsymbol{j})$
For 1d prime $\boldsymbol{n}$, the 2d and 3d representation will be a long string length $\boldsymbol{n}$.

3d to 2d The 2d represention consists of $\boldsymbol{h}$ horisontal slices (levels), $\boldsymbol{i} \times \boldsymbol{j}$, one below the other, starting from the top, i.e. a series of horisontal cross-sections through the 3d volume (see figure 32.9.1).

2d and 3d to 1d This will be represented a nornal 1d space-time patterns, were the cells are indexed $\boldsymbol{n} - \boldsymbol{1}$ to 0 from left to right. The cell scale is automatically reducd to 1 pixel.



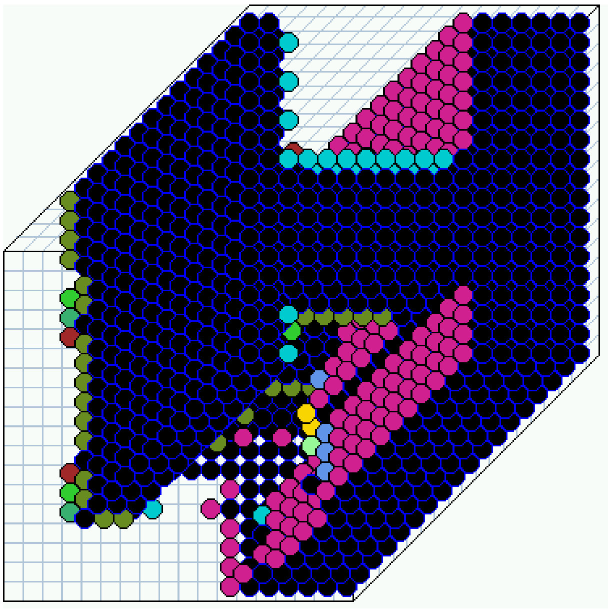


Figure 32.7: A 3d network shown in 2d by toggling 3d to 1d to 2d on-the-fly. *above*: The 3d space-pattern, 18x18x18. The projection is isometric seen from below, as if looking up at the inside of a cage. Cells are shown colored according to neighborhood lookup by default for a clearer picture (instead of by value: 0,1). *left*: The 2d represention consists of **18** horisontal slices, one below the other, starting at the top, i.e. a series of horisontal cross-sections through the 3d volume. This is a k=*7* (nearest neighbor) 3d CA, with the totalistic code 1110100. The network was started with a random initial state, and settled to this semi-stable pattern.

32.9.2 tog 2d-2d+time

t..tog 2d-2d+time: for networks being displayed in 2d, enter **t** to toggle between normal 2d, and 2d plus a time dimention, drawn as a 3d isometric projection (see figure 4.12), best seen when cells are colored according to the neighborhood (see 32.8.3). The current status is shown in the prompt, i.e. if 2d+time is active the prompt is **t..2d+time-2d**. See figure 4.12.

32.9.3 draw 2d plane

p..2d draw plane: only if 2d+time is active (see above), enter **p** to draw a 2d translucent plane within the 2d+time projection at the current time-step to highlight the isometric geometry. This plane is also drawn whenever the seed, rule/s or wiring are changed. See figure 4.12.

32.9.4 tog 3d slice

I..tog 3d slice: If a network is being displayed 3d, enter **I** to toggle between the default display method in Unix and Linux, showing each time-step as one 3d snapshot, and showing the time-step build up of successive horizontal slices or layers starting at the top, similar to 2d+time. For DOS, this is the default and only method.

32.10 frozen/filter

"Frozen" options allow visualizing the activity/stablity of cells. "Filter" options allow progressivly filtering categories of cells, starting with the background domain, to show up interacting configurations (gliders) more clearly. Note the frozen and filter options can be used together. Filtering works for 1d, 2d and 3d networks with a homogenious rule and k (including RBN), but the method relies on data from the input-entropy histogram, so the input-entropy option must be active, the default for 1d networks. For 2d and 3d networks this option would need to be set in section 31.10.

32.10.1 nor-f1-f2-bin

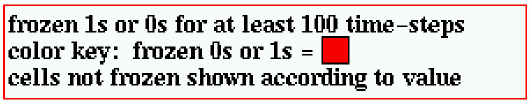
h..nor-f1-f2-bin: This on-the-fly option is a 4-way togle that allows "frozen cells", that have not changed over the last $\boldsymbol{x}$ time-steps to be highlighted (by two methods), or cells to be colored according the fraction of time they have been "on" (i.e. 1) in the same window of x time-steps, falling into preset "frequency bins", examples are shown in figure 32.10.3 (b,c,d,e).

$\boldsymbol{x}$, the "frozen generation" size, and also the number of bins and the bin boundaries can be reset in section 32.10.2. The initial defaults are, frozen generation = 20, number of bins =10, and bin boundaries equally spaced.

Enter **h** to toggle on-the-fly through the following four ways of of displaying cells, **nor-f1-f2-bin** then back to **nor**. The option in the on-the-fly window changes accordingly.

nor ... "normal" colors. If the 4-way toggy returns to **nor**, cells are shown according to the actual cell value 0 or 1, represented by a dark and a light color, even if color by neighborhood "looked up" was originaly set in section 32.8.3.

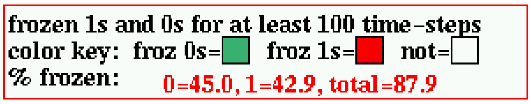
f1 . . . "frozen" (method 1): frozen cells are colored red. Dynamic, unfrozen cells, are show "normal" as above. The following reminder is shown in a top-right window,



frozen 1s or 0s for at least 100 time-steps
color key: frozen 0s or 1s = 🟥
cells not frozen shown according to value

see figure 32.10.3 c.

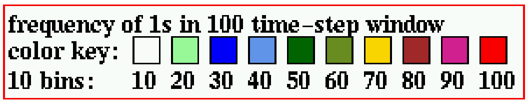
f2 . . . "frozen" (method 2): frozen 1s are colored red, frozen 0s green. Dynamic, unfrozen, cells are colored white. The following reminder, also indicating the percentage of frozen 1s, 0s, and the total frozen, is shown in a top-right window,



frozen 1s and 0s for at least 100 time-steps
color key: froz 0s=🟩 froz 1s=🟥 not=□
% frozen: 0=45.0, 1=42.9, total=87.9

see figure 32.10.3 d.

bin . . . colors are assigned from a spectrum representing the fraction of time each cell have been "on" (i.e. 1) in the frozen generation time window of x time-steps, falling into a preset number of frequency "bins" and bin boundaries. The following reminder showing the bin color scheme and bin boundaries is shown in a top-right window,



frequency of 1s in 100 time-step window
color key: □ 🟩 🟦 🟦 🟩 🟩 🟨 🟫 🟪 🟥
10 bins: 10 20 30 40 50 60 70 80 90 100

see figure 32.10.3 e.

The bin boundaries give the upper limit of each bin. these parameters can be reset in section 32.10.2.

32.10.2 frozen parameters

H..f-gens (now 20)/bins(10): Enter **H** to change "frozen" parameters, the default number of frozen generations, or the number of bins and the bin boundaries, relating to section 32.10.1 above. The following top-right prompts are presented in turn (showing the initial defaults),

enter new 'frozen' Generation size (now 20):
reset bin boundaries, enter new total (max 10, now 10):

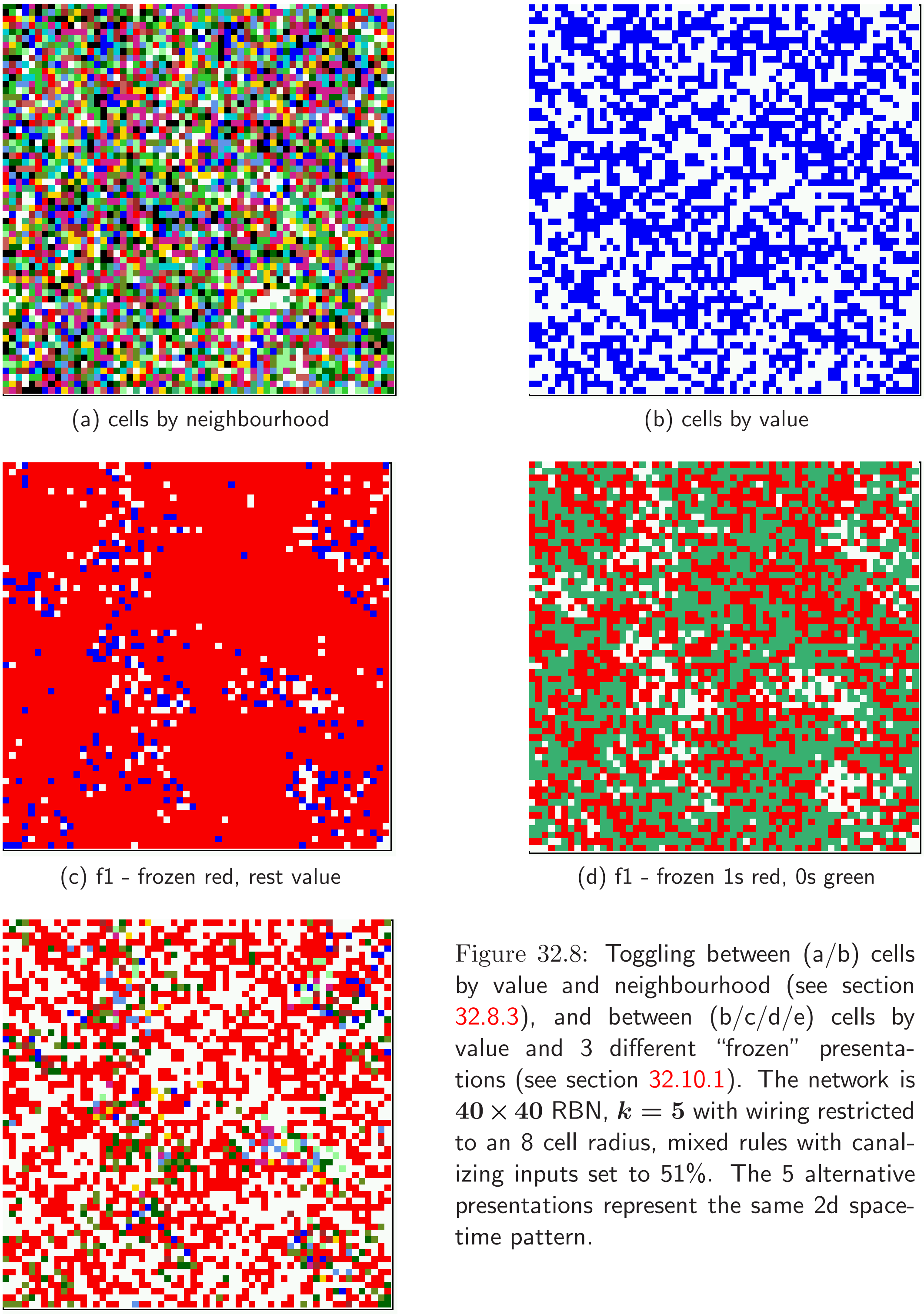
Note that the if the generation size is made less than the number bins, the number (and maximum number) of bins is reduced automaticaly, as this cannot exceed the generation size.

32.10.3 frozen generations

The frozen generation size is the number of time-steps that a cell must remain unchanged to qualify as "frozen". The following top-right prompt allows this to be changed,

enter new 'frozen' Generation size (now 20):

Enter a new frozen generation size, which also sets the maximum number of bins. The current frozen generation size is shown in both the on-the-fly window and in this prompt. The initial defailt is 20 time-steps.



(a) cells by neighbourhood

(b) cells by value

(c) f1 - frozen red, rest value

(d) f1 - frozen 1s red, 0s green

(e) bin - frequency of 1s in time window

Figure 32.8: Toggling between (a/b) cells by value and neighbourhood (see section 32.8.3), and between (b/c/d/e) cells by value and 3 different "frozen" presentations (see section 32.10.1). The network is 40×40 RBN, $k = 5$ with wiring restricted to an 8 cell radius, mixed rules with canalizing inputs set to 51%. The 5 alternative presentations represent the same 2d space-time pattern.

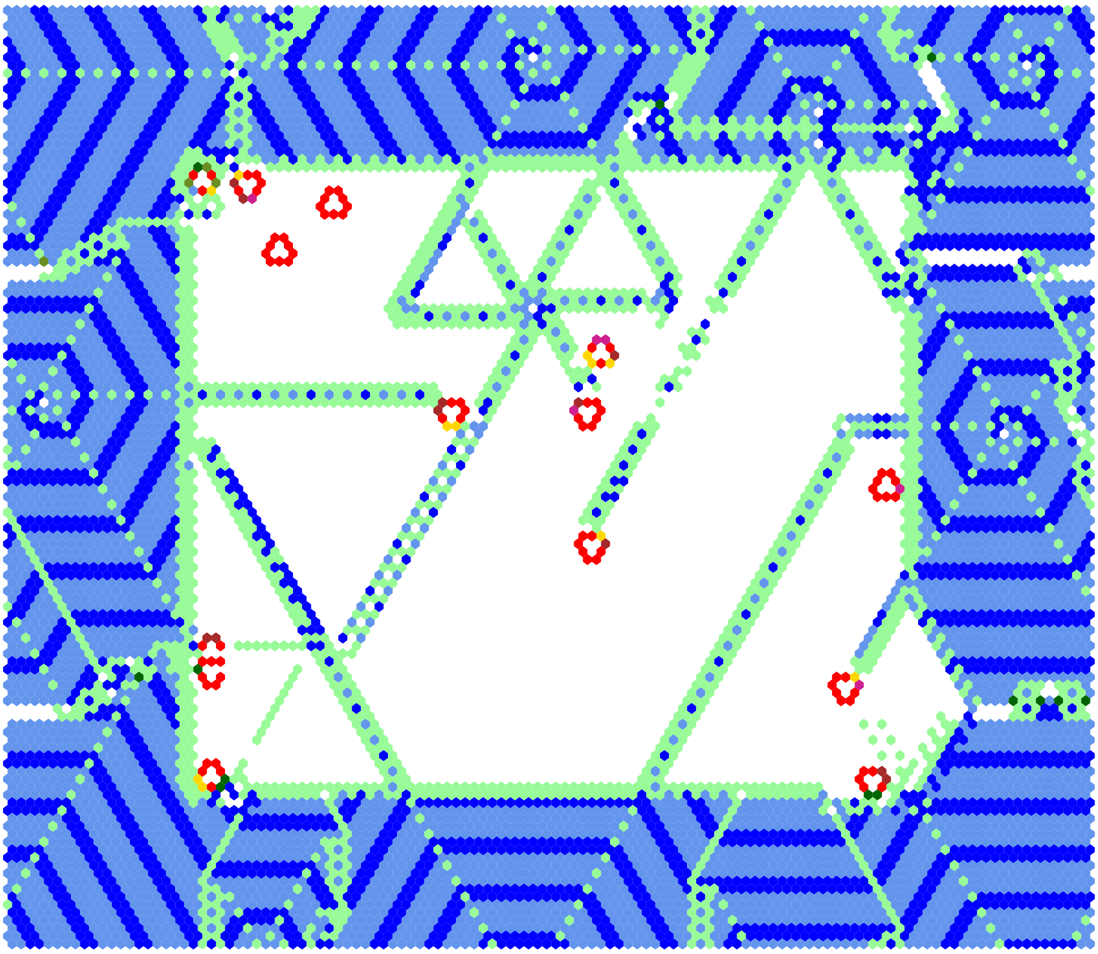


Figure 32.9: Space-time snapshot of a 2d CA inside another 2d CA with different $\boldsymbol{k}$, similar to figure 19.6, but coloured according to frequency bins (see section 32.10.1). The 2d $\boldsymbol{k} = \mathbf{7}$ CA spiral-rule (80×80) was loaded into a 2d $\boldsymbol{k} = \mathbf{6}$ complex CA (120×120) which generates spiral structures.

32.10.4 frequency bins

To change the number of bins and bin boundaries, relating to section 32.10.1, the following top-right prompt is presented after the frozen generation prompt in section 32.10.3,

...
reset bin boundaries, enter new total (max 10, now 10):

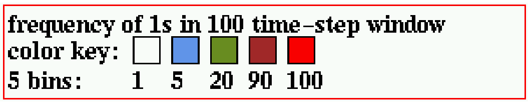
The current number of bins is shown in both the on-the-fly window and in this prompt. First enter the new bin number, which cannot be greater than the frozen generation size. Once set, a further top-right prompt allows setting the bin boundaries,

unequal bins-u, equal-def:

Enter **return** to set equal size bins (or as equal as possible) automatically. Otherwise enter **u** to set the bin boundaries by hand. The bin boundaries give the upper limit of each bin. The following top-right prompts are presented in sequence,

> **enter bin boundary 1 (def 2):**
> **enter bin boundary 2 (def 4):**
> etc...

When cell colors are assigned according to "frequency bins" (section 32.10.1), a reminder of the current parameters is shown in a top-right window. In the example below, the "frozen generation" size was set to 100, the number of bins to 5, and the bin boundaries to 1, 5, 20 and 100.



```
frequency of 1s in 100 time-step window
color key: □ 🟦 🟩 🟫 🟥
5 bins:     1  5  20  90  100
```

The bin boundaries must not exeed the current number of frozen generations, otherwise the following top-right warning message is presented,

> **invalid bin boundaries, cont-ret:**

In this case enter **return** to reset the bin boundaries. The initial defaults are, frozen generation size=20, number of bins=10, and bin boundaries equally spaced.

32.10.5 filter/undo/all

f/F/a..filter/undo/all: Enter **f** to progressively filter space-time, **F** to unfilter in reverse order, and **a** to immediately restore the space-time pattern. Filtering works by suppressing the printing of cells that updated with reference to the current most frequent unsuppressed neighborhood. This suppresses the background domain to show up gliders more clearly (see figures 5, 32.10 - 32.12).

A dot is shown alongside the look-up frequency histogram (see section 31.10) indicating which neighbourhoods are currently suppressed (see figure 32.11. As well as progressively filtering/unfiltering, particular neighbouhoods can be filtered in isolation in section 32.14.7. On-the-fly filtering works for any single rule network, and for 2d and 3d provided that the input-entropy was selected in section 31.10.2.

For most glider rules, only a few neighbourhoods need to be suppressed to filter domains. Rules with very complicated background domains, such as the k=3 rules 54 and 110, must first be transformed to equivalent rules with greater k (k=5 in this case) for successful filtering, which requires suppressing a number of the k=5 neighbourhoods (see figures 32.12), 32.10, 32.11).

Filtering can also reveal discontinuities within or between chaotic domains (see figure 32.13.

32.11 analysis

These on-the-fly options relate to various methods of real-time analysis of space-time patterns.

32.11.1 tog entropy-density

s..tog entropy-density: enter **s** to toggle between showing the input-entropy and pattern density as desribed in section 31.10 and figure 31.5. The current status is given by the prompt order, i.e.

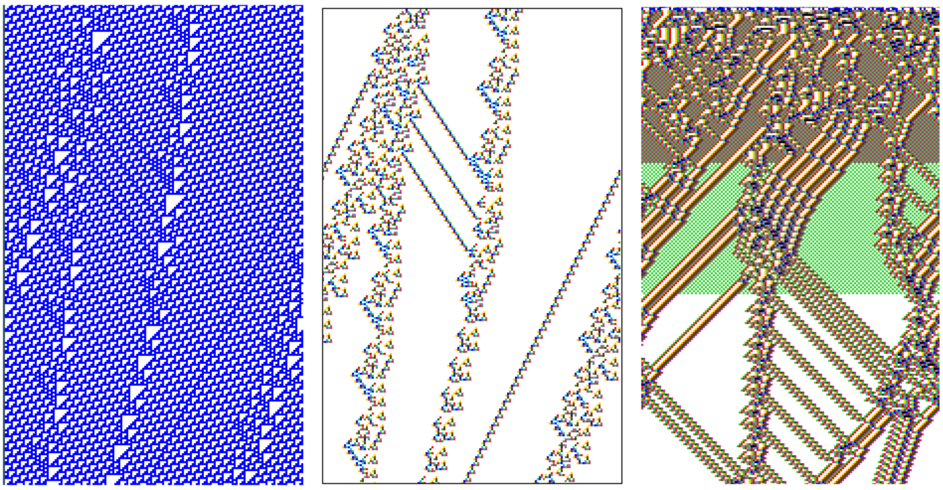


Figure 32.10: Examples of filtering space-time patterns to show up gliders more clearly. *left*: space-time patterns of the k=3 rule 110, transformed to the equivalent k=5 rule 3cfc3cfc, n=150 (cells by value). *centre*: the same space-time pattern filtered (cells by neighbourhood lookup). *right*: Space-time patterns of the k=5 rule 360a96f9. Cells are shown by neighbourhood lookup, and are progressively filtered in 2 stages. About 200 time-steps are shown in each case.

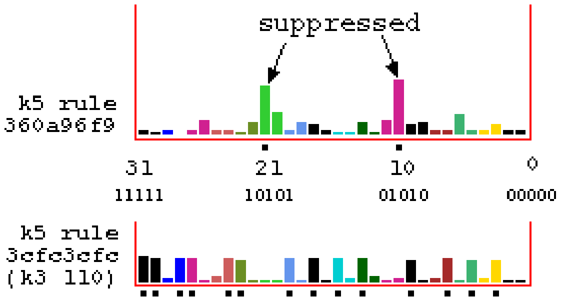
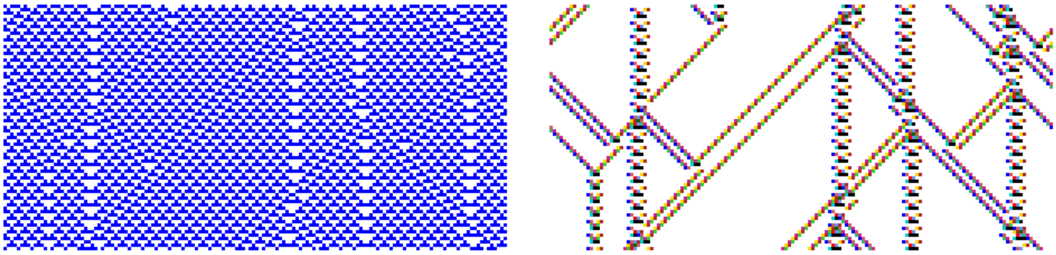


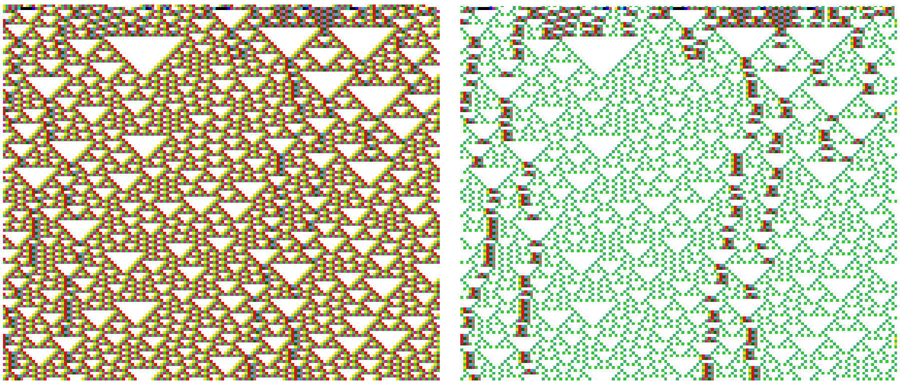
Figure 32.11: The look-up frequency histograms relating to figure 32.10. (*above*) k=5 rule 360a96f9, (*below*) k=3 rule 110 transformed to k=5 rule 3cfc3cfc. Suppressed neighborhoods are indicated with a dot. Note that the look-up frequency histogram is presented on its side in DDLab.



cells by value

cells by look-up and filtered

Figure 32.12: Space-time patterns from the same initial state showing interacting gliders, which are embedded in a complicated background. ***Left***: cells by value. ***Right*** cells by neighbourhood lookup, with the background filtered. The k=3 rule 54 was transformed to its equivalent k=5 rule (hex) 0f3c0f3c, n=150.



k=3 rule 18 k=3 rule 18, partly filtered

Figure 32.13: Unfiltered and partly filtered space-time patterns of k=3 rule 18 (transformed to k=5 rule 030c030c). n=150, about 130 time-steps from the same random initial state, showing discontinuities within the chaotic domain.

s..tog density-entropy indicates that **density** is current. These measures relate to a moving window of time-steps (default 10) that can be reset in section 32.11.5. For 2d and 3d networks the input-entropy and pattern density must first be set in section 31.5.

32.11.2 tog ent-fuz-both

j..tog ent-fuz-both: if input-entropy is active (see 32.11.1), **j** is a 3-way toggle to change the display between the following,

ent ... showing just the input-entropy graph.

fuz ... showing the lookup frequency of each neighborhood as a set of superimposed graphs. The colors of the graphs are the same as the colors of the bars in the input frequency histogram described in section 31.10.2.

both ... showing both of the above simultaneously.

The current status is shown in the prompt order, i.e. if **fuz** is active the prompt is **fuz-both-ent**, if **both** is active the prompt is **both-ent-fuz**. Figure 32.11.5 gives an example.

32.11.3 tog hist-graph

L..tog hist-graph: if input-entropy is active (see 32.11.1), enter **L** to toggle between showing the input-frequency histogram in 2d, and expanded to 3d, with an extra time axis, shown as an isometric projection, as in figure 31.6 (see section 31.10.2).

32.11.4 entropy/density plot

u..tog entropy/density graph: enter **u** to show/delete a density/entropy scatter plot, where the entropy (x) axis is plotted against the density (y) axis, at each time-step, in the bottom center of the screen.

32.11.5 a-generations (analysis)

G..a-gens (now 10): Enter **G** to change the default number of analysis generations, the size of the moving window of time-steps relating to input-entropy and pattern density (options 32.11.1 - 32.11.4 above). The following top-right prompt is presented,

> **enter new 'analysis' Generation size (now 20):**

The current number of analysis generations is shown in both the on-the-fly window and in this prompt. The initial default is 10 time-steps.

Superimposed plots for a number of k=5 complex rules are shown in figure 32.15. Each rule produces a characteristic cloud of points which lie within a parabolic envelope because high entropy is most probable at medium density, low entropy at either low or high density. Each complex rule produces a plot with its own distinctive signature, with high input-entropy variance. Chaotic rules, on the other hand, give a compact cloud at high entropy (at the top of the parabola). For ordered rules the entropy rapidly falls off with very few data points because the system moves rapidly to an attractor.

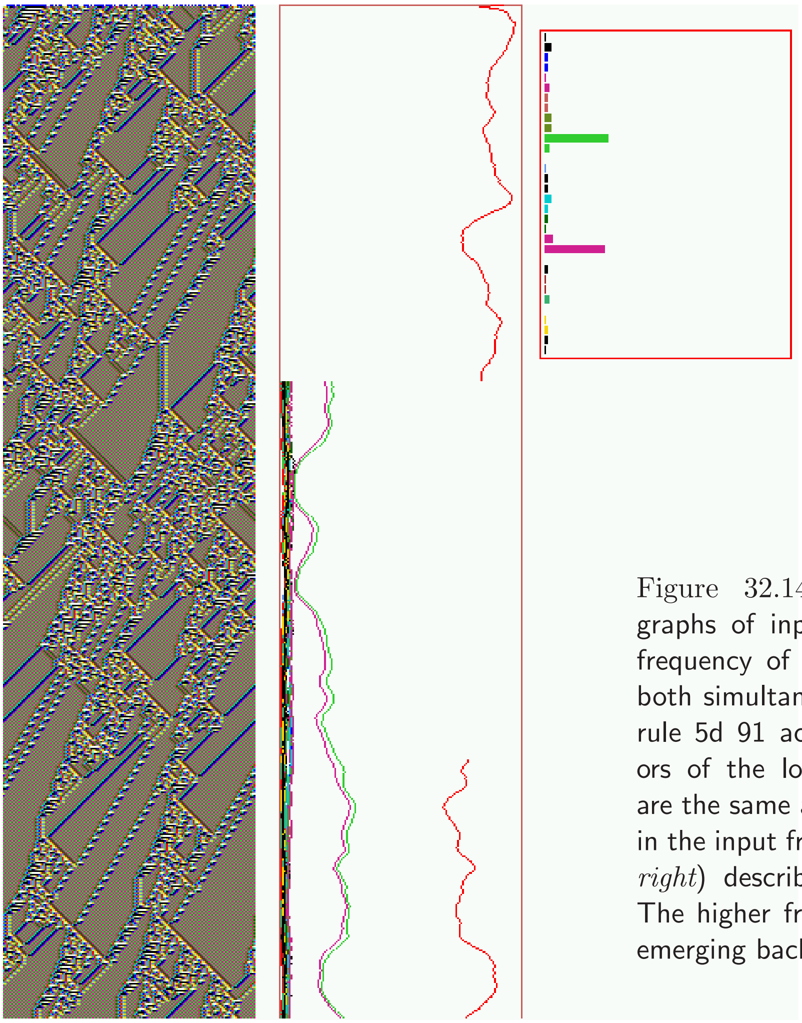


Figure 32.14: Toggling between graphs of input-entropy, the lookup frequency of each neigborhood, and both simultaneously, for the k=5 CA rule 5d 91 ac 17, n=150. The colors of the lookup frequency graphs are the same as the colors of the bars in the input frequency histogram (*top right*) described in section 31.10.2. The higher frequencies represent the emerging background domain.

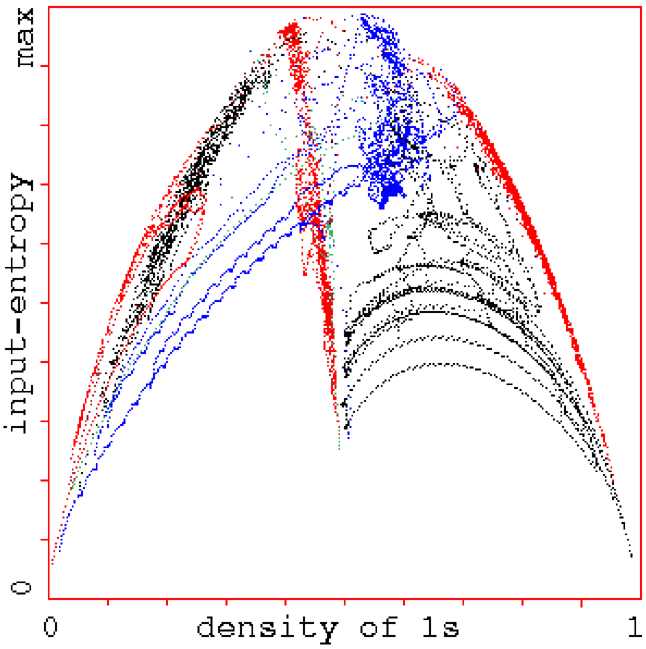


Figure 32.15: Entropy/density scatter plot[27]. Input-entropy is plotted against the density of 1s relative to a moving window of 10 time-steps. Plots for a number of k=5 complex rules (n=150) are show superimposed, each of which has its own distinctive signature, with a marked vertical extent, i.e. high input-entropy variance. About 1000 time-steps are plotted from several random initial states for each rule.

32.11.6 return map

D..return map: enter **D** to show/delete the "return map" scatter plot in a 2d-pase plane (see also section 31.14). Each state (bitstring) $B_0, B_1, B_2, B_3 \ldots B_{n-1}$ is converted into a decimal number 0-2 as follows,

$$B_0 + B_1/2 + B_2/4 + B_3/8 + \cdots + B_{n-1}/2^{n-1}$$

As the network is iterated, this state value at time-step t (x-axis) is plotted against the state value at timestep t+1 (*y-axis*) in the bottom center of the screen. From a classical dynamical systems viewpoint, note the fractal structure of the resulting trajectories and attractors. An example is shown in figure 31.8.

32.11.7 state-space matrix

y..state-space matrix: enter **y** to show/delete the "state-space matrix" (see also sections 31.14 and 24.5). The state-space matrix plots each state in the space-time pattern on a 2d grid near the bottom of the screen, plotting the left half of the bit string against the right half. The y-axis represents the right $n/2$ bits. If n is odd, the extra bit is included on the left, and the grid is a flat rectangle, otherwise the grid is square. An example is shown in figure 31.7.

32.12 miscellaneous

These final miscellaneous on-the-fly options control the display of the on-the-fly key index itself, and also pausing, scrolling, stepping, and changing the speed of space-time patterns.

32.12.1 index display

X..index display: enter **X** to show/delete the on-the-fly key index itself.

32.12.2 end pause

for 1d (and 2d+time) space-time patterns only

***..tog end pause (on)**: enter * (the star key) to toggle the end pause on/off. If scrolling is not active, 1d space-time patterns (and 2d+time) are displayed in successive vertical sweeps. If the end pause on, when the itterations reach the foot of the screen further options are presented in a top-right window (see section 32.14), otherwise the sweeps continue indefinitely. The current status is indicated in the prompt by **(on)** or **(off)**.

32.12.3 scrolling

for 1d pace-time patterns only

#/&..tog scrolling: enter # (the hash key[1]) or **&** to toggle the scrolling on/off (see also section 31.8). 1d space-time patterns can either be displayed as successive vertical sweeps or to be scrolled upwards. The current status is indicated in the prompt by **(on)** or **(off)**. If scrolling is set, the input-entropy or pattern density plots (if set) will also be scrolled (see sections 32.11.1 and 31.10).

32.12.4 time-step pause

+..time-step pause: enter + (the plus key) to pause after each time-step (this turns scrolling off). The following top-right prompt is presented,

time-step 6828, next-ret, reset count-r, end pause -q: *(values shown are examples)*

Enter **return** for the next time-step, **r** to reset the time-step count to 1, or **q** to end the time-step pause.

32.12.5 slow/max speed

</>..slow/max speed: enter < (the left arrow key) to slow down the iteration speed of space-time patterns by half (eventually iteration will stop). Enter < to restore iteration back to full speed.

32.12.6 pause/no options

q/Q..pause/no-options: enter **q** to pause space-time patterns. This will also display details about the network (section 32.13.1) and present further options (section 32.14) in top-right windows. To avoid showing these windows which may cover required parts of the screen, enter **Q**. Thereafter **q** will pause but no longer displays the windows, and the following botton right prompt is presented instead,

[1] & is provided as an alternative because # is apparently not available on some Mac systems

ops-Q cont-ret:

Enter **Q** to restore the original pause method, where **q** will pause space-time pattern and display further option windows, or **return** to continue space-time patterns.

32.13 Interrupting space-time patterns

If the space-time pattern itteration is interupted on-the-fly with **q** or if "end pause" is set in section 32.12.2 and the space-time pattern reaches the foot of the screen, the program will pause.

Two top-right windows are displayed (if not suppressed in section 32.12.6). The upper window (single rule networks only) shows rule data described in section 32.13.1 below. The second window presents a number of further space-time pattern options, described in section 32.14.

32.13.1 Rule details for space-time patterns

for single rule networks only

On pausing the space-time pattern, a top-right window shows data about the current rule similar to section 16.19. Note that showing this window when pausing can be suppressed in section 32.12.6. The window is updated as rules are transformed or mutated.

An example of the data in a rule window for a $[\boldsymbol{v},\boldsymbol{k}]$=[2,5] rule is shown below,

■·■····■ ···■■·■■ ■■·····■ ··■··■■· **2702950694 a1 1b c1 26** *(the rule)*
v2k5 ld=0.406 ld-r=0.812 zl=0.617 zr=0.578 Z=0.617188
C=0/5 1d=150 time-step=1597

key to rule details

(the rule) . . . its bit pattern (or start), in decimal (for small tables), in hex (or start).

k5-rule . . . the neighborhood size *k*.

ld, ld-r . . . $\boldsymbol{\lambda}$ parameter, $\boldsymbol{\lambda}$ ratio.

zl, zr, Z . . . $\boldsymbol{Z_{left}}$, $\boldsymbol{Z_{right}}$, the $\boldsymbol{Z}$ parameter.

C=0/5 . . . the number of canalizing inputs, in this case none out a possible 5 (k=5).

3** . . . (if applicable) shows exactly which of the *k* inputs are canalizing (if none this is not shown), in this example there is 1 canalizing input, at neighborhood index 3.

1d=150 . . . the network dimention and size for 1d, or the equivalent for 2d or 3d, for example, **2d=40x40** or **3d=20x20x20**.

time-step . . . the current time-step number.

32.14 Space-time pattern pause options

On pausing the space-time pattern, a top-right window displays a range of further context dependent "pause options" (some similar to the options in section 30.5), for amending the rule, network, seed, and some other functions such as displaying the network, network-graph and generating a Postcipt file. Showing this window when pausing can be suppressed in section 32.12.6.

The following is presented in a top-right window,

rule:save/load/revise/rnd/trans-s/l/v/r/t net-n sample:load/keep-E/K
PScript-P Z:high/low-Z/z canal-C g-rules(rnd/seq)-G
state:rev-e rnd/Ham/blk/orig/last-R/H/k/o/∼ sng:pos/neg-5/6 save/load-S/L
graph-g/p, border-B filter-f skip-X pause-N step-x/+ top-T
hide-W no-ops-Q back-q cont-ret:

These options are explained in the rest of thischapter.

Note that some of the options depend on the context, and only appear under the following conditions,

sample:load/keep-E/K . . . not for mixed rules.

Z:high/low-Z/z . . . single networks, and not in TFO-mode.

canal-C . . . not in TFO-mode.

g-rules(rnd/seq)-G . . . if a glider rule sample file exists for the current $\boldsymbol{v}$ and $\boldsymbol{k}$.

PScript-P . . . if the current presentation is 1d or 2d, whatever the native dimension.

graph-g/p . . . **p** appears if a network-graph is active, to save the current network-graph space-time pattern to vector PostScript.

32.14.1 Load/keep the rule sample

sample:load/keep-E/K . . . enter **E** in section 32.14 to load, display and probe a sample of automatically classified as rule-space (a `*.sta` file) as described in section 32.5.2. Chapter 33explains more.
Enter **K** to display and probe a sample that has already been loaded.

32.14.2 revising rule/net

The following rule and network options in section 32.14, are explained below,

rule:save/load/revise/rnd/trans-s/l/v/r/t
net-n
canal-C ...(*not in TFO-mode)*
Z:higher/lower-Z/z ...(*single rule networks only)*

option . . . *what it means*

save/load-s/l . . . enter **s** or **l** in section 32.14 to save or load the rule as a `.rul` file (see Filing, chapter 35) For mixed rule networks this will apply to the rule at cell index 0 only.

revise-v . . . enter **v** in section 32.14 to revise the rule. For mixed rule networks the following prompt is presented below the top-right window to select the cell index,

enter cell index, 149-0: *(for example)*

Then the rule is selected in a lower right window as described in chapter 16.

rnd-r ... enter **r** in section 32.14 to reset a new random rule, or all rules at random in a rule mix.

trans-t ... enter **t** in section 32.14 to transform the rule as described in chapter 18. For mixed rule networks this will apply to the rule at cell index 0 only (section 18.5), but canalizing inputs can be set for the whole network (see chapter 15).

net-n ... enter **n** in section 32.14 for the many network architecture options described in chapter 17, including viewing, revising and filing, the Derrida plot (chapter 22, and learning (chapter 34).

canal-C ... enter **C** in section 32.14 to change the canalizing inputs for the rule in a single rule network (section 18.5) (except the options are presented in a lower right window), or for the whole network in a mixed rule network (chapter 15).

Z:higher/lower-Z/z ... for single rule networks only, enter **Z** in section 32.14 to progressively force the Z-parameter higher, or enter **z** to force it lower, as in sectiona 32.4.3 and 16.3.

32.14.3 space-time patterns on the network-graph

option ... *what it means*

graph-g ... enter **g** in section 32.14 to set up a network-graph (see chapter 20, and to run space-time pattern simultaneously in the usual way and on the network-graph, according to the graph layout and node sizes. This allows considerable presentation flexibility. Section 32.15 gives further details. Enter **g** to deactivate graph space-time patterns if active.

graph-p ... (*if the network-graph is active*)
enter **p** to save a snapshot of the current graph (see section 32.15.1 and chapter 36).

32.14.4 capturing space-time patterns as a vector PostScript file

option ... *what it means*

PScript-P ... enter **P** in section 32.14 to capture a 1d space-time pattern, or a snapshot of a 2d space-time pattern, irrespective of the the native dimension. Chapter 36 gives further details.

32.14.5 revising the seed

The following network state (seed) options in section 32.14, are explained below,

state: rev-e rnd/Ham/blk/orig-R/H/k/o sng:pos/neg-5/6 save/load-S/L

option . . . *what it means*

rev-e . . . enter **e** in section 32.14 to revise the seed. The seed options, described in chapter are 21, are presented in a lower center window.

rand-R . . . enter **R** in section 32.14 for a new random seed.

Ham-H . . . enter **H** in section 32.14 for a new random seed with a set number of cells ($\boldsymbol{H}$) that differ from the original seed, distributed at random. For binary networks ($\boldsymbol{v} = \mathbf{2}$), $\boldsymbol{H}$ is known as the Hamming distance. For $\boldsymbol{v} > \mathbf{2}$ the $\boldsymbol{H}$ cells will be assigned values at random that differ from the original value. The following top-right prompt is presented,

enter % ramdom Hamminig change to current state:

blk-k . . . enter **k** in section 32.14 for a central random central block of cells. The rest can be kept as is, or set to 0. The block size was originally specified in section 21.3 but can be changed here, and this will remain the block size for a on-the-fly block set with **v** The following top-right prompt is presented,

random block: change size (now 30), rest: keep-k, all 0s-(def):

orig-o . . . enter **o** in section 32.14 to restore the original seed.

sng:pos/neg-5/6 . . . enter **5** in section 32.14 for a positive singleton seed, a central cell set to 1, the remainder set to 0. Enter **6** for a negative singleton seed.

save-S . . . enter **S** in section 32.14 to save the seed or the current state. The top-right following prompt is presented,

save: original seed-s, current state-c:

This will be saved as a `.eed` file, (see Filing, chapter 35).

load-L. . . in section 32.14 enter **L** to load a new seed, as a `.eed` file, (see Filing, chapter 35).

32.14.6 miscellaneous pause options

The following miscellaneous pause options in section 32.14, are explained below,

filter-f skip-X pause-N step-x/+ top-T no-ops-Q back-q cont-ret:
(**filter-f** *single rule networks only)*

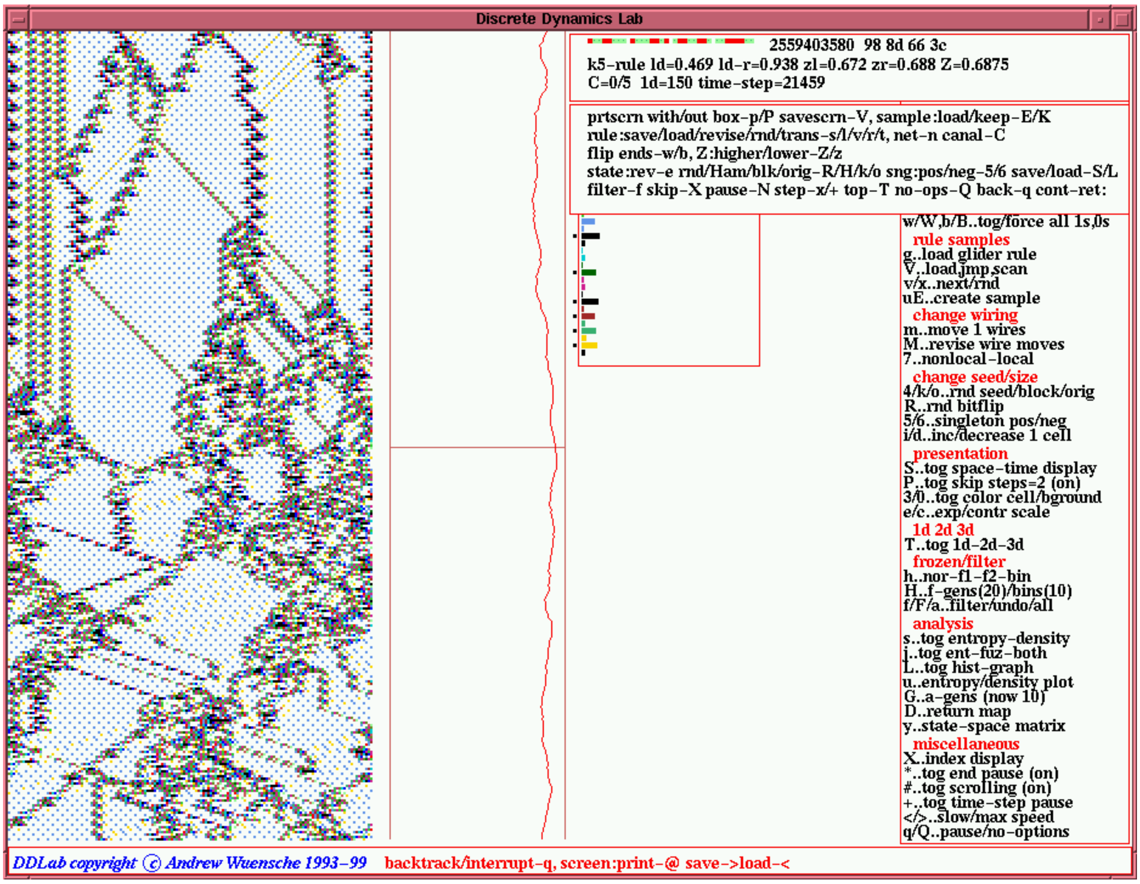
32.14.7 finer control of filtering

filter-f: for a single rule network only, enter **f** in section 32.14 for finer control of filtering than in section 32.10.5. The following top-right prompt is presented (for example, for the k=5 CA 36 0a 96 f9 filtered twice in section 32.10.5),

filtered: 21 10 (*if unfiltered* **none** *shown here*)
filter/undo enter +/- k-index (31-0), max-m reset-r:

The top numbers (21 10) indicate the neighbourhood indices that are currently filtered, otherwise **none** is indicated. Specific neighbourhoods can be filtered and unfiltered. For example, to filter neighbourhood index 26 enter **+26**, to unfilter neighbourhood index 21 enter **-21**. Enter **m** to filter the current most frequent unsuppressed neighborhood. Enter **r** to reset, i.e. unfilter all neighborhoods. Any change is immediately indicated in the top line of the prompt. Enter **return** to accept. On resuming space-time patterns the changes will be applied.

32.14.8 Skipping time-steps



```
Discrete Dynamics Lab

2559403580  98 8d 66 3c
k5-rule ld=0.469 ld-r=0.938 zl=0.672 zr=0.688 Z=0.6875
C=0/5  1d=150 time-step=21459

prtscrn with/out box-p/P savescrn-V, sample:load/keep-E/K
rule:save/load/revise/rnd/trans-s/l/v/r/t, net-n canal-C
flip ends-w/b, Z:higher/lower-Z/z
state:rev-e rnd/Ham/blk/orig-R/H/k/o sng:pos/neg-5/6 save/load-S/L
filter-f skip-X pause-N step-x/+ top-T no-ops-Q back-q cont-ret:

w/W,b/B..tog/force all 1s,0s
  rule samples
g..load glider rule
V..load,jmp,scan
v/x..next/rnd
uE..create sample
  change wiring
m..move 1 wires
M..revise wire moves
7..nonlocal-local
  change seed/size
4/k/o..rnd seed/block/orig
R..rnd bitflip
5/6..singleton pos/neg
i/d..inc/decrease 1 cell
  presentation
S..tog space-time display
P..tog skip steps=2 (on)
3/0..tog color cell/bground
e/c..exp/contr scale
  1d 2d 3d
T..tog 1d-2d-3d
  frozen/filter
h..nor-f1-f2-bin
H..f-gens(20)/bins(10)
f/F/a..filter/undo/all
  analysis
s..tog entropy-density
j..tog ent-fuz-both
L..tog hist-graph
u..entropy/density plot
G..a-gens (now 10)
D..return map
y..state-space matrix
  miscellaneous
X..index display
*..tog end pause (on)
#..tog scrolling (on)
+..tog time-step pause
</>..slow/max speed
q/Q..pause/no-options

DDLab copyright © Andrew Wuensche 1993-99   backtrack/interrupt-q, screen:print-@ save->load-<
```

Figure 32.16: Skipping time-steps in a 1d CA, n=150, k=5 rule 98 8d 66 3c. The time-step skip was set to 2, and activated at about the mid point of the space-time pattern. The space-time pattern was also filtered and expanded to 2 pixels.

skip-X: enter **X** in section 32.14 to change the number of time-steps to be skipped with on-the-fly option

P..tog skip steps=1 (off) *(for example)*

in section 32.8.2. The default is **steps=1**, i.e. showing every second time-step. If this is changed to **steps=2**, every third time-step will be shown, and so forth. Figure 32.16 gives an example.

32.14.9 Set time-step pause

pause-N: enter **N** in section 32.14 to set a pause in the space-time pattern after a preset number time-steps. The following top-right prompt is presented,

pause: screen full-p, after f-gens(20)-f
or enter time-step (no pause-def):

Enter **p** to set the end pause *on*, so a pause allways occurs when the screen is full, as in section 32.12.2 (only for 1d space-time patterns if not scrolling, and 2d+time), or **return** set the end pause *off*.

Enter **f** for a one time pause after the frozen generation number of time-steps set in section 32.10.3 (default 20), or enter a number for a one time pause after that number of time-steps.

32.14.10 Step through blocks of time-steps

step-x . . . enter **x** (repeatedly) in section 32.14 to step through space-time patterns in blocks of time-steps separated by a pause. The block size is the number of time-steps that were set in section 32.14.9 above. (default 20).

step-+ . . . enter + (the plus key) in section 32.14 to pause after each time-step as in section 32.12.4 (this turns scrolling off). The following top-right prompt is presented,

time-step 56, next-ret, reset count-r, end pause -q:
(values shown are examples)

Enter **return** for the next time-step, **r** to reset the time-step count to 1, or **q** to end the time-step pause.

32.14.11 Start time-steps at top

top-T: enter **T** in section 32.14 to continue 1d (and 2d+time) space-time patterns starting at the top of the screen. This also resets the time-step count.

32.14.12 Quit and further options

back-q: enter **q** in section 32.14 to quit space-time patterns and backtrack. The following top-left prompt is first presented,

graphics-g image:prt/save/load-p/s/l ops-o
seed-e net-n back-q cont-ret:

Enter **q** to backtrack to previous options, **return** to continue the space-time pattern, or select an option below,

option . . . *what it means*

graphics-g . . . enter **g** to alter the graphics setup described in section 6.3.

image:prt/save/load
-p/s/l . . . enter **p**, **s** or **l** to print, save, or load the DDLab screen image as described in section 5.6, but without the top-left prompt window in this section.
Printing, saving, and loading the DDLab screen can also be done in the usual way (but showing the screen as is), by entering the following at any time when the prompt cursor is flashing (see section 5.5),

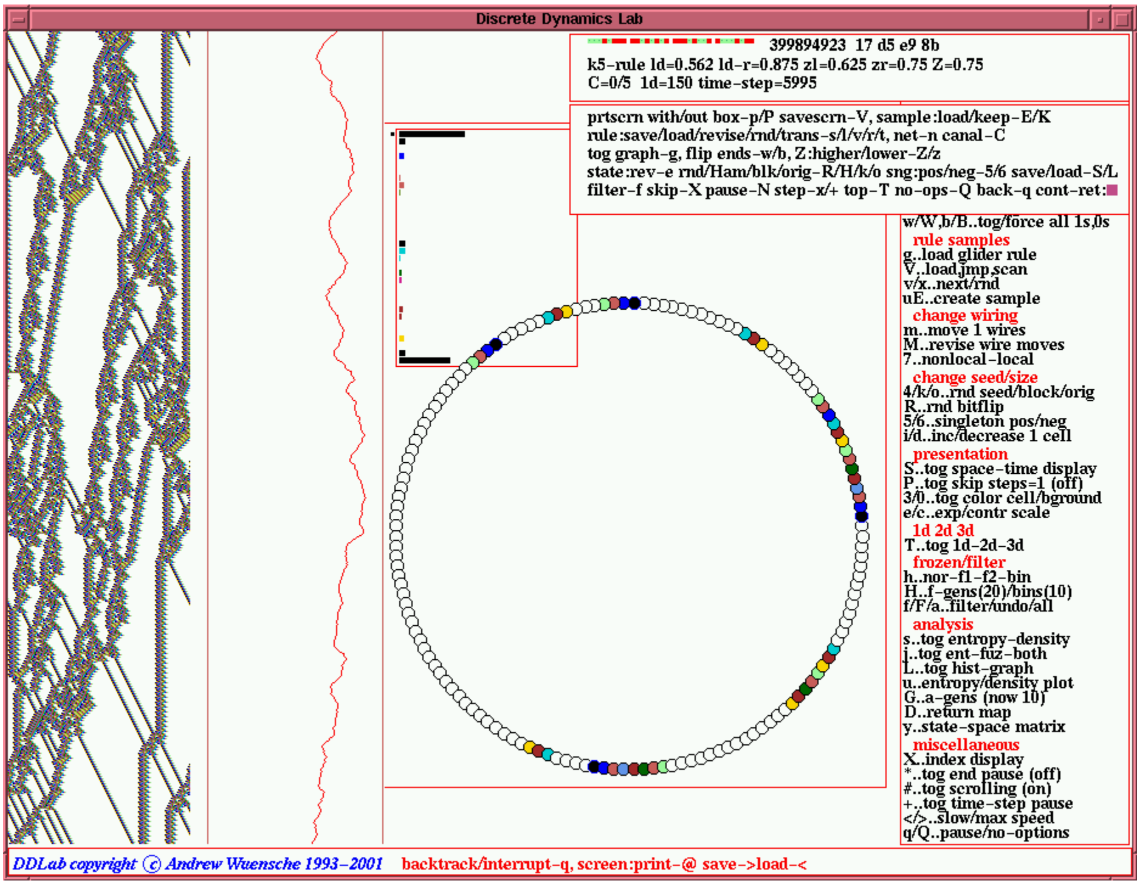
@ - to print
> - to save
< - to load

ops-o . . . enter **o** to revert to the rule and further space-time pattern options in section 32.13.

seed-e . . . enter **e** to revise the seed. The seed options described in chapter are 21 presented in a lower central window, a repeat of **seed:rev-e** in section 32.14.5.

net-n . . . enter **n** for the many network architecture options described in chapter 17, including viewing, revising and filing, the Derrida plot (chapter 22, and learning (chapter 34), a repeat of **net-n** in section 32.14.2).

32.15 Graph layout of space-time patterns

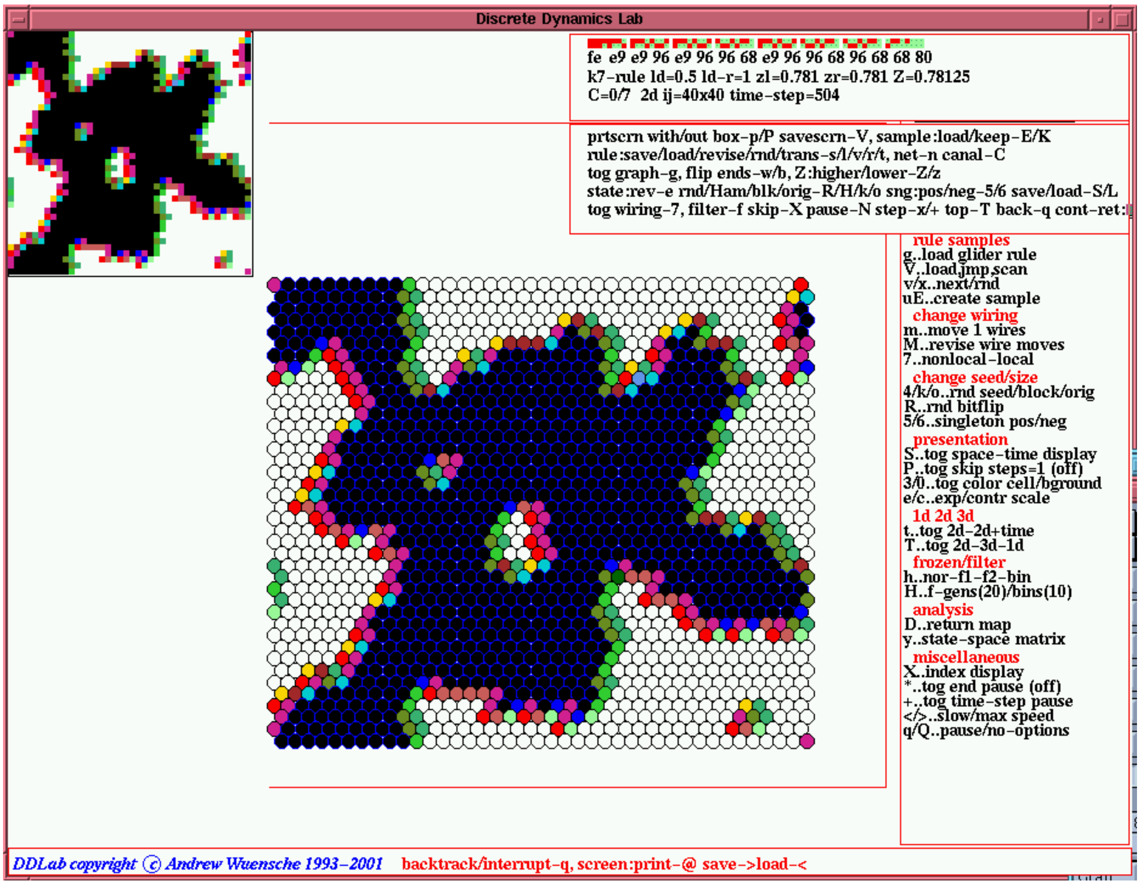


```
399894923  17 d5 e9 8b
k5-rule ld=0.562 ld-r=0.875 zl=0.625 zr=0.75 Z=0.75
C=0/5  1d=150 time-step=5995
```

```
prtscrn with/out box-p/P savescrn-V, sample:load/keep-E/K
rule:save/load/revise/rnd/trans-s/l/v/r/t, net-n canal-C
tog graph-g, flip ends-w/b, Z:higher/lower-Z/z
state:rev-e rnd/Ham/blk/orig-R/H/k/o sng:pos/neg-5/6 save/load-S/L
filter-f skip-X pause-N step-x/+ top-T no-ops-Q back-q cont-ret:
```

DDLab copyright © Andrew Wuensche 1993–2001 backtrack/interrupt-q, screen:print-@ save->load-<

Figure 32.17: 1d CA space-time patterns shown according to the current network-graph layout, in this case the default circle layout which reflects the actual geometry of a 1d CA with periodic boundary conditions. This is a complex 1d CA, n=150, k=5, rule 17d5e98b, with the background filtered. The normal space-time patterns on the left of the screen, and on-the-fly options, continue to function.



```
fe  e9 e9 96 e9 96 96 68 e9 96 96 68 96 68 68 80
k7-rule ld=0.5 ld-r=1 zl=0.781 zr=0.781 Z=0.78125
C=0/7  2d ij=40x40 time-step=504

prtscrn with/out box-p/P savescrn-V, sample:load/keep-E/K
rule:save/load/revise/rnd/trans-s/l/v/r/t, net-n canal-C
tog graph-g, flip ends-w/b, Z:higher/lower-Z/z
state:rev-e rnd/Ham/blk/orig-R/H/k/o sng:pos/neg-5/6 save/load-S/L
tog wiring-7, filter-f skip-X pause-N step-x/+ top-T back-q cont-ret:
```

Figure 32.18: 2d CA space-time patterns shown according to the current network-graph layout, in this case the default hexagonal grid for k=6 or 7. The network is 40x40, k=7, with a modified majority rule, totalistic code 232. The normal 2d space-time patterns, to left of the screen, and on-the-fly options, continue to function.

Space-time patterns can be shown according to the network-graph layout (see chapter 20) in addition to the normal presentation described in this chapter and chapter 31. This allows enormous flexibility, as the network-graph has a number of default layouts, circle, spiral, 1d, 2d, 3d, as well as allowing dragging nodes and components, and many other options for rearranging the graph.

Enter **g** in section 32.14. The following top-right option is first displayed,

show links-L (avoid for large networks), layout only-def:

Enter **return** for the layout only, without links, which is much faster and more economical with memory, and may be essential for large (2d or 3d) networks. This option only appears if space-time patterns are interrupted. However, the links can be shown in the normal way by entering **L**. The network-graph appears in a large central window, but for a layout only, the options in sections 20.2, 20.4, and 20.5 are abbreviated to omit link options.

In this case the initial top-right options (compare with section 20.2) are as follows,

NETWORK-graph (no links): drag-(def)
settings-S rotate-x/X flip-h/v, exp/contr: nodes-e/c links-E/C both-B/b
window-w
layout: file-f circle/spiral-o/O 1d/2d/3d-1/2/3 rnd-r/R quit-q:

The top-right options for dragging nodes and fragments (compare with section 20.5) are as follows (for example),

in-out node 0: drag/release - left mouse button node 23
not active?-click right button first, rotate-x/X flip-h/v
block-B
single-s all-a exit-q:

Rearrange the graph and the node sizes by the methods described in chapter 20. On exiting the graph routine, space-time patters will be shown according to the final graph layout in a large central window. The normal space-time patterns, and on-the-fly options continue to work. Note that colors will correspond to the normal space-time patterns. Some examples of are shown in figure 32.17-32.18.

32.15.1 Graph snapshot as vector PostScript

If the graph space-time pattern is active, the current snapshot can be saved to a vector PostScript file. Enter **P** in pause options (section 32.14). The following top-right prompt is displayed,

save graph snapshot to postScript: abort-q greyscale-P color-def:

Enter **return** for color, **P** for greyscale (to toggle the cell outline enter **@** on-the-fly beforehand). The top-right filing prompt (section 35.2) will be displayed to select the filename (default `my_sgPS.ps`).

Chapter 33

Classifying rule space

This chapter describes methods of automatically classifying CA rule-space[27] to distinguish ordered, complex and chaotic rules, including creating, displaying, sorting, analysing and probing the sample. Unsorted and sorted 1d CA k=5 sample files, `test5.sta` and `test5ss.sta`, will be used as examples.Three larger sample files (for k=5, 6 and 7, `five5ss.sta`, `six5ss.sta` and `sev5ss.sta`) will be used to illustrate the classification. The files are included with DDLab. As well as showing the distribution of rule classes in rule-space, the method (illustrated in figure 2.6) is able to identify rules that support interacting gliders and related complex dynamics[27].

The method works by generating rules at random, then keeping track of the input-entropy for a moving window of x time-steps (default x=10, set in section 32.11.5), i.e. at each time-step the input-entropy is taken on the preceding block of x time-steps including the current time-step. To automatically classifying rule-space, the mean input-entropy is plotted against the standard deviation of the input-entropy for successive random rules, were the position of the rule on the plot indicates the rule class.

33.1 Setting parameters

To automatically create a rule sample, or run a test, first set up any CA with the desired parameters of n and k, for example a 1d CA n=150 and k=5, and run its space-time patterns (see chapter 32). For 1d scrolling should be set off in section 32.12.3. The input-entropy plot should be active, the default for 1d, but not for 2d and 3d (see section 31.10 and 32.11.1).

While the space-time patterns are iterating, select on-the-fly option **uE..create sample**, enter **u** followed by **E**. The initial **u** initiates an entropy-density plot (section 32.11.4). If this plot is current, the on-the-fly prompt is **E..create sample**, so just enter **E**. The following prompts are presented in turn in a top-right window (enter **q** to backtrack or exit),

sample: automatic-a, test-def:
specify start - end time-step of entropy record
start (def 30): end (def 430): sample size for each rule (def 5):

option . . . *what it means*

automatic-a . . . Enter **a** to create an automatic sample (section 33.3).

test-def . . . Enter **return** for a test (section 33.2), showing the measures and the entropy/density scatter plot for various rules.

start (def 30): . . . Enter the time-step when measures should start. By not including the initial time-steps, the dynamics is allowed to settle into typical behaviour before measures are recorded.

end (def 430): . . . Enter the end time-step when space-time patterns stop and measures end. The measures are recorded over *end-start+1* time-steps.

33.2 Running a test

If **return** was entered in section 33.1, a test run will be made on the current rule, for example as illustrated in figure 33.1. Space-time patterns will be run from a number of initial states (set in section 32.11.5) for the number of time-steps set in section 33.1. After each run, various measures including the varience and standard deviation of the input-entropy are shown in a top-right window. At the same time the entropy/density scatter plot (section 32.11.4) is drawn in a lower center window. The plot shows distinctive signatures for complex rules, and is drawn in alternating colors for successive runs. When the runs are complete, the following top-right window presents data and options (for example),

run=5/5 k5-rule=(dec)3181533384 (hex)bda258c8
mean=0.618633, average dev=0.159212, standard dev=0.182426
varience=0.042649, skewness=-0.082958, kurtosis=-0.861457
automatic-a, next rule: glider-g same-s random-def:

The top line shows the current rule in decimal (for $\boldsymbol{k} \leq \mathbf{5}$) and in hex (for $\boldsymbol{k} \leq \mathbf{7}$). If single cell input-entropy was set in section 31.11 the cell index will be indicated, for example **single=22**. The next 2 lines show the various measures. The last line shows prompts as follows,

option . . . *what it means*

automatic-a . . . Enter **a** to create an automatic sample (section 33.3).

next rule: glider-g . . . Enter **g** to test a random glider rule (see section 32.5.1).

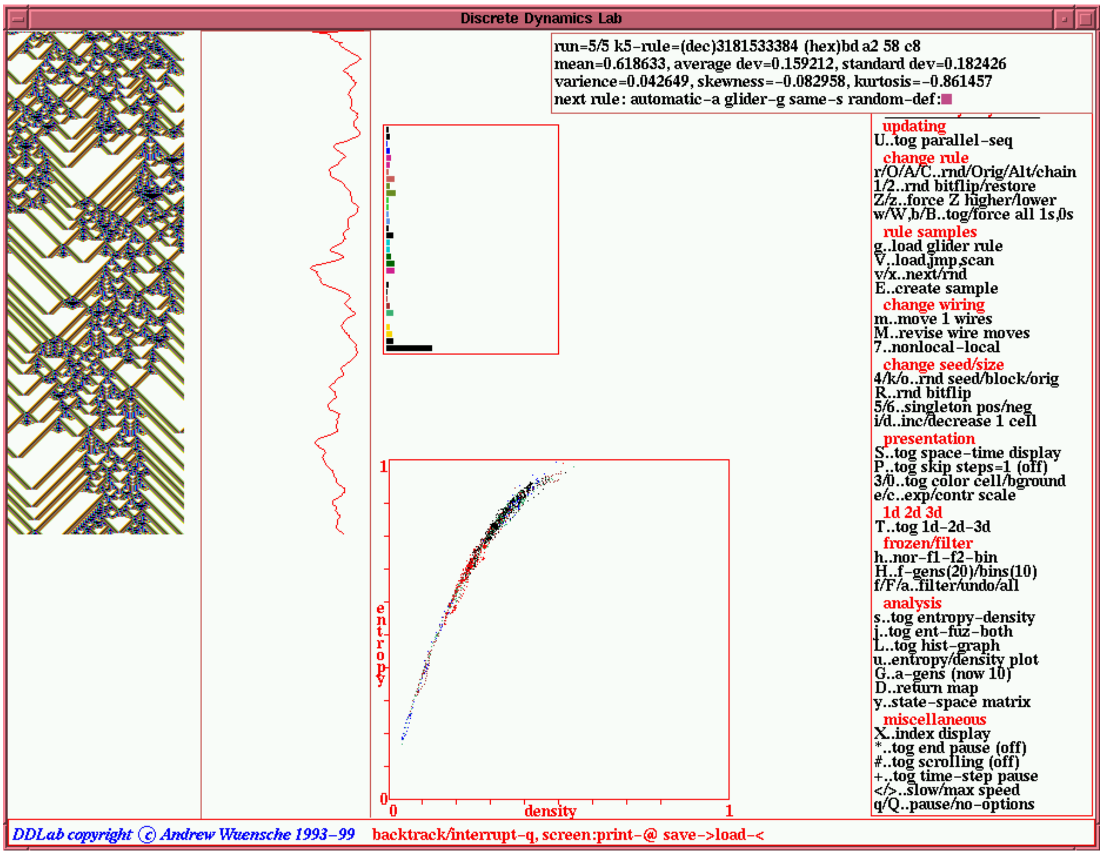
same-s . . . Enter **s** to re-test the same rule.

random-def . . . Enter **return** to test a random rule.

33.3 Creating a rule sample

If **a** is entered in sections 33.1 or 33.2, a file name for the sample (`*.sta`) will first be selected in a top-right window (see Filing, chapter 35), followed by a top-right prompt to append the data to an existing file, or start a new file ,

my_sta.sta: append data-a, new file-def, add + to pause: *(for example)*



```
run=5/5 k5-rule=(dec)3181533384 (hex)bd a2 58 c8
mean=0.618633, average dev=0.159212, standard dev=0.182426
varience=0.042649, skewness=-0.082958, kurtosis=-0.861457
next rule: automatic-a glider-g same-s random-def:
```

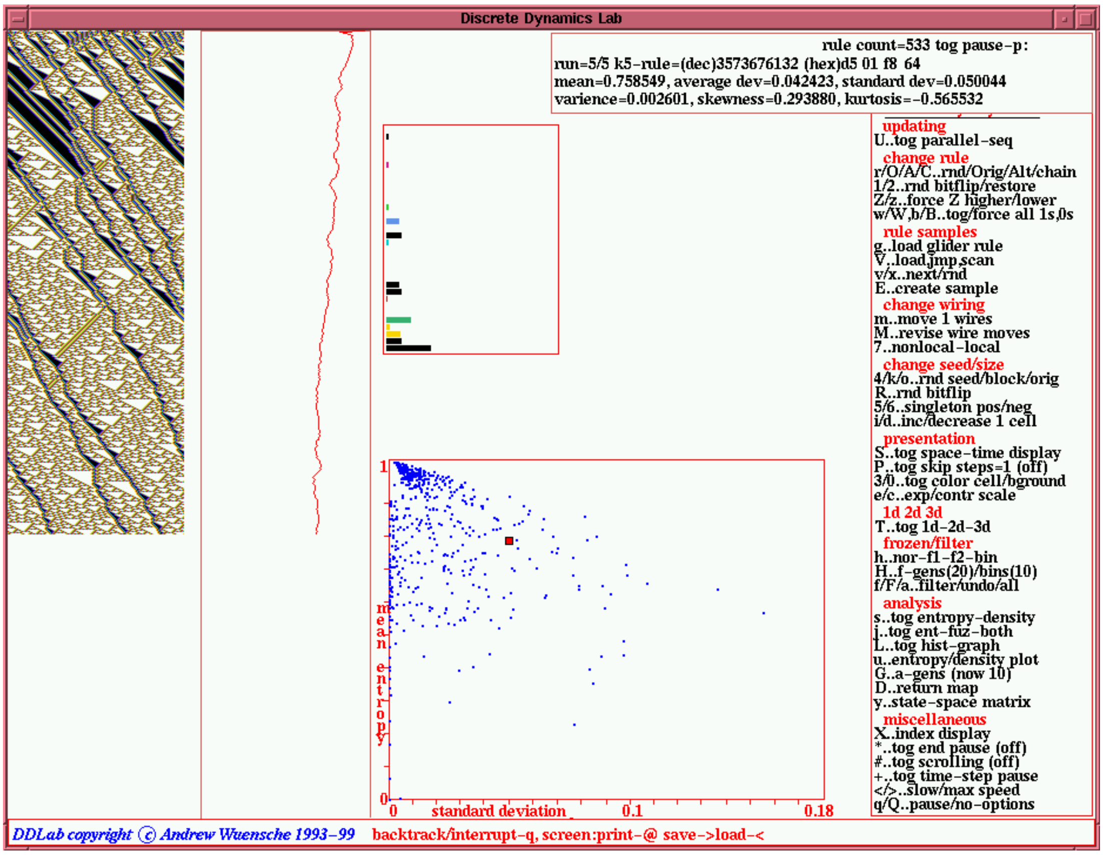
Figure 33.1: Running a test with various measures including varience and standard deviation of the input-entropy, show in a top-right window. This example for a 1d CA, k=5 rule is bd a2 58 c8, n=150. At the same time the entropy/density scatter plot (section 32.11.4) is drawn in a lower center window. The plot shows distinctive signatures for complex rules, and is drawn in alternating colors for successive runs. The default parameters (start 30, end 430, size 5) were used, meaning the measures were started at time-step 30 and ended at 430, i.e. they were taken over 401 time-steps, and the number of runs from different random initial states was 5.

To pause after each new rule, enter or add +, i.e. **a+** to append and pause. The automatic sample will then start, plotting the mean entropy against the standard deviation of the entropy, one point for each new random rule in the scatter plot (see figure 33.2). After the set of runs for each rule is complete, a small red square is plotted. This is replaced by a blue dot when the next rule is complete, and so on.

For each run the same data as in section 33.2 will appear in a top-right window, but including a count of the rule so far, and an option to toggle the pause, both on-the-fly and after each rule is complete, for example

rule count=1672 tog pause-p *(as in figure 33.2)*

If the pause is set, enter **return** (or **p** to toggle the pause on/off) to continue with the next rule. The sample will continue indefinitely until paused with **p**, or interrupted with **q** (enter **q**



```
rule count=533 tog pause-p:
run=5/5 k5-rule=(dec)3573676132 (hex)d5 01 f8 64
mean=0.758549, average dev=0.042423, standard dev=0.050044
varience=0.002601, skewness=0.293880, kurtosis=-0.565532
```

Figure 33.2: Creating an automatic sample. Various measures including varience and standard deviation of the input-entropy, and the rule count, are shown in a top-right window. The sample is for a 1d CA, k=5, n=150. At the same time the mean entropy - standard deviation scatter plot is drawn in a lower center window. After the runs for each rule are complete, a small red square is plotted. This is replace by a blue dot when the next rule is complete, and so on. Note that any standard deviation above 0.18 is rounded down to 0.18

again to backtrack). If interrupted, the sample can always be continued later by appending a new sample run to an old file name.

To considerably speed up computation, turn off the space-time graphics with on-the-fly toggle **S** (section 32.8.1). Note that most on-the-fly options such as "presentation" (section 32.8) and "frozen/filter" (section 32.10) generaly work while the sample is in progress, but other on-the-fly options should be used with caution as the the consequenses are unpredictable.

33.4 Loading, sorting and displaying a sample

To load an automatic sample, enter on-the-fly option **V..load,jmp,scan**, or when interrupting space-time patterns (section 32.13) enter **E** (or **K** to "keep" the sample if previously loaded) see section 32.14.1. Once the file name is selected (see Filing, chapter 35), the following top-right prompt is presented,

list: all-l pos+p, sort-s quit-q
entropy standard dev plot: Z-Z lda-L ent-def smalldots+d:

option ...	*what it means*
list: all-l ...	enter **l** to list the sample, and select a rule (sections 33.5 - 33.5.2)).
+p ...	enter **lp** to list as above, adjacent to a given mean entropy and standard deviation coordinate(section 33.5.1).
sort-s ...	enter **s** to sort the sample (below in this section).
quit-q ...	enter **q** to quit the sample and backtrack.
standard dev plot: ...	*plot (and probe) the sample in various ways, section 33.6 - - 33.6.1).*
Z-Z ...	enter **Z** to plot standard deviation against the Z parameter.
lda-L ...	enter **L** to plot standard deviation against the $\boldsymbol{\lambda}$ parameter.
ent-(def) ...	enter **return** to plot standard deviation against mean entropy.
smalldots+d ...	enter **d** (or **Zd** or **Ld**) for the above plots, but with smaller dots. For a low resolution DDLab screen (less than 640 pixels wide), the default is small dots, so the prompt reads **bigdots+d**.

Enter **l** to list the sample, or or **lp** to list just the rules at (or adjacent to) a given position on the mean entropy - standard deviation scatter plot. These option are explained in section 33.5.

Enter **s** to sort an unsorted sample. A prompt for a new file name (`*.sta`) for the sorted file will be presented (see Filing, chapter 35). The rules are sorted by decreasing standard deviation, then by decreasing mean entropy for each standard deviation. For a large sample this might take a little time.

Enter **return** or **d** to show the mean entropy - standard deviation plot (**d** shows the plot with smaller dots). Further options include probing the plot with the mouse and listing/selecting rules, (see section 33.6.1), and a 2d frequency histogram, where the vertical axis represents rule frequency. Enter **Z** or **L** for alternative plots of the Z-parameter or $\boldsymbol{\lambda}$-parameter against standard deviation. (**+d**, i.e. **Zd** or **Ld**, show the plots with smaller dots). These plots include the same further options.

33.5 Listing a sample

If **l** is entered in section 33.4 a sorted or unsorted list of the rules in the sample is presented in a top-right window (see figure 33.3). For an unsorted file, the list is in the order saved.

The top line **533 k=5 rules   m-ent stan-dev** shows the number of rules in the k=5 rule sample. The rules are then listed, as many as will fit in the window, with a rule index list on the left, the rule in hex, followed by the mean entropy and standard deviation. Sets of rules separated by lines have same standard deviation. The following prompt is show at the bottom of the list,

printbox-p
more-ret jump-j quit/select-q:

option ...	*what it means*
printbox-p ...	enter **p** to print just the list window (DOS only), or the full screen in Unix and Linux (see section 5.6)

more-ret . . . if there are more rules than will fitt in the window, enter **m** to see the next batch

jump-j . . . enter **j** to junp to a new list number. The following extra prompt is presented,

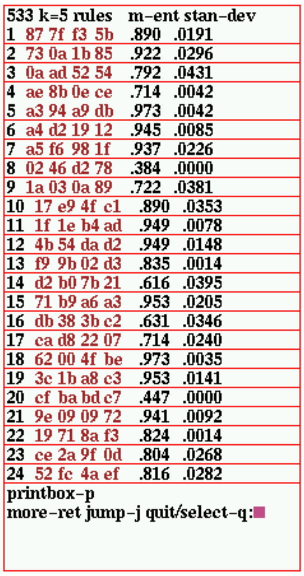
> **enter rule index 1-533:**
> **jump-j quit/select-q more-ret:**

Enter the rule index which becomes the first on the list.

quit/select-q . . . if **q** is entered, the following extra prompt is presented,

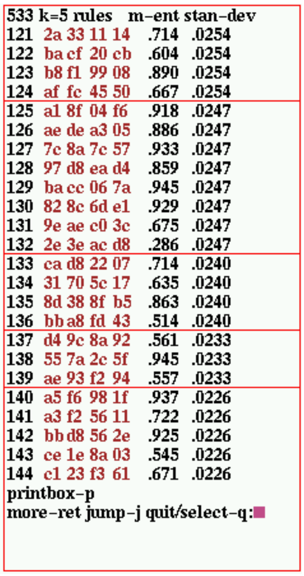
> **part list, quit-q select-s:**

This is explained in sectio 33.5.2.



```
533 k=5 rules   m-ent stan-dev
1  87 7f f3 5b  .890  .0191
2  73 0a 1b 85  .922  .0296
3  0a ad 52 54  .792  .0431
4  ae 8b 0e ce  .714  .0042
5  a3 94 a9 db  .973  .0042
6  a4 d2 19 12  .945  .0085
7  a5 f6 98 1f  .937  .0226
8  02 46 d2 78  .384  .0000
9  1a 03 0a 89  .722  .0381
10  17 e9 4f c1  .890  .0353
11  1f 1e b4 ad  .949  .0078
12  4b 54 da d2  .949  .0148
13  f9 9b 02 d3  .835  .0014
14  d2 b0 7b 21  .616  .0395
15  71 b9 a6 a3  .953  .0205
16  db 38 3b c2  .631  .0346
17  ca d8 22 07  .714  .0240
18  62 00 4f be  .973  .0035
19  3c 1b a8 c3  .953  .0141
20  cf ba bd c7  .447  .0000
21  9e 09 09 72  .941  .0092
22  19 71 8a f3  .824  .0014
23  ce 2a 9f 0d  .804  .0268
24  52 fc 4a ef  .816  .0282
printbox-p
more-ret jump-j quit/select-q:■
```

unsorted



```
533 k=5 rules   m-ent stan-dev
121  2a 33 11 14  .714  .0254
122  ba cf 20 cb  .604  .0254
123  b8 f1 99 08  .890  .0254
124  af fc 45 50  .667  .0254
125  a1 8f 04 f6  .918  .0247
126  ae de a3 05  .886  .0247
127  7c 8a 7c 57  .933  .0247
128  97 d8 ea d4  .859  .0247
129  ba cc 06 7a  .945  .0247
130  82 8c 6d e1  .929  .0247
131  9e ae c0 3c  .675  .0247
132  2e 3e ac d8  .286  .0247
133  ca d8 22 07  .714  .0240
134  31 70 5c 17  .635  .0240
135  8d 38 8f b5  .863  .0240
136  bb a8 fd 43  .514  .0240
137  d4 9c 8a 92  .561  .0233
138  55 7a 2c 5f  .945  .0233
139  ae 93 f2 94  .557  .0233
140  a5 f6 98 1f  .937  .0226
141  a3 f2 56 11  .722  .0226
142  bb d8 56 2e  .925  .0226
143  ce 1e 8a 03  .545  .0226
144  c1 23 f3 61  .671  .0226
printbox-p
more-ret jump-j quit/select-q:■
```

sorted

Figure 33.3: Listing the rule sample in figure 33.2 with 533 k=5 rules. *left*: the unsorted sample shown in the order saved starting from rule index 1 (enter **m** to see more). *right*: the sorted sample starting rule index 121. Sorting is by decreasing standard deviation, then by decreasing mean entropy for each standard deviation. Sets of rules seperated by lines have same standard deviation.

33.5.1 Limiting the list according to plot coordinates

Entering **lp** in section 33.4, allows just the rules at (or adjacent to) a given position on the mean entropy - standard deviation scatter plot to be listed. The following top-right prompts are presented in sequence,

> **mean entropy pos 0-255:** **standard dev pos 0-255:**

The y,x coordinates are plotted according to unsigned char values, so the mean entropy (0 to 1) and standard deviation (0 to 0.18) are in the range 0 to 255. Also any standard deviation above 0.18 is reset to 0.18. Enter a number (0 to 255) in each case to select a position on the plot. The same can be achieved by clicking on the plot with the left mouse buttton in section 33.6. The list will appear (similar to figure 33.3) for example,

533 k=5 pos 250x5 rad=0 *(values shown are examples)*
left mouse button to select *(applies to section 33.6)*
420 94 52 a6 ce   .980   .0035 *(rules at these coordinates)*
420 2a c7 51 b6   .980   .0035

quit-q select-s rad-(max 9):

rad=0 indicates that only rules at the specific coordinate selected are shown, there may be none. **rad-(max 9)** to expand the coordinate area to include more rules, enter a radius from 1 to 9.

33.5.2 Selecting a rule from the list

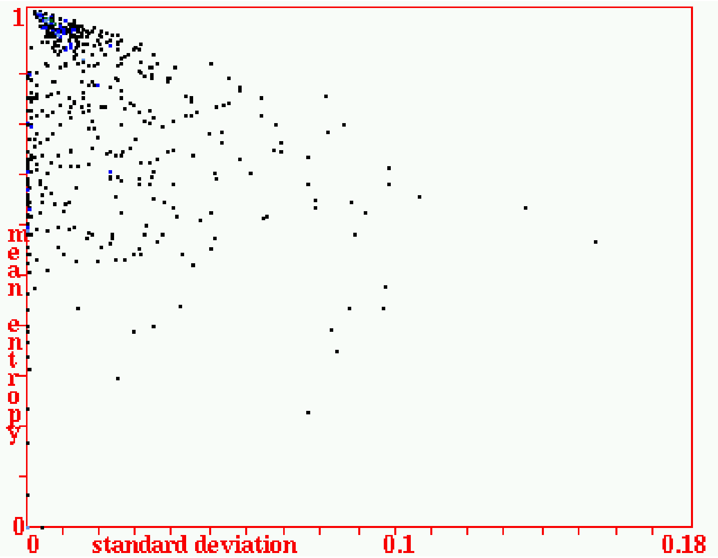
Enter **s** in section 33.5 or 33.5.1 to select a rule from the list. The following extra prompt is presented in the list window,

enter rule index 1-533: *(values shown are examples)*

Enter a rule number. The program reverts to the prompt in section 33.4. Enter **q** to quit, then return to continue with the space-time patterns of the selected rule.

Enter **q** in section 33.5 or 33.5.1 to return to the prompt in section 33.4. Note that the plots which can be selected from this prompt allow allows probing with the mouse, and also listing/selecting rules.

33.6 Plot standard deviation against mean entropy, Z or λ



from the file test5ss.sta

Figure 33.4: The mean entropy plotted against the standard deviation of the entropy. This is the test sample as in figure 33.2 with 533 1d CA k=5 rules. The sample files, unsorted (test5.sta) and sorted (test5ss.sta), are included with DDLab. A color scheme indicates frequency, clearer in figures 33.6.2 to 33.9 with a much larger samples. To select a rule anywhere on the plot, click on a point with the left mouse button to list the rule or rules, then select the rule (see sections 33.5.1 and 33.4).

If **return** is entered in 33.4 the plot of mean entropy - standard deviation is displayed in a lower center window. If **Z** or **L** is entered, alternative plots Z-parameter or $\boldsymbol{\lambda}$-parameter against standard deviation are displayed. The following top-right prompt is presented,

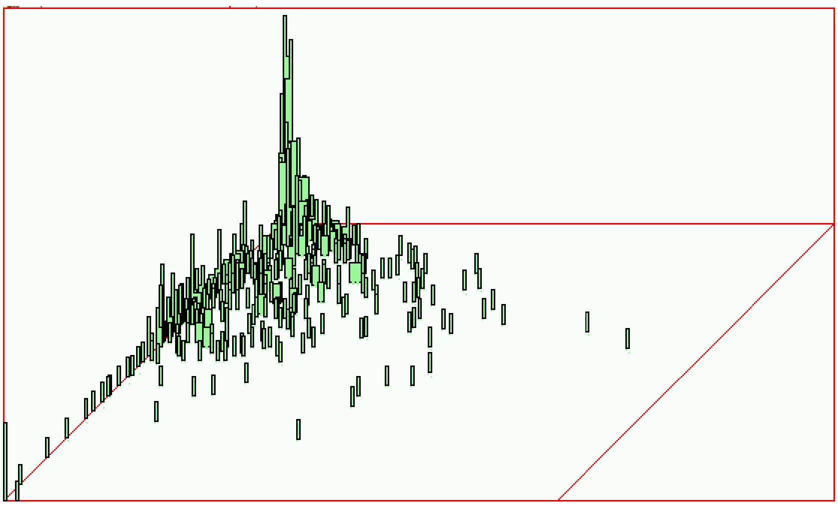
probe/select rules: point-click left mouse buttton
frequency histogram, or quit, -q

33.6.1 Probing the plot with the mouse, and selecting rules

Clicking the left mouse buttton on a point in the plot produces a top right window excatly as described in section 33.5.1, listing rules at that point, if any. The list can be expanded by setting a radius around the point to expand the coordinate area to include more rules. This is also useful to capture an isolated rule if ones aim when clicking is not so accurate. A rule can be selected from the list, and its space-time patterns run, as described in section 33.5.2.

Enter **q** (or click again) to delete the list, and click the plot again at a new position to show a new list.

33.6.2 Showing the plot as a 2d frequency histogram



from the file test5ss.sta

Figure 33.5: The 2d frequency histogram of the mean entropy plotted - standard deviation plot. The vertical axis indicates the frquency of rules falling on each quadrant of a 256x256 grid. This is the sorted test sample (test5ss.sta) as in figure 33.2 and 33.4, with 533 k=5 rules. This plot only works with a sorted sample. See figures 33.6.2 - 33.9 for 2d frequency histograms with a much larger sample.

If **q** is entered in section 33.6 the following top-right prompt is presented,

533 k=5 rules: frequency histogram-f, quit-q, cont-ret:

Enter **q** or **return** to backtrack to the prompt in section 33.4. Enter **f** for the entropy - standard deviation 2d frequency histogram. The following top-right prompts are presented in sequence,

freq hist, subdiv (16,32,64, etc. def=128, max-256):
save data-d, step-s, and show%-S: show freq log-l: show grid-g:

option . . . *what it means*

subdiv (16,32,64, etc. def=128, max-256):
. . . The vertical axis of the 2d histogram indicates the frquency of rules falling on each quadrant of a grid laid over the entropy plotted - standard deviation plot. The resolution of this grid can be 16×16, 32×32, etc. up to 255×255. Enter **return** for the default 128×128, or enter **16,32,64,. . . ,256** for a different resolution.

save data-d . . . Enter **d** to save the height data to a an ascii file (`*.dat`), laid out in rows and columns according to the grid resolution above. For example, for a 16×16 grid, the test5ss.sta sample data (as in figure 33.2 - 33.5,is as follows,

```
125 35 2 0 0 0 0 0 0 0 0 0 0 0 0 0
22 32 19 4 1 0 0 0 0 0 0 0 0 0 0 0
20 11 8 8 2 3 0 1 0 0 0 0 0 0 0 0
17 12 7 5 5 1 1 2 0 0 0 0 0 0 0 0
18 8 8 3 2 2 3 0 1 0 0 0 0 0 0 0
21 1 10 4 2 1 2 0 1 1 0 0 0 0 0 0
24 4 2 3 2 2 1 1 1 0 0 0 1 0 0 0
10 5 9 3 3 0 0 1 0 0 0 0 0 1 0 0
4 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
3 1 1 2 0 0 0 2 1 0 0 0 0 0 0 0
2 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

step-s, and show%-S . . . Enter **s** or **S** to pause after each row in the 2d histogram, to see its structure in greater detail; enter **return** for the next row. If **S** is entered, the height data for successive rows is displayed in a top-right window.

show freq log-l . . . Enter **l** for a log scale of the vertical axis.

show grid-g . . . For a grid of 64×64 or less, enter **g** to overlay the grid on the plotted - standard deviation plot.

The 2d histogram is presented in a large upper left window. Note that this plot only works correctly with a sorted sample file.

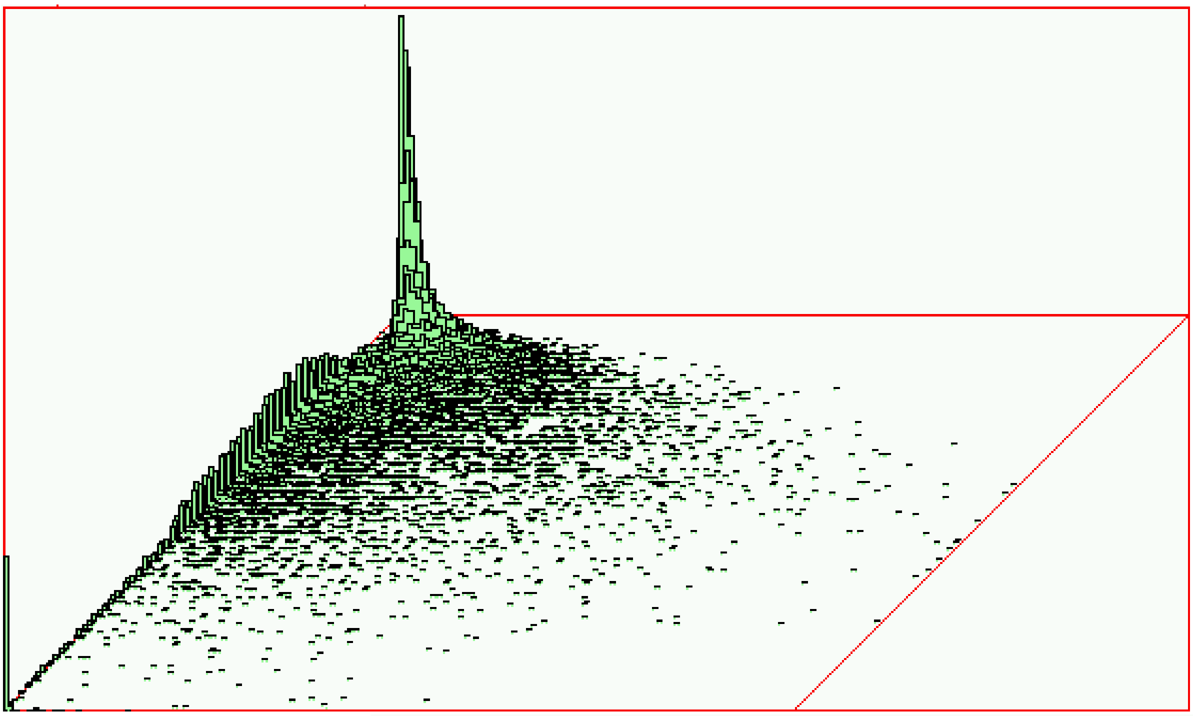
33.7 Samples files included with DDLab

As well as the 1d CA k=5 sample files, `test5.sta` and `test5ss.sta` used in the examples above, three larger sample files for k=5, 6 and 7 (`five5ss.sta`, `six5ss.sta` and `sev5ss.sta`) illustrate the classification of rule-space. The files are included with DDLab.

In these samples, for each random rule, data was gathered from 5 runs from random initial states, for 430 time-steps, discounting the first 30 to allow the system to settle, and made relative to a moving window of 5 time-steps. The three samples are shown in figure 33.6.2 - 33.9.

Looking at the k=5 2d histogram (figure 33.9), the "tower" in the upper left hand corner represents chaotic rules with low standard deviation and high mean entropy. The ridge on the left represents ordered rules with low standard deviation and a spread of lower mean entropy. Complex rules, as in figure 33.7, have higher standard deviation, and are spread out towards the right. There is a low diagonal valley between the tower and the ridge, a boundary between ordered and chaotic rules, but a gradual transition from both towards the complex rules.

Comparing the three plots for k=5,6 and 7 (figures 33.6.2 - 33.9), as k increases there is an increasing frequency of chaotic rules, a declining frequency of complex rules, and a sharp decrease in ordered rules.



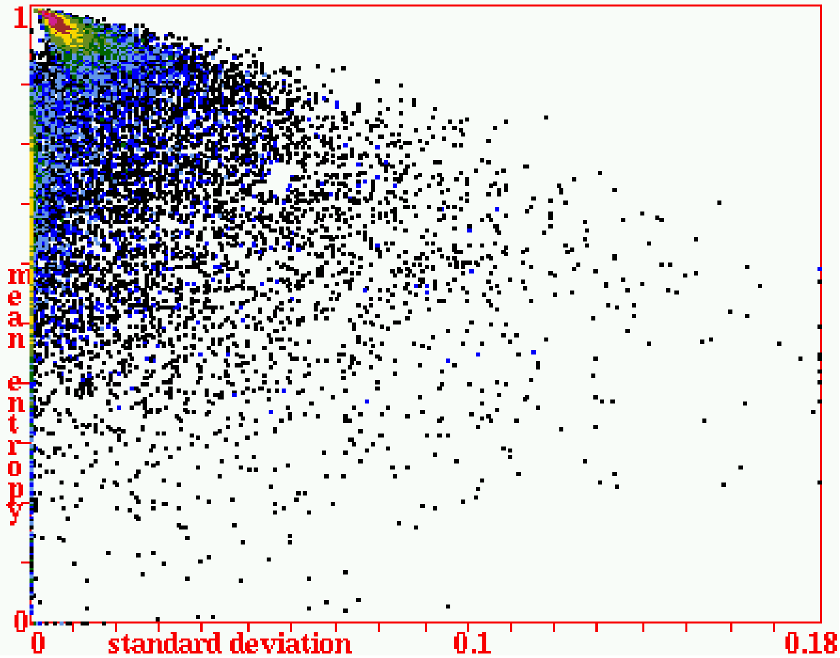


Figure 33.6: *above*: The 2d frequency histogram. The vertical axis indicates the frquency of rules falling on each quadrant of a 256x256 grid. Chaotic rules are in the "tower", ordered rules in the ridge on the left. Complex rules are spread out towards the right. *right*: The mean entropy - standard deviation plot, color coded for frequency. This is the sorted sample `five5ss.sta`, with 17680 1d CA k=5 rules.

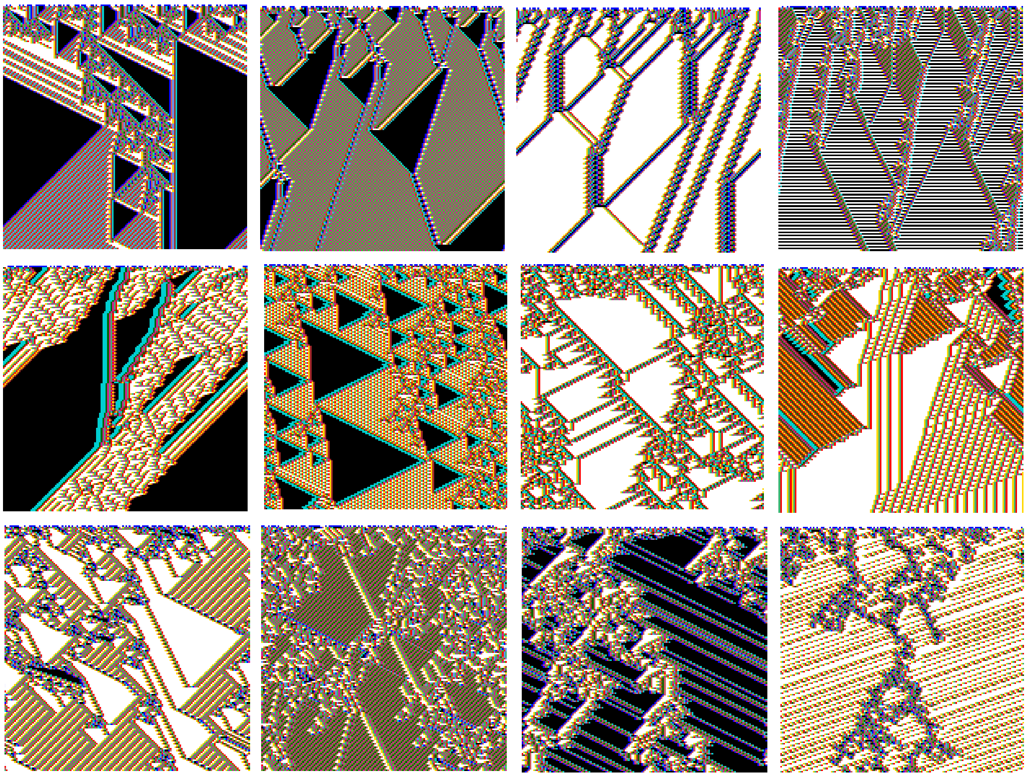


Figure 33.7: Examples of 1d CA complex space-time patterns with high standard deviation, for $\boldsymbol{k}$=5 (*top row*), $\boldsymbol{k}$=6 (*center row*) and $\boldsymbol{k}$=7 (*bottom row*). These examples, taken from the sample files `five5ss.sta`, `six5ss.sta` and `sev5ss.sta`. $\boldsymbol{n}$=150, 150 times-steps from random initial states. Cells are colored according to the neighbourhood.

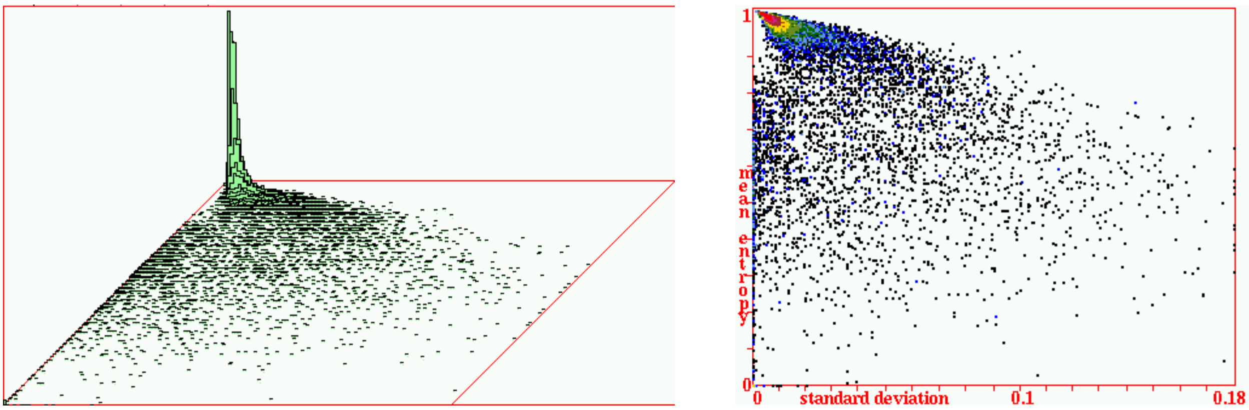


Figure 33.8: The mean entropy - standard deviation plot and histogram for the sorted sample `six5ss.sta`, with 15425 1d CA k=6 rules. Other parameters are the same as in figure 33.6.2

.

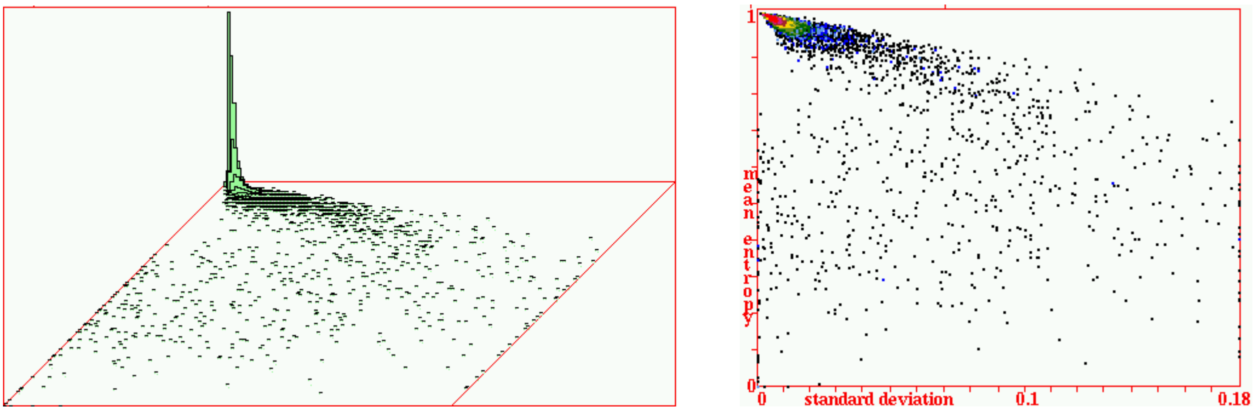


Figure 33.9: The mean entropy - standard deviation plot and histogram for the sorted sample `sev5ss.sta`, with 14221 1d CA k=7 rules. Other parameters are the same as in figure 33.6.2

.

33.8 Rule sample encoding

A `.sta`) rule sample file is a binary file of an unsigned char array, recording (A) each rule, (B) its average entropy, and (C) the standard deviation of the entropy. The file records A,B,C for each rule in the sample. If the array was sorted before being saved (see section 33.4), the rules are ordered by decreasing standard deviation, then by decreasing mean entropy for each standard deviation.

The encoding of A,B,C is as follows: A (the rule-table) consists of $\mathbf{2^k}$ bytes (minimum 1 byte), B and C are both normalised to a number 0-255 and stored as one byte.

Chapter 34

Learning, forgetting, and highlighting

For rcode only

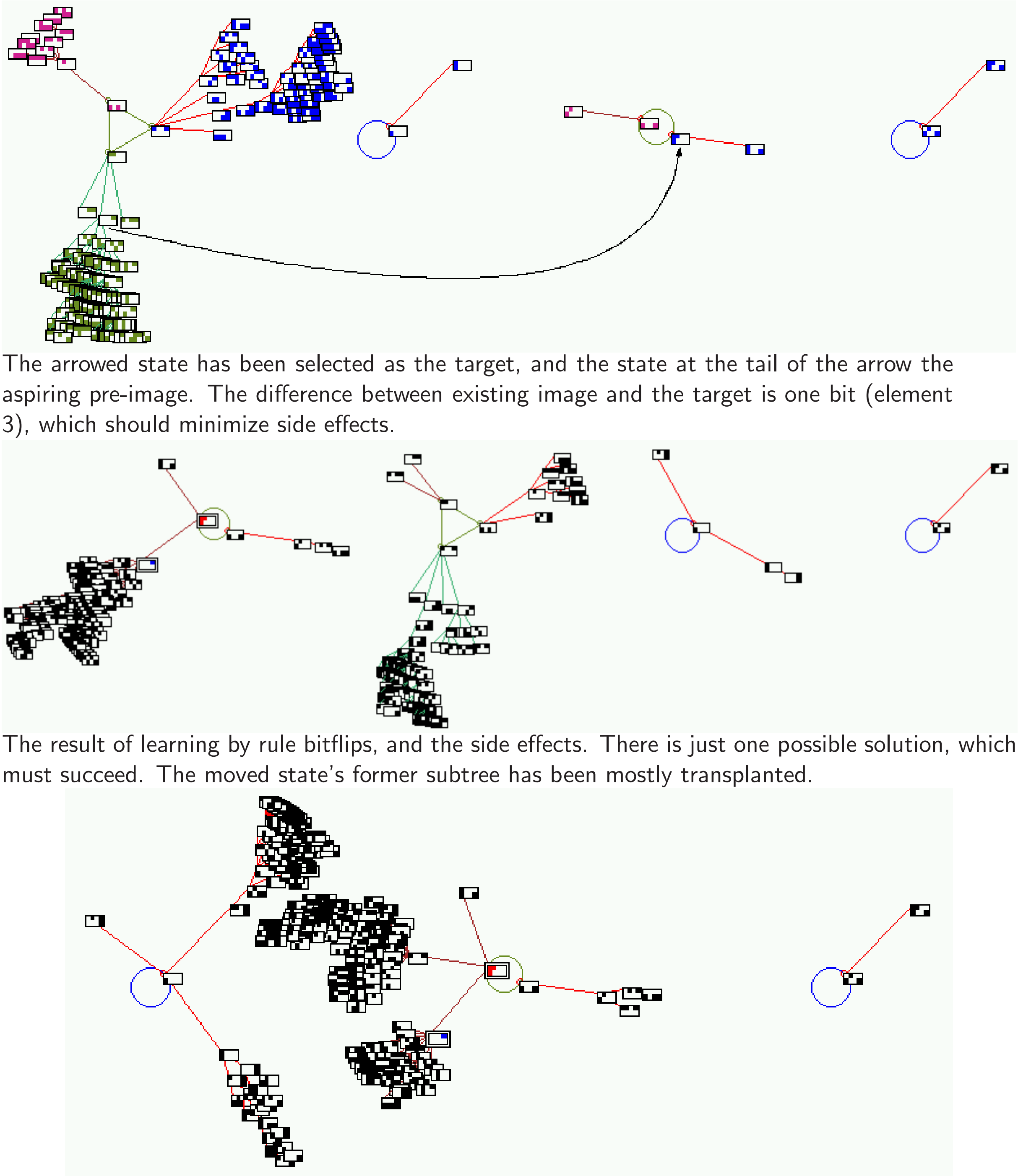
Attractors classify state-space into broad categories, the network's "content addressable" memory in the sense of Hopfield[10]. Furthermore, state-space is categorized along transients, by the root of each subtree forming a hierarchy of sub-categories. This notion of memory far from the equilibrium condition of attractors greatly extends the classical concept of memory by attractors alone[20, 22]. DDLab provides tools for "sculpting" the basin of attraction field towards a desired category structure, by adjusting the network, flipping bits in rules or moving wires. Because any such change, especially in a small network, usually has significant side effects, the methods are not good at designing categories from scratch, i.e. solving the inverse problem or reverse engineering. A rudimentary solution to that is provided in section 18.10. However, the learning/forgetting methods is useful for fine tuning a network which is already close to where its supposed to be.

The methods apply to rcode only. The wiring is treated as non-local even if set up as local, but the rule mutation algorithm requires a rule-mix. Its also possible to just highlight (without learning) a list or subset of states to see their location in the basin of attraction field.

One or more states can be added or deleted as pre-images of a given state. Adding or deleting pre-images is analogous to learning and forgetting. The network architecture is automatically revised to produce the required change either by moving wires or mutating rules. It turns out that the rule mutation algorithm is bound to learn a list of aspiring pre-images without forgetting the state's pre-existing pre-images; pre-images not on the list may also be learned. There is just one specific rule scheme mutation, which is bound to succeed. By contrast there are many alternative wire move mutations that may achieve the desired result, though success is not guaranteed, and pre-existing pre-images may be forgotten.

When applying both rule and wire learning algorithms, side effects are very almost certain to occur elsewhere in the basin of attraction, or in the basin of attraction field. The revised attractor basin can be immediately re-drawn to show the results and side affects of learning. The attractor basin can be progressively sculpted (and the network adapted) to produce a desired scheme of hierarchical categorization. In general, forgetting pre-images causes less severe side effects than learning, as minimal changes to the network architecture are required.

Note that learning by wire moves requires a network with non-local (random) wiring (see 12.4.1 to treat local wiring as if it where random). Mixed neighborhoods, and 2d ore 3d CA, are always treated non-local. Learning by rule mutation requires a network with mixed rules (see 14.4.3 to set up a rule mix where all the rules are the same).

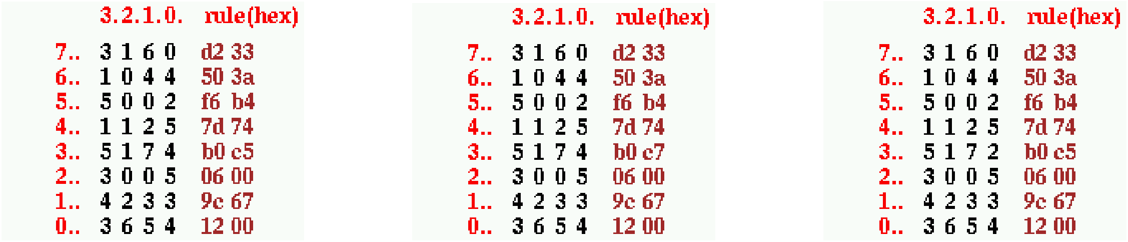


The arrowed state has been selected as the target, and the state at the tail of the arrow the aspiring pre-image. The difference between existing image and the target is one bit (element 3), which should minimize side effects.

The result of learning by rule bitflips, and the side effects. There is just one possible solution, which must succeed. The moved state's former subtree has been mostly transplanted.

The result of learning moving wires, with greater side effects. There are multiple possible solutions, including none.

Figure 34.1: The basin of attraction field of a RBN, n=8, k=4, before and after learning. The target state and its pre-image are highlighted with a double outline. Note that the order and orientation of the basins has changed.



```
     3.2.1.0.  rule(hex)
7..  3 1 6 0   d2 33
6..  1 0 4 4   50 3a
5..  5 0 0 2   f6 b4
4..  1 1 2 5   7d 74
3..  5 1 7 4   b0 c5
2..  3 0 0 5   06 00
1..  4 2 3 3   9c 67
0..  3 6 5 4   12 00
```

the original network

```
     3.2.1.0.  rule(hex)
7..  3 1 6 0   d2 33
6..  1 0 4 4   50 3a
5..  5 0 0 2   f6 b4
4..  1 1 2 5   7d 74
3..  5 1 7 4   b0 c7
2..  3 0 0 5   06 00
1..  4 2 3 3   9c 67
0..  3 6 5 4   12 00
```

learned by rule bitflips

```
     3.2.1.0.  rule(hex)
7..  3 1 6 0   d2 33
6..  1 0 4 4   50 3a
5..  5 0 0 2   f6 b4
4..  1 1 2 5   7d 74
3..  5 1 7 2   b0 c5
2..  3 0 0 5   06 00
1..  4 2 3 3   9c 67
0..  3 6 5 4   12 00
```

learned by wire moves

Figure 34.2: The RBN networks from figure 34.1 before and after learning. *left*: the original network. *center*: the altered network, learned by rule bitflips. The rule at element 3 has changed. *center*: the altered network, learned by wire moves. One input wire to element 3 has been moved.

34.1 Highlighting states in attractor basins

The nodes corresponding to a list of states may be highlighted in attractor basins (see also section 26.2. This highlights the results of learning and forgetting, but is also useful for seeing how a particular set of states is distributed, and how the distribution changes when network parameters are altered. Highlighted nodes are shown with an outer border. The target node is shown in red and aspiring pre-images in blue. For highlighted nodes shown as bit patterns, these are the colors of 1s, otherwise 1s are shown black. Nodes other than highlighted nodes may be suppressed entirely. The highlighting feature may be used for any network including regular 1d CA.

34.2 Basin layout for learning

The default layout and node display of the basin of attraction field may need some adjustment for a good presentation for learning. Setting layout is described in section 25.2), and node display in section 26.2 – 2d bit/value pattern is usually a good choice, as in figure 34.2.

34.3 Selecting the wiring graphic

Whenever the network is reviewed as a 1d, circle, or 2d wiring graphic (section 17.1), learning may be implemented (option **learn-l**). Learning and highlighting is usually done while drawing basin of attraction fields (or single basins). When a field or basin is complete, the following top-left prompt is presented (see section 30.4),

graphics-g image: prt/save/load-p/s/l, ops-o
toggle: single-field-t STP-P, seed-e (**seed-e** *not for the basin of attraction field)*
net-n, back-q cont-ret:

Enter **n** to select the top-right network architecture prompts (section 17.1), which include the following options,

graphic: 1d+timesteps-1 circle-c, 2d-2: (*or* **2d+3d-3:** *for a 3d network*)

This prompt may also be reached from various other points in DDLab, described in section 17.1. When pausing space-time patterns, the prompt can be reached from the option in section 32.14.

34.4 Selecting the learn/highlight window

Enter **1**, **c** or **2** in section 34.3 above (**1** is probably the better choice) to show the wiring graphic in the lower half of the screen, and its top-right prompt window (see section 17.1). Note that changes to the wiring and rules may be made directly from this prompt, which may be used in conjunction with automatic learning/forgetting.

One of the prompts reads **learn-l**. Enter **l** to open the 'learn/forget/highlight" window.

34.5 Select the target state

The initial prompts sets the "target" state, to which pre-images (i.e. direct predecessors), will be directly attached (learning), or detached (forgetting). Prompts are presented in two separate windows, one above the other. The upper window indicates that the target state is to be set,

learn/forget/highlight: set target state

The lower window presents the first of a series of prompts to select the target state,

example for a 1d network
Select target stare (1d n=14), win-w empty-e fill-f dec-d rand-r
bits1d-b bits2d-B hex-h repeat-p load-l (def-r): (**d** *if* $n \leq 32$)

These prompts are the same as for selecting a seed (the initial state), described in section 21.1, but the default selection method is setting 2d bits (see sections 21.4 and 21.4.6).

34.6 Select aspiring pre-image/s

Once the target state has been selected, one or more states may be specified as its "aspiring pre-images" (i.e. direct predecessors). These states will be will be directly attached (learning), or detached (forgetting). Alternatively the states may just be just highlighted.

The set of states may be an arbitrary list, a range of decimal equivalents (if $n \leq 32$), or may be based on Hamming distance, parity, or the previous list may be repeated. The following upper prompt is presented,

pre-images: as before-b parity-p range-r hamm-h1/2 number-(def 1):
(**range-r** *if* $n \leq 32$)

option ... *what it means*

before-b . . . enter **b** to repeat the previous set of aspiring pre-images. If a list (above) was previously selected, prompts will be presented as above, but the default for each pre-image will correspond to the previous list.

parity-p . . . enter **p** to select states based on odd or even parity.

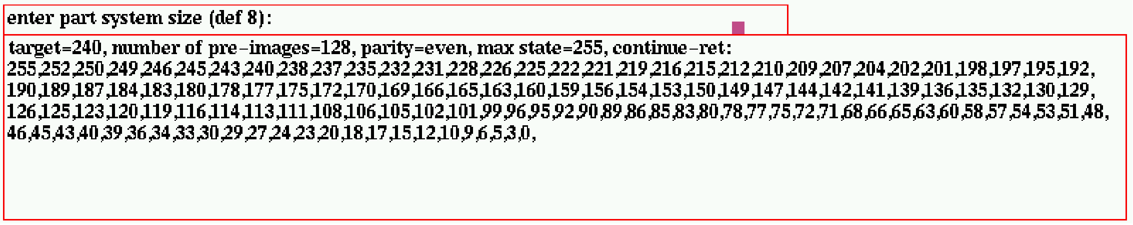
range-r . . . If $\boldsymbol{n} \leq \mathbf{32}$, enter **r** to specify a range of decimal equivalents.

hamm-h1/2 . . . enter **h1** or **h2** to select states based on a hamming distance of 1, or both 1 and 2, from the target state.

number-(def 1) . . . enter **return** to specify just one aspiring pre-images, or a number to specify the size of an arbitrary list of pre-images (see section 34.6.4).

Once set, the target state and list of aspiring pre-images are reviewed in a window (see section 34.6.5).

34.6.1 Odd or even parity



```
enter part system size (def 8):
target=240, number of pre-images=128, parity=even, max state=255, continue-ret:
255,252,250,249,246,245,243,240,238,237,235,232,231,228,226,225,222,221,219,216,215,212,210,209,207,204,202,201,198,197,195,192,
190,189,187,184,183,180,178,177,175,172,170,169,166,165,163,160,159,156,154,153,150,149,147,144,142,141,139,136,135,132,130,129,
126,125,123,120,119,116,114,113,111,108,106,105,102,101,99,96,95,92,90,89,86,85,83,80,78,77,75,72,71,68,66,65,63,60,58,57,54,53,51,48,
46,45,43,40,39,36,34,33,30,29,27,24,23,20,18,17,15,12,10,9,6,5,3,0,
```

Figure 34.3: Selecting even parity in section 34.6.1, for network size, n=8. A list of the $\boldsymbol{n/2}$ decimal equivalent even parity values are displayed.

Odd and even parity signifies that the total of 1s in a bit string is an odd or even number (zero has even parity). If **p** is entered in section 34.6, the aspiring pre-images will be automatically specified as those states with the given parity relating (at most) to the first 20 cells (where the remaining cells have value 0). The following upper prompt is presented,

parity: odd-1, or even-(def):

Enter **return** for even parity, **1** odd or even parity. A further upper prompt allows the parity to relate to part only of the network, size n, where $\boldsymbol{n} \leq \mathbf{20}$, or part only of the first 20 network elements,

enter part system size (def 8): *(for n=8)*

If a size x, less than the system size n (or less than 20 if $\boldsymbol{n} > \mathbf{20}$), is entered, states with the given parity relating only to cell indices 0 to x (and where cell indices x+1 to n are equal to 0) will be selected.

34.6.2 Range of decimal equivalents

If $\boldsymbol{n} \leq \mathbf{32}$, enter **r** to automatically set a range of aspiring pre-images, set between two selected values (the maximum decimal equivalent that can be set is $\mathbf{2^{32}-1}$. The following (upper) sequence of prompts is presented to set the start and end decimal equivalents, and the "step" or increment size (default 1),

range of pre-images (def 1-8, max 255), start: end: step:
(for n=8)

34.6.3 Pre-images according to Hamming distance

Enter **h1** or **h2** in section 34.6, automatically set the aspiring pre-images based on a Hamming distance of 1, or both 1 and 2, from the target state, i.e. states that differ by just 1 bit, or by 1 bit or 2 bits, from the target state.

34.6.4 List of aspiring pre-images

Enter **return** to specify just one aspiring pre-images, or a number to specify the size of an arbitrary list of pre-images, in section 34.6. A prompt, or series of prompts to select the pre-image/s will be presented in the lower window. These prompts are the same as for selecting a seed (see section 21.1), except for the heading **Select aspiring pre-image 1**, followed by **...pre-image 2**, **...pre-image 3**, etc. for a list. The default selection method is setting 2d bits (see sections 21.4 and 21.4.6).

34.6.5 Review target state and aspiring pre-images

applies only if the maximum decimal equivalent $< \mathbf{2^{32}}$

Once the target state and list of aspiring pre-images have been set in sections 34.6 to 34.6.4 above, the decimal equivalents of the selected states, and further information, will be displayed in the lower window, as in the example for parity in figure 34.3. Examples for a range, Hamming distance, and an arbitrary list, are given below,

a range of decimal equivalents, section 34.6.4
target=240, pre-images=16, range 2-255, step=16, cont-ret:
2,18,34,50,66,82,98,114,130,146,162,178,194,210,226,242, *(for n=8)*

the set of 1-bit mutants, Hamming distance=1, section 34.6.3
target=240, pre-images=8, 1-bit mutantsrange 2-128, step=16, cont-ret:
2,18,34,50,66,82,98,114 *(for n=8)*

an arbitrary list of 5 states, section 34.6.4
target=240, pre-images=5, cont-ret:
50,167,152,88,94 *(for n=8)*

If there are more states than will fit in the window, the following prompt allows more to be displayed, or to abandon the display and continue, **more-m cont-ret:**

34.7 Learn, forget, or highlight only

Once the target state and aspiring pre-image/s have been set (sections 34.5 to 34.6.4), and reviewed (section 34.6.5), prompts are presented to learn, forget, or just highlight the aspiring pre-images (as well as the target state).

For a 1d CA (with local wiring and as single rule), or a network with 1d local wiring and mixed rules, a prompt (in the upper window) first allows the wiring to be redefined as nonlocal,

1d CA
local 1d: nonlocal wiring-w:
1d local wiring and mixed rules
local 1d, rule-mix: nonlocal wiring-w:

Enter **w** to redefine the wiring as nonlocal, allowing learning by wire moves (and removing compression in attractor basins, if set for CA, section 26.1). Otherwise the prompt continues as follow,

... highlight only-(def):

Enter **return** to highlight the selected states. The program skips directly to the prompt in section 34.11 to exit the learning routine. Highlighting is the only possibility for a network were the wiring is defined as local 1d, using the local 1d reverse algorithm (section 2.18). Note that 1d local wiring can also be redefined so that its treated as nonlocal in section 12.4.1. Highlighting on its own can be useful to show how a selected set of states is distributed in attractor basins.

For networks that do not have 1d local wiring (or if **w** was entered above) the following prompt is presented,

highlight only-h, forget-f, learn-def:

Enter **l** or **f** to learn or forget, i.e. add/remove the aspiring pre-image/s from the existing pre-image fan of the target state. When the attractor basin is redrawn according to the altered network, the target and aspiring pre-image states will be highlighted (see section 34.1). Enter **h** to just highlight the selected states; the program skips directly to the prompt in section 34.11 to exit the learning routine.

34.7.1 Learning/forgetting by wire moves or bit-flips

If **l** or **f**, to learn or forget , was entered in section 34.7, a prompt (in the upper window) is presented to proceed by moving wires or flipping bits in rule-tables. For a single rule network, only learning/forgetting by wire moves is possible, and the prompt is as follows,

learn by wire-moves only-(def): *(or* **forget...***)*

Section 14.4.3 describes how to To set up a single rule network that is treated as a rule-mix.

For a mixed-rule network with local 1d wiring, only learning/forgetting by bit-flips possible, and the prompt is as follows,

learn by bit-flips only-(def): *(or* **forget...***)*

Otherwise, for networks with nonlocal wiring and a rule mix, and also networks with mixed-k which by definition has both, the prompt is as follows,

learn by wire-moves -w, bit-flips-(def): *(or* **forget...***)*

34.8 Learning/forgetting by bit-flips

The bit-flip learning algorithm[20] changes specific bits in rule-tables to make the target state the actual successor of each aspiring pre-image, instead the original successor under the original network parameters. The procedure must succeed. Pre-existing (or just learnt) pre-images can not be "forgotten", i.e. detached as pre-images of the target state, by learning more states as pre-images. However other states not on the list of aspiring pre-images may also be learned, and side effects will occur elsewhere in the attractor basin. To "forget" a pre-image just one of a set of bit-flips is required. This is chosen from the set at random. As fewer network changes are needed there will be fewer side effects.

If "**bit-flips**" was selected in section 34.7.1, the required changes to the network will be made, and the following (upper) prompt is presented,

learn/forget/highlight more-m:

Enter **m** to repeat the procedure from section 34.7, otherwise the network as it stands is accepted.

34.9 Learning/forgetting by wire-moves

The wire-move learning algorithm[20] moves a single wire at each cell position where there is a mismatch between the actual successor cell and the relevant cell in the target state. There is a choice of wires to select from the pseudo-neighborhood, and a choice of new coupling positions, some of which will correct the mismatch. The aspiring pre-image order, the pseudo-neighborhood order, and the coupling order, are all shuffled at random, and the first successful move is selected (if it exists). This may cause pre-existing (or just learnt) pre-images may be forgotten, i.e. detached as pre-images of the target state. If "just learnt" states are forgotten when learning the next pre-image, the next wire-coupling alternative in the shuffled order will be tried. If none succeed the algorithm continues with the next aspiring pre-image. On completing the learning pass, the state/s learned (and not learned) are listed by their decimal equivalents (for $\boldsymbol{n} \leq \mathbf{31}$ only), with the following prompt in the lower window,

failing to learning one pre-image by re-wiring
target state=73,
193,<-no
re-learn by wire-moves -w, bit-flips -b: *(or* **re-forget ...** *)*
cont-ret:

tried rewiring again, successfully learning one pre-image
target state=73,
193,<-ok
success - aspiring pre-image learned *(or ...* **forgotten***)*
cont-ret:

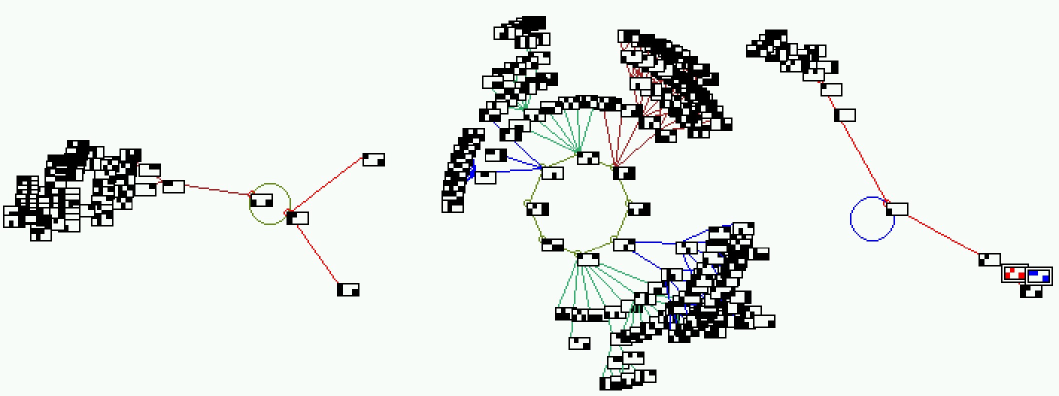


Figure 34.4: Learning one pre-image by rewiring, starting with the network in figure 34.1(*top*). The target state and its pre-image (in the last basin, on the right hand transient) are highlighted with a double outline.

> *learning multiple pre-images, examples for Hamming distance 1 from state 8*
> **target state=8, 136,0<-no**
> **24,72,12,9,10,40,<-ok re-learn by wire-moves -w, bit-flips -b:** *(or* **re-forget . . .** *)*
>
> *if all multiple pre-images are learned (or forgotten)*
> **. . .**
> **success - all aspiring pre-image learned** *(or . . .* **forgotten***)*

<-ok indicates learnt states, **<-no** unlearnt states. If there are unlearnt states remaining, options are offered to try re-learn again by wire moves, or by bit-flips (which always succeeds). If **w** is selected, the last **<-no** state will be learned first, followed by the set of shuffled **<-ok** states, then the remaining shuffled **<-no** states. On completing the pass the resultant listing and prompts are repeated. Note that for a large system with a large list of aspiring pre-images, a wire-move learning pass may take an unpredictably long time.

To forget a pre-image just one appropriate wire move is required. Forgetting is generally much easier and quicker than learning. As fewer network changes are needed there will be fewer side effects. Note that in the forgetting procedure **<-ok** indicates states still attached to the target state, i.e. not forgotten, and **<-no** indicates states successfully forgotten, for example if all are **<-no** the following prompt is presented below the listed states,

> **. . .**
> **success - aspiring pre-image forgotten**
> **cont-ret:**

If **return** is entered at the various prompts above, the following prompt is presented,

> **learn/forget/highlight more-m:**

Enter **m** to repeat the procedure from section 34.7, otherwise the network as it stands is accepted.

34.10 Highlighting options

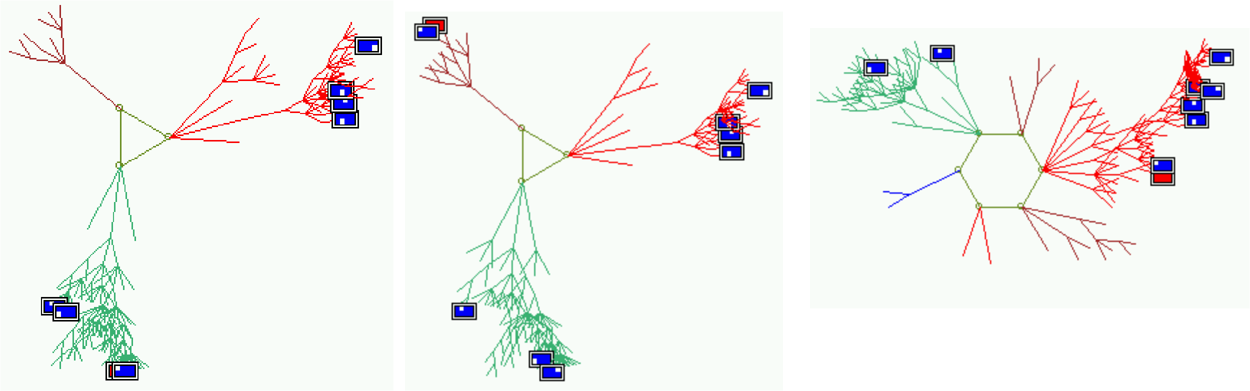


Figure 34.5: Tracking a set of states states, at Hamming distance 1 from the state all 1s, for the RBN network in figure 34.1(*top*), for a series if 1-bit rule mutations. Just one basin is show, which contains the whole set. *left*: the original network. *to the right*: a series if 1-bit rule mutations. The highlighed states have a double outline, with the "target" state shown in red, and the set of states at Hamming distance 1 in blue. All other nodes have been suppressed.

Once learning is complete, or if the "**highlighting only**" option was selected in section 34.7, the nodes representing the target state, and the set of aspiring pre-images, will be highlighted in the attractor basin. The particular form of highlighting depends on the node type selected in section 26.2,

highlight: none-n, suppess all other nodes-s:

Enter **return** to highlight selected nodes and show other nodes normally according to the node type selected in 26.2. If a 2d bit pattern was selected, the non-highlited nodes will be shown in black and white, the target state in red and white, and aspiring pre-images in blue and white, as in figure 34.4

Enter **s** to suppress the display of nodes other than highlighted nodes. Enter **n** not to highlight, but show all nodes as normal.

34.11 Learning/forgetting/highlighting complete

Following the options section 34.10, the wiring graphic and associated prompts (section 17.1) will reappear. Enter return to exit the wiring graphic, and revert to the top-left "attractor basin complete" (section 30.4) prompt, or the pause options for space-time patterns (section 32.14).

Enter **return** to redraw the attractor basin (or restart space-time patterns) with the new (learnt) parameters. Attractor basins will be drawn with the specified highlighting. Note that after the attractor basins have been drawn, if **return** is entered again, the network will be changed by the mutations set (by default) in chapter 28. This may be what is required, for example to see how a set of highlited states is redistributed for a succession of mutated attractor basins, as in figure 34.5. However, when “sculpting” the basin of attraction field, its best to first deactivate mutation in section 28.1.

Intentionally Blank

Chapter 35

Filing

DDLab has functions for saving and loading a variety of DDLab specific file types, which have default three letter extensions, for example `myrul_v3.rul`). PostScript files generated in DDLab have the two letter extension `.ps`.

DDLab was first developed in the DOS, and DOS file name conventions still apply for all versions, including Linux-like operating systems. A DOS file name has up to eight characters (starting with a letter) plus a three character extension (except PostScript `.ps`).

The lowercase letter **q** cannot be included because in DDLab **q** is for backtracking or deleting input. The filename extension is not entered as part of the filename – it will be automatically added according to the type of file.

This chapter lists the file types, and describes the methods for saving and loading files. The following file types are described in more detail in other sections in the manual, including their encoding, and loading constraints for compatibility with the current network.

Network architecture files . . . (`.mix`, `.w_s`, `.r_s` and `.wrs`) Described in detail in chapter 19. Encoding: sections 19.3 and 19.3.1. Constraints that apply to loading different types of file into a given network: section 19.4.1.

Single rcode, kcode or tcode files . . . (`.rul .vco .tco`) Encoding: section 16.16. [$\boldsymbol{v}, \boldsymbol{k}$] constraints: the value-range $\boldsymbol{v}$ and neighborhood size $\boldsymbol{k}$ in the file must be the same as in the current setup.

The seed or state (and rule-table) . . . (`.eed`), for the seed or initial state or the rule-table saved as a seed, or a defined patch. Encoding: section 21.9. Size/dimension loading constraints: section 21.7.1.

35.1 File types, default filenames, and extensions

The DDLab specific file types, default extensions, and default filenames are listed below.

type	*default file name*	
K-MIX	`my_mix.mix`	The neighborhood-mix, (sections 9.5, 9.12.3, 19.5). Encoding: see section 19.3.1.
WIRING ONLY	`mywso.w_s`	The wiring scheme (chapter 19). Encoding: sections 19.3.

SINGLE RULE
SINGLE KCODE
SINGLE TCODE
	`myrul_v2.rul`
	`myvco_v3.vco`
	`mytco_v4.tco`
	etc.
		...for the full lookup-table ...*(for example)*
		...for the totalistic kcode-table ...*(for example)*
		...for the totalistic tcode-table ...*(for example)*
		A single rule, kcode or tcode for a given $\boldsymbol{v}$ and $\boldsymbol{k}$ (chapter 16, and sections 30.5.1, 32.14.2). Encoding: section 16.16. The default filename includes $\boldsymbol{v}$ and $\boldsymbol{k}$.
		When setting a rule, kcode, or tcode, in chapter 16, the most recent will be automatically saved, with the filename `l_v2k3.rul`, `l_v2k3.vco`,`l_v2k3.tco` (depending $\boldsymbol{v}$ and $\boldsymbol{k}$); these rules can then be automatically reloaded.

TABLE	`mytbl_v2.tbl` `mytbl_v3.tbl`	The rule-table, or in TFO-mode the kcode-table or tcode-table, for a given value-range $\boldsymbol{v}$, can be saved as an ascii file (section ??). The default filename depending on $\boldsymbol{v}$.

RULE SCHEME
KCODE SCHEME
TCODE SCHEME
	`myrso_v2.r_s`
	`myrso_v3.r_v`
	`myrso_v4.r_t`
	etc.
		...for full lookup-tables ...*(for example)*
		...for totalistic kcode-tables ...*(for example)*
		...for totalistic tcode-tables ...*(for example)*
		The rule, kcode, or tcode, scheme (with corrresponfing extensions `.r_s`, `.r_v`, `.r_t` (chapter 19). For homogeneous $\boldsymbol{k}$ only, otherwise use the WIRING+RULE SCHEME below. The default filename includes the value-range $\boldsymbol{v}$. Encoding: sections 19.3.

WIRING+RULE SCHEME	`mywrs_v2.wrs`
	`mywrs_v3.wrv`
	`mywrs_v4.wrt`
	etc.
		...for the full lookup-table ...*(for example)*
		...for the totalistic kcode-table ...*(for example)*
		...for the totalistic tcode-table ...*(for example)*
		Both wiring and rules (chapter 19).
		Encoding: sections 19.3.
		The default filename includes the value-range $\boldsymbol{v}$.

NETWORK DATA	`my_net.dat`	...for the network data as an ascii file (section 19.6.

SEED	`mysee_v2.eed`
	`mysee_v3.eed`
	`mysee_v4.eed`
	etc.
		A state bit/value-string, including the value-range $\boldsymbol{v}$, size $\boldsymbol{n}$, dimension (1d, 2d or 3d), and the $\boldsymbol{i, j, h}$ coordinates (sections 30.5.2, 32.14.5). Encoding: section 21.9. The default filename includes the value-range $\boldsymbol{v}$.
		As above, but the default filename changes to `mypch_v`$\boldsymbol{x}$ when saving a patch – part of a state.

BASIN DATA	`my_data.dat`	An ascii file for the field/basin/subtree data and list of states data (sections 27.6-27.8.2, 19.6). Encoding: ascii.
PRE-HIST-DATA	`my_prh.prh`	Data for the pre-image histogram. (section 24.6.5). Encoding: a binary file of an unsigned long array, recording each in-degree frequency, from zero up.
HIST-DATA	`my_his.his`	Data for various other histograms (sections 9.12.1, 17.8.13, 31.17, 31.19.4, 31.20). Encoding: a binary file of an unsigned short array, recording the height of each histogram bar, from zero up.

EXHAUSTIVE	`myexh_v2.exh`
	`myexh_v3.exh`
	`myexh_v4.exh`
	etc.
		An exhaustive mapping, listing each state and it successor (section 29.8.1, 29.7.2, 18.1.2). Encoding: a binary file of a char array recording a list of $\boldsymbol{v^n}$ successor bit/value strings, each in $\boldsymbol{v^n/8}$ bytes (minimum 1 byte), where the pre-image is the array index, from zero up. The default filename includes the value-range $\boldsymbol{v}$.

ORDER-DATA	`my_order.ord`	For the sequential updating order. (section 29.9, 29.9.2). Encoding: a binary file of an unsigned short array, recording the sequential order according to the array index, from zero up.
STAT-SAMPLE	`mystat.sta`	The statistical sample of automatically classified rule-space, (chapter 33, sections 33.3, 33.4, 32.14.1). Encoding: section 33.8.
GRAPH LAYOUT	`my_grh.grh`	The layout of the network-graph (section 20.2), or the attractor jump-graph (section 20.3). Encoding: a binary file of a signed short array, recording the $\boldsymbol{x, y}$ coordinates of each graph node according to the array index, from one up.
SCREEN	`myimage.nat`	The screen image, (section 5.6.1). Encoding: (the same for all platforms) has a DDLab specific format too involved to explain here. A `.nat` file, saved on a given platform can be successfully loaded back into DDLab on the same platform, even if the screen size has been changed. If saved on one platform and loaded back to another, the results are less reliable.

PostScript	`my_sPS.ps`	... for space-time patterns (and rule-tables).
	`my_3dPS.ps`	... for 3d space-time patterns.
	`my_bPS.ps`	... for attractor basins.
	`my_ngPS.ps`	... for network-graphs.
	`my_jgPS.ps`	... for jump-graphs.
	`my_mxPS.ps`	... for the wiring-matrix.
	`my_wgPS.ps`	... for the wiring-graphic. These are a vector PostScript files described in chapter 36. The files can be loaded in `gv` (GhostView), and included in LaTex documents. Encoding: ascii - in the PostScript language.

35.2 The filing prompt

In general when saving or loading a file, the following filing prompt appears in a top-right window,

SAVE [or **LOAD**] [type] **- filename (no ext) .**[extension] **will be added**
change dir-d, now: [current path]
list-?, quit-q, default [filename]**:**

for example,

loading a $\boldsymbol{v} = \mathbf{3}$ *seed in Linux-like systems*
LOAD SINGLE RULE-filename (no ext) .rul will be added
change dir-d, now:/home/andy/sussex/ddlabm06/ddfiles:
list-?, quit-q, default myrul_v3:

saving a single $\boldsymbol{v} = \mathbf{3}$ *rule in DOS*
SAVE SEED-filename (no ext) .eed will be added
change dir-d, now: C:\ddlabm06 *(for example)*
list-?, quit-q, default mysee_v3:

To load or save, enter a legal filename, up to 8 characters starting with a letter but excluding **q**, and without the filename extension, which is automatically added, or enter **return** to accept the default filename. When the filename is entered, say `rule193` to load a single rule, the lowest line of the prompt will respond with,

filename=rule193.rul REVISE-q cont-ret: *(for example)*

Enter **return** to load the filename, or **q** to revise. If the file is not found, the following message appears in a top top-right window,

file error rule193.rul, cont-ret: *(for example)*

Enter **return** to continue.

35.3 Loading restrictions on the value-range $[\boldsymbol{v}, \boldsymbol{k}]$ – rules

When loading any file that includes rules (`.rul`, `.vco`, `.tco`, `.tbl`, `.r_s`, `.r_v`, `.r_t`, `.wrs`, `.wrv`, `.wrt`) the $[\boldsymbol{v}, \boldsymbol{k}]$ parameters (value-range and neighborhood size) of the file, and the current (base) parameters $[\boldsymbol{v}, \boldsymbol{k}]$, must be equal, otherwise the following error messages are displayed in a top top-right window,

*(current-**v** not equal to file-**v**)*
not loaded! v-range(3) not equal to file v-range(4), cont-ret: *(for example)*
*(current-**k** not equal file-**k**)*
not loaded! file-k(5) not equal to base-v(7), cont-ret: *(for example)*

Enter **return** to continue.

35.4 Loading restrictions on the value-range $\boldsymbol{v}$ – seeds

When loading a seed the value-range $\boldsymbol{v}$ of the file would usualy be equal to the current (base) $\boldsymbol{v}$, but a file with a different a $\boldsymbol{v}$ can still be loaded, in which case the following prompt appears,

file-v(2) != base-v(8), abandon-q, load anyway-ret: *(for example)*

Details about constraints for loading a seed are given in section 21.7.1.

35.5 Changing the directory

The current directory is shown in the filing prompt (section 35.2), and can be changed by entering **d**. The following prompt appears,

example for DOS
current path C:\DDLABM06, CHANGE drive/directory
new path, (ie c:\dir\subdir):

or

example for Linux-like systems
current path /home/andy/sussex/ddlabm06/ddfiles, CHANGE drive/directory
new path or subdir: (ie **otherdir** or "**..**" for the parent directory)

In DOS, enter the full new path, (i.e. c:\newdir\newsub\etc) - directory names are limited in the same way as file names - 8 characters starting with a letter but excluding **q**.

In Linux-like systems enter the child directory, or "**..**" (dot dot) for the parent directory - **q** cannot be included in a directory name, otherwise there is no restriction.

The filing prompt (section 35.2) will reappear with the new path, which will stay current until changed again. If the new path has an illegal name, the following message appears,

cant change to [illegal path]

On **return** the program reverts to the filing promp (section 35.2).

35.6 List files

To list the files with the current file name extension, enter **?** in the filing prompt (section 35.2). A list will be displayed in a window on the right of the screen. The following prompt appears at the bottom of the list,

list again-a
more-m *(if there are too many files to fit)*
or filename
: *(the file name (without extension) can be entered here*

If there are too many files to fit, enter **m** to see more. A filename (without extension) may be selected in this window, or enter **return** and select the filename in the filing prompt (section 35.2).

Chapter 36

Vector PostScript capture of DDLab output

Graphic output in DDLab can be captured in vector PostScript files. Vector graphics are preferable for publication quality images over bitmap graphics, as they can be scaled without degrading the resolution. Most figures in this manual are vector images, others are bitmap graphics[1]. At the time of writing, vector PostScript files are available for the following DDLab graphic output, shown with the default filenames:

default flilename ... *graphic output*

`my_sPS.ps` ... 1d and 2d snapshot from seed information, including the 2d version of 3d (sections 21.4.1), and also the rule-table as a 1d seed (section 16.4.3).
1d and 2d snapshots can also be captured directly from interupted space-time patterns. (section 32.14).

`my_3dPS.ps` ... For the 3d isonometric from seed information, (section 21.4.7).

`my_sgPS.ps` ... Snapshot of the the network-graph (if active) which can be captured when space-time patterns are interrupted.

`my_ngPS.ps` ... The network-graph.

`my_jgPS.ps` ... The attractor jump-graph, including basins at nodes.

`my_bPS.ps` ... Attractor basins of all types and presentations, including for a range of system size n.

`my_mxPS.ps` ... The wiring-matrix (17.2.1).

`my_wgPS.ps` ... The wiring-graphic (1d time-steps, 1d circle, 2d, 3d)

The vector PostScript options are initiated from prompts relating to various the graphic output listed above, and described in the relevant sections. The methods work in both Linux-like systems and DOS. The top right prompt for saving to PostScript allows presentation alternatives, then the top-right filing prompt (section 35.2) appears for naming the file with the extension **.ps** added automatically, with a default filename as shown above.

[1] Bitmap graphics are captured with the XV program (`http://www.trilon.com/xv/`).

Vector PostScript files are ascii files in the PostScript language, so can be easily edited. Functions in DDLab generate these files automatically according to the the current presentation of the DDLab graphic, but also with various options for the PostScript image itself such as color or greyscale. There is no limit on the size of a PostScript file, so for images with many components, lines, nodes etc. proceed with caution, especially for attractor basins, to avoid excessively large files – in such cases a bitmap image might be preferable (see section 2.14). Note that if an attractor basin is interrupted, the PostScript file of part of the graphic generated up to that point will still be valid.

36.1 Cropping and editing vector PostScript

A vector PostScript file created in DDLab is an ascii file which can be edited allowing the image to be cropped, images merged and annotations added. Instructions for the PostScript language can be found online[2], so there is no need to delve further except to note that the image can be cropped by editing the second line of the file to define a new BoundingBox with the bottom left and top right coordinates, for example,

%%BoundingBox: 20 40 924 670

The coordinates can be found by a readout of the cursor position when the file is opening in GhostView.

Another way of cropping is within the LaTex graphics command, for example,

```
\epsfig{file=filename.ps, bb=81 85 437 255, clip=, width=1\linewidth}
```

Where “`bb=81 85 437 255`” is an example of the clip coordinates (0,0 is bottom left in PostScript) and “`clip=`” forces LaTex to ignore anything outside the clip limits.

36.2 PostScript space-time patterns

For 1d, 2d and 3d, saving in PostScript can be done by entering **F** from *Setting the seed as bits or values* prompts (sections 21.4 and 21.4.7). The image of a rule pattern can also be saved in PostScript by entering **F** from *Setting the rule as bits or values* (section 16.4) and 16.4.3).

Saving in PostScript can also be done during a forward run of space-time patterns when paused with **q**. For 1d or 2d, an option **PostScript-p** is available in the top-right pause window (section 32.14) to save the PostScript file immediately. For 3d its necessary to first display the space-time pattern in a seed revision window **state:rev-e** and from there save a PostScript file of the 3d isometric.

The immediate 1d and 2d space-time pattern PostScript options apply if the current *presentation* is in 1d or 2d, whatever the *native* dimension of the network which could be 1d, 2d or 3d;

[2] For example, the PostScript Language Tutorial and Cook Book (`http://www-cdf.fnal.gov/offline/PostScript/BLUEBOOK.PDF`).

the presentation can be toggled between 1d, 2d and 3d with on-the-fly option **T** (see section 32.9.1).

36.2.1 Directly scanning space-time patterns

Directly scanning the DDLab space-time image to generate a PostScript file is the only method available for 1d, and one of the possible options in 2d, as opposed to using information about a state held in DDLab. An advantage of direct scanning is that the PostScript file can then be exactly as presented in DDLab at that moment, according to the many alternative on-the-fly presentations options (chapter 32, such as color by neighborhood, filtering, and frozen cells. For 1d there could be new seeds and rules within one space-time pattern.

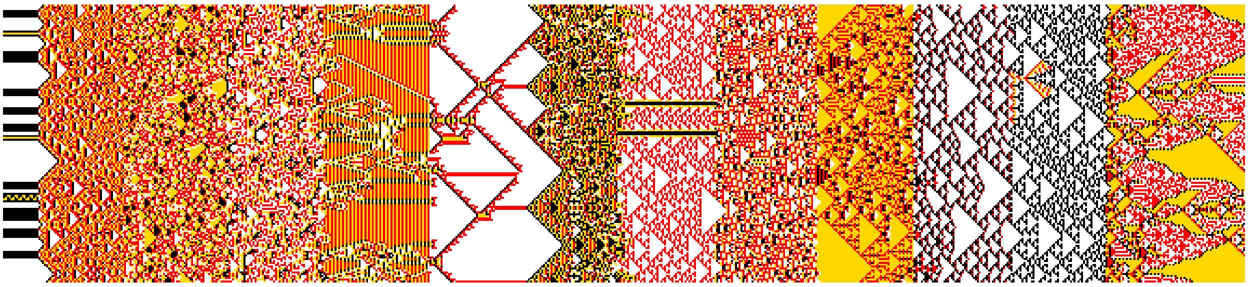


Figure 36.1: A Postscript image from a scanned 1d space-time pattern, with neighborhood color: random $\boldsymbol{v4k3}$ kcode was changed on-the-fly (see section 32.4.1) several times as the space-time pattern scrolled. Each new rule picked up the current state as its initial state. $\boldsymbol{n}$=150, 669 time-steps. The space-time pattern image is rotated by 90° so the time flows from left to right.

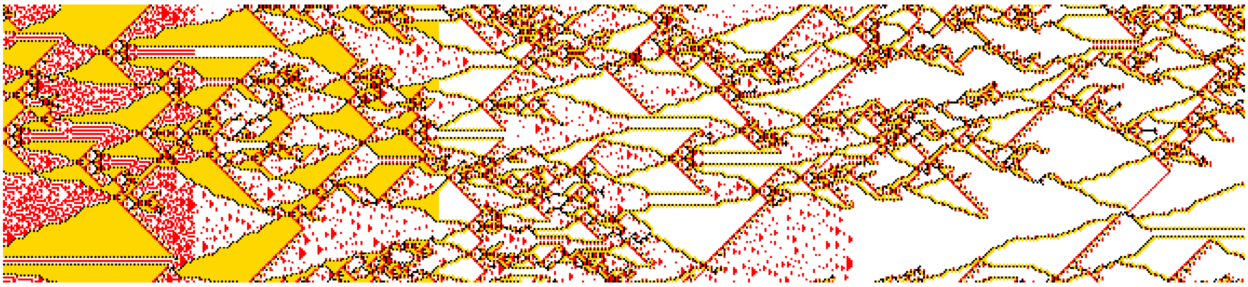


Figure 36.2: A Postscript image from a scanned 1d space-time pattern with neighborhood color: The space-time pattern in figure 36.1, was allowed to to run on keeping the last rule. Here the space-time pattern is shown filtered on-the-fly (see section 32.10) 4 times as the space-time pattern scrolled. $\boldsymbol{v4k3}$ kcode (hex) 89d17b2278) $\boldsymbol{n}$=150, 669 time-steps. The space-time pattern image is rotated by 90° so the time flows from left to right.

36.2.2 1d PostScript space-time patterns

For 1d, the PostScript file is created by scanning the space-time pattern image on the DDLab screen, though just part can be scanned and saved, and the PostScript file can be created with various presentations.

Enter **p** for the top-right PostScript options, where each prompt ending in a colon (**:**) will pause for input,

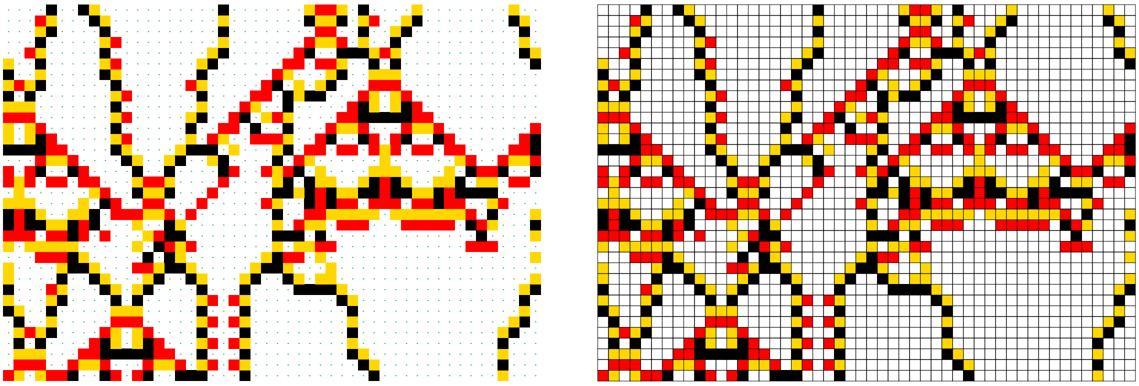


Figure 36.3: Postscript images from a scanned 1d space-time pattern, filtered, and with neighborhood color, for the same rule as in figure 36.2, *left*: showing dots in the white spaces, *righ*: showing black divisions between cells. n=50, 35 time-steps. The space-time pattern is orientated as in DDLab, with time flowing down.

create PostScript image for 1d space-time patterns, exit-q
1d size (max/def=100): time-steps (max/def=669): *(values shown are examples)*
cellscale=1.00 dots(off)=0.70 divs(off), amend settings-a:

To save just a top left patch, enter the size (the number of cells horizontally from the left), followed by the number of of time-steps (from the top of the screen down). Defaults are the size and the number of time-steps visible- enter **return** to accept.

If **a** is entered (**amend settings-a** the cell scale, dots on white cells and divisions between cells can be amended. The following further series of prompt are presented,

change: cellscale: togdots reate PostScript image for 1d space-time patterns, exit-q
1d size (max/def=100): time-steps (max/def=669): *(values shown are examples)*
cellscale=1.00 dots(off)=0.70 divs(off), amend settings-a:

There are further presentation prompts for the cell scale, dots on zero value cells, and divisions between cells described in section ?? below. Enter **a** to amment the default settings or **return** to accept. The cellscale (in pixels) is whatever is currently on thescreen.

36.2.3 2d PostScript space-time patterns

As in 1d, a 2d snapshot can be scanned from the DDLab image to create a PostScript file, but the PostScript file can also be created from the information about a state held in DDLab. Thus for 2d the **PostScript-p** option is also available from section 21.4 *Setting the seed as bits or values*; to access, first enter **F** (**file-F**), then (section 21.4.7) **p** (**save PostScript-p part+S:**. Thus **pS** would save any rectangle within the 2d snapshot, defined in section 21.4.7.

36.3 PostScript attractor basins

to write

36.4 PostScript wiring matrix

to write

36.5 PostScript wiring graphic

to write

36.6 PostScript network-graph and jump graph

to write

Intentionally Blank

Glossary

Some of the labels, acronyms, and technical terms used in the manual, in alphabetical order.

term . . . *what it means*

active cell . . . the cell position that is highlighted in the wiring graphic (section 17.3).

attractor . . . or attractor cycle, in a basin of attraction – made up of cycle of repeating states (chapter 23).

attractor basin . . . collective term used for any of the following: basin of attraction field, single basin, tree or subtree (see also "state transition graph").

asynchronous updating . . . where cells update in a predetermined or random sequence, or partial order, by the notion of time-step still applies.

basin of attraction. . . the set of states that flow to an attractor, including the attractor itself (chapter 23).

basin of attraction field . . . the basins of attraction comprising state-space (chapter 23).

block . . . a block of cells in 1d, 2d, or 3d, highlighted in the wiring graphic (section 17.3).

CA . . . Cellular Automata: a local neighborhood of $\boldsymbol{k}$ inputs (1d, 2d, 3d) and one rule (but possibly a mix of rules to extend the definition).

CA-wiring . . . same as **local wiring** below.

DDN . . . Discrete Dynamical Networks: as RBN, but allowing a value-range $\boldsymbol{v \geq 2}$. Binary CA are a special case of RBN, and RBN and multi-value CA are special cases of DDN.

effective $\boldsymbol{k}$ . . . can be smalled than actual $\boldsymbol{k}$ because of redundant wiring (section 18.9).

exhaustive algorithm . . . finding pre-images by first cereating a list of exhaustive pairs, each state in state-space and its successor. (section 29.7).

exhaustive pairs . . . each state in state-space and its successor, for the exhaustive algorithm (section 29.7).

field . . . see basin of attracion field (chapter 23).

FIELD-mode . . . to show the basin of attraction field, which does not require an initial state (the seed).

full lookup-table . . . where the lookup-table lists the output for every possible neighborhood, defined by rcode.

garden-of-Eden state . . . a state having no pre-images, also called a leaf state.

homogenious-k . . . same as **k-hom**.

history limit . . . the number of time-steps recorded when checking for the attractor (section 29.3.1).

i, j . . . the size of a 2d network, width × depth, or columns × rows.

I, J . . . the cell coordinates in a 2d i,j network.

i, j, h . . . the size of a 3d network, width × depth × height, or columns × rows × levels.

I, J, H . . . the cell coordinates in a 3d i,j,h network.

jump graph . . . a graphic representation, which can be manipulated, of the probability of jumping between basins of attraction (chapter 20.

k . . . the number of inputs to a cell, or size of the neighborhood.

k-hom . . . homogenious-k, where each cell in the network has the same number of inputs, or neighborhood size, k. Note that a homogenious-k network can be created as a k-mix network if required.

k_{Lim} . . . the maximum size of k which depends on v and is more generous if TFO-mode is active (section 7.2).

k-mix . . . or mixed-k, where each cell in the network can have a different number of inputs, or neighborhood size, k.

kcode . . . a k-totalistic rule, where the lookup-table lists the output for each combination of frequencies of values (colors) in the neighborhood, with $S = (v + k - 1)!/(k! \times (v - 1)!)$ entries. (section 13.6.1).

k_{Lim} . . . The maximum number of input wires to a cell, currently supported in DDLab – depends on TFO-mode and value-range v as listed in section 7.2.

k_{max} . . . a deliberate selection of the maximum k in a k-mix. Note that $k_{max} \leq k_{Lim}$.

leaf state . . . a state having no pre-images, also called a garden-of-Eden state.

local wiring . . . or CA-wiring, where inputs may arrive from the local neighbourhood as defined in chapter 10. Local wiring may be set up as non-local if required.

mixed-k . . . same as **k-mix** above.

n . . . the size of the network.

n_{Lim} . . . the maximum size of the network. In FIELD-mode n_{Lim} varies according to the value range v (section 7.3. In SEED-mode or TFO-mode n_{Lim}=65025 (section 8.3).

n_{exhL} . . . the maximum size of the network for the exhaustive testing algorithm (section 29.7).

neighborhood . . . the cells that are included in a given cell's update logic or transition rule. The "neighborhood" usually signifies a local CA type neighborhood of nearest (+ next nearest) neighbors in 1d, 2d or 3d (pre-defined in chapter 10).

network-graph . . . a graphic representation of the network, which can be manipulated, showing all nodes and connections (chapter 20.

non-local wiring . . . where inputs may arrive from from anywhere, but may also be biased. Local wiring may be set up as non-local.

outer-totalistic rules . . . where a different totalistic rule (tcode or kcode) applies according to the value of the center cell, so $\boldsymbol{v}$ rules. Applies to TFO-mode only, and not for mixed-$\boldsymbol{k}$. (section 13.7).

partial order updating . . . where subsets of cells update sequentially, but within each subset the updating is synchronous.

pseudo-neighborhood . . . required for non-local wiring where inputs to a cell can arrive from anywhere. The non-local inputs are wired to a notional CA (local) neighborhood (in 1d, 2d or 3d), allowing the rule to be applied to this indexed pseudo-neighborhood (section 10.2).

pre-images . . . a state's immediated predecessors.

$\boldsymbol{R}$ **. . .** the number of bytes required to encode a rule-table (section 16.16).

RBN . . . Random Boolean Networks: random wiring of $\boldsymbol{k}$ inputs (but possibly with mixed-$\boldsymbol{k}$) and a mix of rules (but possibly just one rule).

rcode . . . a full lookup table, listing the outputs for every neighborhood, with $\boldsymbol{S} = \boldsymbol{v}^{\boldsymbol{k}}$ entries.

reaction-diffusion rules . . . or excitable-media[7], where cells may be either resting, excited, or refractory, and change according to thresholds and values, resulting in waves, spirals and related patterns.(sections 13.8, 13.8.2, and 14.2.1).

rule/s . . . any kind or rule or lookup table, including rcode, kcode or tcode, as defined in chapter 13.

rule-mix . . . where each cell in the network can have a different rule (chapter 14).

$\boldsymbol{S}$ **. . .** the size of a rule-table (section 16.16).

seed . . . the initial state set in SEED-mode or TFO-mode (chapter 21.

SEED-mode . . . when not in TFO-mode, to run *forward* for space-time patterns, or generate a *single basin* of attraction, or a *subtree*, which requires an initial state or seed.

state . . . usually refers to the global state, the bit/value string, as opposed to the cell state or cell value/color.

subtree . . . part of a transient tree in a basin of attraction defined by its root (chapter 23).

state transition graph . . . the graph representing an attractor basin, but the following terms are also used for the various types of state transition graph: basin of attraction field, single basin, tree or subtree (see also "attractor basin").

synchronous updating . . . where all cells in the network update in together, in parallel.

tcode . . . a t-totalistic rule, where the lookup-table lists of the outputs of all the possible totals when the values in the neighborhood are simply added, with $\boldsymbol{S = k(v - 1) + 1}$ entries. (section 13.6.3).

TFO-mode . . . stands for "totalistic forwards only"and constrains DDLab to run just forwards to generate space-time patterns (attractor basin functions are suppressed). TFO-mode have lower constraints of the size of value-range $\boldsymbol{v}$ and nieghborhood size $\boldsymbol{k}$ (section 7.2). At the same time the rules are limited to various types of totalistic, outer-totalistic and reaction diffusion rules (no full lookup-tables – rcodes). TFO-mode requires an initial state or seed.

totalistic rules . . . rules that depend on the totals of each value, in the neighborhood, not the pattern of values. There are two main types, k-totalistic defined by kcode and t-totalistic defined by tcode.

tree . . . or transient tree, in a basin of attraction defined by its root on the attractor (chapter 23).

$\boldsymbol{V_{bits}}$ **. . .** the number of bits required to encode a rule-table output or cell value; 1 bit for $\boldsymbol{v}$=2, 2 bits for $\boldsymbol{v}$=3 or 4, 3 bits for $\boldsymbol{v}$-5 to 8 (section 16.16).

trajectory . . . the sequence of state in forward dynamics (chapter 23).

transient . . . a trajectory leading to an attractor (chapter 23).

uniform state . . . a global state (bit/value string) were all cells have the same value (color) (setion 26.1.2.

$\boldsymbol{v}$ **. . .** see "value-range".

value-range . . . the number, $\boldsymbol{v}$, of a cell's internal states (chapter 7.

wiring . . . the connections between network elements or cells. For CA the *local* wiring created the network geometry, 1d, 2d or 3d.

$\boldsymbol{v}$ **. . .** the value-range, or number of cell colors, or internal states or the size of the "alphabet".

$\boldsymbol{v2k3}$ **. . .** the notation for showing the value-range and neighborhood size, i.e. $\boldsymbol{v3k7}$ etc.

References

[NOTE] For publications by A.Wuensche refer to
http://www.cogs.susx.ac.uk/users/andywu/publications.html
where on line versions, abstracts, summaries and further details may be found.

[1] Adamatzky,A,(Ed) "Collision-Based Computing", Springer, London, 2002.

[2] Albert,R., H. Jeong, and A-L Barabasi, (2000) "Error and attack tolerance in complex networks", Nature, vol 406, July 2000.

[3] Bilotta, E., Lafusa, A. and Pantano, P. (2003). "Is self-replication an embedded characteristic of the artificial/living matter?", in Artificial Life VIII, eds. Standish and Bedau, MIT Press, 38-48.

[4] Burraston,D., and A. Martin, (2006) "Digital Behaviors and Generative Music", Special Issue, Leonardo Electronic Almanac Vol 14, No. 7–8.

[5] Conway,J.H., "What is Life?" in "Winning ways for your mathematical plays", Berlekamp,E, J.H.Conway and R.Guy, Vol.2, chap.25, Academic Press, New York, 1982.

[6] Das, R., M. Mitchell, and J. P. Crutchfield, (1994) " A Genetic Algorithm Discovers Particle-based Computation in Cellular Automata", In Y. Davidor, H.-P. Schwefel, and R. Mnner, eds., Parallel Problem Solving from Nature III, Springer-Verlag, 344– 353

[7] J. M. Greenberg, J.M., and S. P. Hastings, "Spatial patterns for discrete models of diffusion in excitable media", SIAM J. Appl. Math. 34 (1978).

[8] Gutowitz,H.A., ed.,"Cellular Automata, Theory and Experiment", MIT press, 1991.

[9] Harris,S.E., B.K.Sawhill, A.Wuensche, and S.Kauffman, "A Model of Transcriptional Regulatory Networks Based on Biases in the Observed Regulation Rules", COMPLEXITY, Vol.7/no.4, 23-40, 2002. previously:"Biased Eukaryotic Gene Regulation Rules Suggest Genome Behaviour is Near Edge of Chaos", Santa Fe Institute Working Paper 97-05-039, 1997.

[10] Hopfield,J.J. "Neural networks and physical systems with emergent collective computational abilities", Proceedings of the National Academy of Sciences 79:2554-2558, 1982.

[11] W.Hordijk, J.P.Crutchfield, M.Mitchell, "Mechanisms of Emergent Computation in Cellular Automata", In A.E. Eiben, Th. Back, M. Schienauer, and H-P. Schwefel (eds.), Parallel Problem Solving from Nature, Springer-Verlag, pp. 613-622, 1998

[12] Kauffman,S.A., "The Origins of Order, Self-Organization and Selection in Evolution", Oxford University Press, 1993.

[13] Langton,C.G., (1986) "Studying Artificial Life with Cellular Automata", Physica D, 22, 120-149.

[14] Langton,C.G., "Computation at the Edge of Chaos: Phase Transitions and Emergent Computation", Physica D, 42, 12-37, 1990.

[15] Somogyi,R., and C.Sniegoski, "Modeling the Complexity of Genetic Networks: Understanding Multigenetic and Pleiotropic Regulation", COMPLEXITY, Vol.1/No.6, 45-63, 1996.

[16] Shmulevich,I., H. Lhdesmki, E. R. Dougherty, J. Astola, W. Zhang, "The role of certain Post classes in Boolean network models of genetic networks", Proceedings of the National Academy of Sciences of the USA, Vol. 100, No. 19, 10734-10739, 2003.

[17] Wolfram,S., "Statistical Mechanics of Cellular Automata", Reviews of Modern Phisics, vol 55, 601-644, 1983.

[18] Wolfram,S., ed. "Theory and Application of Cellular Automata", World Scientific, 1986.

[19] Wuensche,A., and M.J.Lesser. (1992) "The Global Dynamics of Cellular Automata; An Atlas of Basin of Attraction Fields of One-Dimensional Cellular Automata", Santa Fe Institute Studies in the Sciences of Complexity, Addison-Wesley, Reading, MA, 1992.

[20] Wuensche,A., (1994), "The Ghost in the Machine; Basin of Attraction Fields of Random Boolean Networks", in Artificial Life III, ed C.G.Langton, Santa Fe Institute Studies in the Sciences of Complexity, Addison-Wesley, Reading, MA.

[21] Wuensche,A., (1994), "Complexity in One-D Cellular Automata; Gliders, Basins of Attraction and the Z Parameter", Santa Fe Institute Working Paper 94-04-025, 1994.

[22] Wuensche,A., (1996), "The Emergence of Memory", in "Towards a Science of Consciousness", (1996), eds. S.R.Hameroff, A.W.Kaszniak, A.C.Scott, MIT Press, 1996.

[23] Wuensche, A., (1996,) "The Emergence of Memory: Categorisation Far From Equilibrium", in Towards a Science of Consciousness: The First Tuscon Discussions and Debates, eds. Hameroff SR, Kaszniak AW and Scott AC, MIT Press, Cambridge, MA, 383–392.

[24] Wuensche,A., (1997) "Attractor Basins of Discrete Networks; Implications on self-organisation and memory", Cognitive Science Research Paper 461, Univ. of Sussex, D.Phil thesis, 1997.

[25] Wuensche,A., (1998) "Genomic Regulation Modeled as a Network with Basins of Attraction", Proceedings of the 1998 Pacific Symposium on Biocomputing, World Scientific, Singapore, 1988.

[26] Wuensche,A., (1998) "Discrete Dynamical Networks and their Attractor Basins", Proceedings of Complex Systems '98, University of New South Wales, Sydney, Australia, 1998.

[27] Wuensche,A., (1999) "Classifying Cellular Automata Automatically; Finding gliders, filtering, and relating space-time patterns, attractor basins, and the *Z* parameter", COMPLEXITY, Vol.4/no.3, 47-66, 1999.

[28] Wuensche,A., (2000) “Basins of Attraction in Cellular Automata; Order-Complexity-Chaos in Small Universes”, COMPLEXITY, Vol.5/no.6, 19-25, 2000. (based on an art exhibition in collaboration with Chris Langton)

[29] Wuensche,A., (2001) “The DDLab Manual” (for ddlabx24), `http//:www.ddlab.org`, 2001.

[30] Wuensche,A., (2002) “Basins of Attraction in Network Dynamics: A Conceptual Framework for Biomolecular Networks”, in ”Modularity in Development and Evolution”, eds G.Schlosser and G.P.Wagner. Chicago University Press 2004, chapter 13, 288-311. (Santa Fe Institute working paper 02-02-004, 2002).

[31] Wuensche.A.,(2002), “Finding Gliders in Cellular Automata”, in ”Collision-Based Computing”, ed. A.Adamatzky. Springer, London, 2002.

[32] Wuensche,A., (2002) “Discrete Dynamics Lab: Tools for investigating cellular automata and discrete dynamical networks”. Kybernetes 32, 2003.

[33] Wuensche,A., (2005) “Glider Dynamics in 3-Value Hexagonal Cellular Automata: The Beehive Rule”, int. Journ. of Unconventional Computing, Vol.1, No.4, 2005, 375-39.8

[34] Wuensche,A., A.Adamatzky, (2006), “On spiral glider-guns in hexagonal cellular automata: activator-inhibitor paradigm”, International Journal of Modern Physics C,Vol. 17, No. 7,1009-1026.

[35] Wuensche,A., (2009), ”Cellular Automata Encryption: The Reverse Algorithm, Z-Parameter and Chain-Rules”, Parallel Processing Letters (PPL), Vol 19, No 2, June 2009, 283-297.

[36] Wuensche,A., (2009), ”Discrete Dynamics Lab: Tools for investigating cellular automata and discrete dynamical networks”, ”Artificial Life Models in Software, 2nd Ed,”, eds. M.Komosinski and A.Adamatzky, chapter 8, 215-258, Springer.

Intentionally Blank

Index